2015 IEEE International Conference on Robotics and Automation (ICRA)
Washington State Convention Center
Seattle, Washington, May 26-30, 2015

Multi-scale Perception and Path Planning on Probabilistic Obstacle Maps

Florian Hauer[1] Abhijit Kundu[2] James M. Rehg[3] Panagiotis Tsiotras[4]
Georgia Institute of Technology, Atlanta, GA 30332-0150

***Abstract*— We present a path-planning algorithm that leverages a multi-scale representation of the environment. The algorithm works in n dimensions. The information of the environment is stored in a tree representing a recursive dyadic partitioning of the search space. The information used by the algorithm is the probability that a node of the tree corresponds to an obstacle in the search space. The complexity of the proposed algorithm is analyzed and its completeness is shown.**

I. Introduction

The success of any path-planning algorithm depends on the perception algorithm used to localize the agent and create the map of the environment, including the obstacles. Often, it is advantageous to hierarchically organize the collected data about the environment. This is motivated by the following two observations: First, processing all collected data at the finest resolution may be computationally prohibitive, especially for small robotic platforms with limited on-board computational resources. In addition, processing all collected information at the same temporal and spatial scale may not even be necessary for path-planning purposes, as path feasibility is primarily determined by the obstacles in the vicinity of the vehicle; far away obstacles, on the other hand, affect longer-term objectives, such as exploring the environment, reaching the goal state, etc. Second, this multi-scale hierarchy is often brought about by the perception system itself. Indeed, depending on the sensors, the collected information about the environment is rarely uniformly accurately known, and collected information is often represented by probability values modeling measurement uncertainty [21].

Different approaches have been studied for solving path-planning problems using multi-scale maps. Top-down approaches consist of finding a path at the coarsest resolution level and subsequently progressively increasing its resolution [8], [16]. Bottom-up approaches solve the problem at each node at the finest resolution level and combine the results at different resolution levels. This approach yields optimality, but requires knowing and processing the entire map at the finest resolution [13], which may be computationally expensive. Holte [6] describes an approach that includes bottom-up and top-down analysis of the data by generalizing the multi-scale problem to a multi-abstraction problem. From the finest information, an abstraction of the problem can be constructed such that the topology of the original search space is maintained in the abstraction, and the size of the problem is reduced. Repeating the process leads to a sequence of abstractions topologically similar to the original search space, but with a decreasing size of the resulting abstract search spaces. Another approach is to use information at different resolutions simultaneously. This idea is explored in [3], where areas near the current vehicle are represented accurately, while farther-away areas are coarsely-encoded by using a transformation on the wavelet coefficients. The approach is shown to be complete and very fast. A similar approach is used to create a local map in [1], but quadtrees are used instead of wavelets and only local planning is done. Other algorithms have been developed using multi-resolution maps, but they are often applied to a *given* non-uniform grid, without using the information at different resolution scales for the same region of the search space [4], [17].

In this paper we propose an extension of the algorithm introduced in [3] that is applicable for solving path-planning problems in n-dimensional search spaces. The algorithm uses an approach similar to [1] to create a local representation of the environment. We also propose a new way to incorporate the collected information that takes into account the spatial coherency of the data using a conditional random field (CRF) [12]. The algorithm directly utilizes data-structures created from state-of-the-art perception algorithms, such as octrees [7] or wavelet-coefficient trees [22], which allows the proposed path-planning algorithm to be easily incorporated in such perception algorithms, thus tightly integrating the perception and execution layers in autonomous robotic systems.

II. Problem Formulation

A. Multiresolution World Representation

The environment $\mathcal{W} \subset \mathbb{R}^d$ is assumed to be a d-dimensional grid world contained within a hypercube of side length 2^ℓ. The world $\mathcal{W}$ is not perfectly known, but an estimate of the probability label of each cell is maintained in a tree $\mathcal{T} = (\mathcal{N}, \mathcal{R})$ created from a set

[1]PhD candidate, School of Aerospace Engineering, Email:fhauer3@gatech.edu
[2]PhD candidate, College of Computing and Institute for Robotics and Intelligent Machines, Email:abhijit.kundu@gatech.edu
[3]Profesor, College of Computing and Institute for Robotics and Intelligent Machines, Email:rehg@cc.gatech.edu
[4]Professor, School of Aerospace Engineering and Institute for Robotics and Intelligent Machines, Email:tsiotras@gatech.edu
Support for this work has been provided by ARO MURI award W911NF-11-1-0046, ONR award N00014-13-1-0563 and the Intel Science and Technology Center in Embedded Computing

of measurements $\mathcal{D}$. Each $n_{k,p} \in \mathcal{N}$ has the following properties:

- It represents a hypercube $H(n_{k,p}) \subseteq \mathcal{W}$ of side length 2^k and volume 2^{dk} centered at p.
- It is at depth $\ell - k$ in $\mathcal{T}$.
- Its *children* are $n_{k-1,q_i}, i \in [1, 2^d]$ where $q_i = p + 2^{k-2}e_i$ and where e_i is each of the 2^d (d-dimensional) vectors generated by $[\pm 1, \pm 1, \ldots, \pm 1]$. The hypercubes associated with the children of the node $n_{k,p}$ induce a dyadic partitioning of $H(n_{k,p})$, as follows $H(n_{k,p}) = \bigcup_{i=1}^{2^d} H(n_{k-1,q_i})$.
- The node is a *leaf* of $\mathcal{T}$ if it has no children.
- The *value* $V(n_{k,p})$ of node $n_{k,p}$ is the probability that the label of $H(n_{k,p})$ is obstacle if it is a leaf, or the average of the values of its children if the node $n_{k,p}$ is not a leaf. That is,

$$V(n_{k,p}) = \begin{cases} P(o_{n_{k,p}} = obstacle|\mathcal{D}), & \text{if } n_{k,p} \text{ is a leaf,} \\ \frac{1}{2^d}\sum_{i=1}^{2^d} V(n_{k-1,q_i}), & \text{otherwise,} \end{cases}$$

where $o_{n_{k,p}} \in \{obstacle, free\ space\}$ is the binary occupancy label associated with $H(n_{k,p})$ and $\mathcal{D}$ is the set of measurements used to create $\mathcal{T}$.

B. The Path-Planning Problem

Two nodes of $\mathcal{T}$ are *neighbors* if their corresponding hypercubes share a face, specifically, their intersection is a hypercube of dimension $d-1$. A necessary and sufficient condition for two nodes n_{k_1,p_1} and n_{k_2,p_2} to be neighbors is that both of the following two conditions are satisfied:

a) $\|p_1 - p_2\|_\infty = 2^{k_1-1} + 2^{k_2-1}$,

b) There exists a unique $i \in [1, d]$, such that $|(p_1 - p_2)_i| = 2^{k_1-1} + 2^{k_2-1}$, where $(p_1 - p_2)_i$ is the i^{th} component of the vector.

We define a *path* $\pi = (n_{k_1,p_1}, n_{k_2,p_2}, \ldots, n_{k_N,p_N})$ in $\mathcal{N}$ to be a sequence of nodes $n_{k_i,p_i} \in \mathcal{N}$, each at corresponding position p_i and depth $\ell - k_i$ in $\mathcal{T}$, such that two consecutive nodes of the sequence are neighbors. A path is a *finest information path* (FIP) if all its nodes are leafs of $\mathcal{T}$. Note that a finest information path may contain nodes that are not of unit size. Given $\varepsilon \in [0, 1)$, a node $n_{k,p} \in \mathcal{N}$ is an *ε-obstacle* if $V(n_{k,p}) \geq 1 - 2^{-dk}\varepsilon$. A path π is *ε-feasible* if none of its nodes are ε-obstacles.

Proposition 1. *If a node in $\mathcal{T}$ is an ε-obstacle, then all leaf nodes descendant from this node are also ε-obstacles.*

Proof. Let $n_{k,p} \in \mathcal{N}$ and let n_{m_i,q_i}, where $i \in [1, L]$ be the descendant leaf nodes of $n_{k,p}$. From the definition of $V(n_{k,p})$ it can be easily shown that

$$V(n_{k,p}) = \frac{1}{2^{dk}} \sum_{i=1}^{L} 2^{dm_i} V(n_{m_i,q_i}). \tag{1}$$

Since the leaf nodes descendent from $n_{k,p}$ form a partition of $H(n_{k,p})$, it follows that $2^{dk} = \sum_{i=1}^{L} 2^{dm_i}$.

Suppose now that $n_{k,p}$ is an ε-obstacle and assume, on the contrary, that there exists n_{m_j,q_j}, $j \in [1, L]$ such that n_{m_j,q_j} is not an ε-obstacle, i.e., assume that $V(n_{m_j,q_j}) < 1 - 2^{-dm_j}\varepsilon$. Since $n_{k,p}$ is an ε-obstacle, $V(n_{k,p}) \geq 1 - 2^{-dk}\varepsilon$. It follows, from (1), that

$$V(n_{k,p}) = \frac{1}{2^{dk}} \left(\sum_{i=1, i\neq j}^{L} 2^{dm_i} V(n_{m_i,q_i}) + 2^{dm_j} V(n_{m_j,q_j}) \right)$$
$$< 1 - 2^{d(m_j-k)} + 2^{d(m_j-k)}(1 - 2^{-dm_j}\varepsilon) < 1 - 2^{-dk}\varepsilon.$$

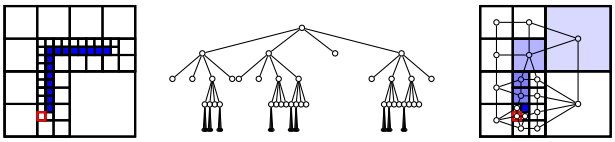
leading to a contradiction. □

Using a threshold for the ε-obstacle that depends on the size of the node extends the work in [3] to any value of $\varepsilon \in [0, 1)$ instead of only small enough ε.

III. Proposed Algorithm: The multi-scale Path Planning (MSPP) Algorithm

The overall idea of the proposed multi-scale Path Planning (MSPP) algorithm is to iteratively solve smaller problems instead of solving the original problem at once. Starting at iteration $i = 0$ from $n_{k_0,p_0} = n_{\text{start}}$, the algorithm uses $\mathcal{T}$ to create a local representation of the environment encoded in the *reduced graph* $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i)$, where $\mathcal{V}_i \subseteq \mathcal{N}$, as shown in Figure 1. The vertices of $\mathcal{G}_i$ are a collection of the nodes of $\mathcal{T}$ forming a partition of the search space, with fine resolution around the current node, say n_{k_i,p_i}, at iteration i, and with progressively coarser resolutions away from n_{k_i,p_i} (see Figure 1). The resolution levels and the horizon of each resolution level are controlled by the parameters ℓ and α (see Section IV-C). The graph $\mathcal{G}_i$ is thus a spatial representation of $\mathcal{W}$, as opposed to $\mathcal{T}$, which is a hierarchical representation of $\mathcal{W}$.

In order to avoid confusion, henceforth we will refer to the elements of $\mathcal{G}_i$ as *vertices*, and the elements of $\mathcal{T}$ as *nodes*. Every vertex corresponds to a node, but the converse is not true. The notation $\texttt{node}(v) \in \mathcal{N}$ will be used to refer to the node in $\mathcal{N}$ corresponding to vertex $v \in \mathcal{V}_i$.



(a) (b) (c)

Fig. 1: (a) Example environment $\mathcal{W}$ with the current cell, n_{k_i,p_i}, shown in red. Obstacles are drawn with solid blue color; (b) Corresponding tree $\mathcal{T}$; (c) Reduced graph $\mathcal{G}_i$ around n_{k_i,p_i} in red and corresponding space partition.

A sequence of path-planning problems are solved in each graph $\mathcal{G}_i$, $(i = 0, 1, 2, \ldots)$ as follows. The shortest path $\pi_i^{\text{goal}} = (v_{\text{start}}^i, \pi_1^i, \pi_2^i, \pi_3^i, \ldots, v_{\text{goal}}^i)$ from v_{start}^i to v_{goal}^i is found in $\mathcal{G}_i$, where v_{start}^i and v_{goal}^i are the vertices in $\mathcal{G}_i$ that are the (unique) ancestors of n_{k_i,p_i} and n_{goal} in $\mathcal{T}$, $\pi_k^i \in \mathcal{V}_i$ and $(\pi_k^i, \pi_{k+1}^i) \in \mathcal{E}_i$. Let $n_{k_{i+1},p_{i+1}} = \texttt{node}(\pi_1^i)$. Then v_{start}^{i+1} is set to $n_{k_{i+1},p_{i+1}}$ and the process is repeated until $\pi_1^i = v_{\text{goal}}^i$ for some $i \geq 0$. At iteration i, let the *partial path candidate* $\pi_{\text{start}}^i = (n_{k_0,p_0}, n_{k_1,p_1}, \ldots, n_{k_i,p_i})$ be the path constructed by the algorithm thus far. By construction (see Proposition 5),

the vertices of $\mathcal{G}_i$ neighboring n_{k_i,p_i} are necessarily leafs of $\mathcal{T}$ and hence are finest information nodes. It follows that π^i_{start} is a FIP, and hence at termination the path $\pi = (n_{k_0,p_0}, n_{k_1,p_1}, \ldots, n_{\text{goal}})$ is also a FIP.

A. Reduced Graph Construction

The vertices of $\mathcal{G}_i$ are selected recursively starting from the root of $\mathcal{T}$ using the function `GetReducedGraphVertices`. A node $n_{k,p} \in \mathcal{N}$ is selected to be included as a vertex in $\mathcal{G}_i$ if all of the following are true:

a) The node $n_{k,p}$ is a leaf, or $\|p - p_i\|_2 - \frac{\sqrt{d}}{2} 2^{k_i} > \alpha 2^k$, where p_i and $\ell - k_i$ are the position and the depth of the current node n_{k_i,p_i}, respectively, and $\alpha > 0$ is a parameter.

b) The node $n_{k,p}$ is not an ε-obstacle.

c) The node $n_{k,p}$ does not contain a part of the partial path candidate $\pi^i_{\text{start}} = (n_{k_0,p_0}, n_{k_1,p_1}, \ldots, n_{k_i,p_i})$, that is, $H(n_{k,p}) \cap H(n_{k_j,p_j}) = \emptyset,\ j = 0, 1, \ldots, i$.

When a node is not selected, its children are then considered, and the process repeats itself until all nodes of $\mathcal{T}$ have been processed. The selected nodes form a partition of the search space from which we have removed ε-obstacles (condition b)) and all nodes corresponding to the current partial path candidate since we want a loopless ε-feasible solution (condition c)). Every pair of nodes selected is tested against the neighborhood tests, and edges are created between neighboring nodes.

Function GetReducedGraphVertices

Data: Node $n_{k,p}$,Vertex list *vertices*, Current node n_{k_i,p_i}

```
if (‖p − p_i‖_2 − (√d/2) 2^{k_i} ≥ α 2^k  OR isLeaf(n_{k,p})) AND doesNotContainPath(n_{k,p}) then
    if cost(n_{k,p}) < M then
        vertices ← vertices ∪ n_{k,p}
else
    foreach n_{m,q} child of n_{k,p} do
        GetReducedGraphVertices(n_{m,q}, vertices, n_{k_i,p_i})
```

B. Backtracking Algorithm

The proposed MSPP algorithm is a backtracking algorithm [18] and is summarized in Algorithm 1. At each iteration i, the algorithm creates the reduced graph $\mathcal{G}_i$ as detailed in Section III-A. It then evokes a search to find the shortest path from the vertex v^i_{start} corresponding to the current node $n_{k_i,p_i} = \texttt{node}(v^i_{\text{start}})$ to the vertex v^i_{goal} corresponding to the goal node n_{goal}. To guarantee completeness of the MSPP algorithm, the invoked search algorithm must return a solution, if one exists. The A* algorithm [5] is used in our implementation. Once the algorithm reaches the goal, it terminates and returns the solution path stored in π^i_{start}. Note that if the algorithm backtracks when $n_{k_i,p_i} = n_{\text{start}}$ then it will have tried every possible path without finding a solution, in which case it reports failure.

Algorithm 1: The MSPP Algorithm

Data: Tree $\mathcal{T}$, Start node n_{start}, Goal node n_{goal}
Result: ε-feasible FIP from n_{start} to n_{goal} or failure

```
1  i ← 0, n_{k_i,p_i} ← n_start, π^0_start ← [n_{k_i,p_i}];
2  visits(n_{k,p}) ← ∅, ∀n_{k,p};
3  while true do
4      (G_i, v^i_start, v^i_goal) ← ReducedGraph(T, n_{k_i,p_i});
5      π^goal_i ← SP(G_i, v^i_start, v_goal,i, visits(n_{k_i,p_i}));
6      if exists(π^goal_i) then
7          n_{k_{i+1},p_{i+1}} ← node(firstElement(π^goal_i));
8          visits(n_{k_i,p_i}) ← visits(n_{k_i,p_i}) ∪ n_{k_{i+1},p_{i+1}};
9          π^{i+1}_start ← [π^i_start  n_{k_{i+1},p_{i+1}}];
10         if n_{k_{i+1},p_{i+1}} = n_goal then
11             return π^{i+1}_start;
12     else
13         visits(n_{k_i,p_i}) ← ∅;
14         π^{i+1}_start = removeLastElement(π^i_start);
15         if π^{i+1}_start = ∅ then
16             Report failure;
17         else
18             n_{k_{i+1},p_{i+1}} ← lastElement(π^{i+1}_start);
19     i ← i + 1;
20 Report failure;
```

In our implementation the cost of traversing a node is chosen as

$$\text{cost}(n_{k,p}) = 2^{dk}(\lambda_1 V(n_{k,p}) + \lambda_2) \tag{2}$$

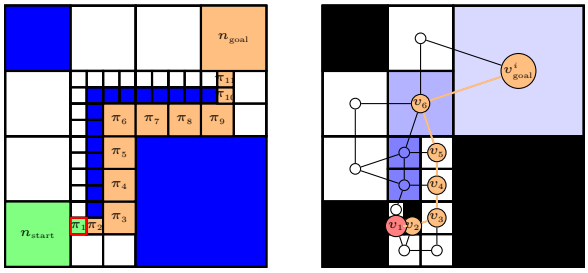
where $\lambda_1, \lambda_2 \in (0, 1]$. Other costs can be chosen depending on the problem at hand. The cost in (2) takes into account the probability that the node $n_{k,p}$ is an obstacle with weight λ_1, and it adds the length of the path with weight λ_2, scaled by the volume of the hypercube corresponding to the node.

It is clear that it is desirable to detect non-promising partial path candidates as soon as possible in order to backtrack early on, and avoid spending computational resources completing a partial path candidate that will not lead to a valid solution. The following corollary guarantees the existence of a path in $\tilde{\mathcal{G}}_i$ when there exists an ε-feasible FIP contained in the space represented by $\tilde{\mathcal{G}}_i$. In the following, $\tilde{\mathcal{R}}_i$ denotes the region in $\mathcal{W}$ represented by the vertices of $\tilde{\mathcal{G}}_i$, that is, $\tilde{\mathcal{R}}_i = \bigcup_{v_i \in \tilde{\mathcal{V}}_i} H(\texttt{node}(v_i))$.

Proposition 2. *Suppose there exists an ε-feasible FIP from n_{k_i,p_i} to n_{goal} contained in $\tilde{\mathcal{R}}_i$. Then there exists an ε-feasible path in $\tilde{\mathcal{G}}_i$ from v^i_{start} to v^i_{goal}.*

Proof. Suppose there exists an ε-feasible FIP $\pi = (\pi_1, \pi_2, \ldots, \pi_L)$ from $\pi_1 = n_{k_i,p_i} = \texttt{node}(v^i_{\text{start}})$ to $\pi_L = n_{\text{goal}}$ contained in $\tilde{\mathcal{R}}_i$. Note that, for each π_j, there exists a unique vertex $v_j \in \tilde{\mathcal{G}}_i$, such that $H(\pi_j) \subset H(\texttt{node}(v_j))$ and consider the path $(v_1, v_2, \ldots, v_L)$ in $\tilde{\mathcal{G}}_i$. Note that $v_1 = v^i_{\text{start}}$ and $v_L = v^i_{\text{goal}}$. Since the path π is ε-feasible, it follows that π_j is not an ε-obstacle. Furthermore,

π_j is a descendant of $\texttt{node}(v_j) \in \mathcal{T}$, hence by the contrapositive of Proposition 1, v_j is not an ε-obstacle. The path $(v_1, v_2, \ldots, v_L)$ is then an ε-feasible path from v^i_{start} to v^i_{goal} in $\tilde{\mathcal{G}}_i$. □



(a) (b)

Fig. 2: (a) Example environment $\mathcal{W}$ with the current cell, n_{k_i,p_i}, shown in red. Example of a ε-feasible FIP π from n_{k_i,p_i} to n_{goal} shown in orange; (b) The graph $\tilde{\mathcal{G}}_i$ and the underlying environment partition. v^i_{start} is drawn with red and orange is used for the path $(v_1, \ldots, v_L)$ in $\tilde{\mathcal{G}}_i$ corresponding to π. Note that all the nodes from π_7 to n_{goal} are mapped to the same vertex v^i_{goal}.

IV. Algorithm Properties

A. Completeness

At iteration i, let a *valid extension* of the current partial path candidate $\pi^i_{\text{start}} = (n_{k_0,p_0}, n_{k_1,p_1}, \ldots, n_{k_i,p_i})$ be an ε-feasible FIP from n_{k_i,p_i} to n_{goal} that does not have common nodes with π^i_{start}. Note that a valid extension is a FIP and hence it is a path consisting only of leafs of $\mathcal{T}$. The following proposition guarantees that any valid extension of π^i_{start} is contained in $\tilde{\mathcal{R}}_i = \bigcup_{v_i \in \tilde{\mathcal{V}}_i} H(\texttt{node}(v_i))$, where $\tilde{\mathcal{R}}_i$ is the region in $\mathcal{W}$ represented by the vertices of $\tilde{\mathcal{G}}_i$. Similarly, we will use $\mathcal{R}_i = \bigcup_{v_i \in \mathcal{V}_i} H(\texttt{node}(v_i))$ to denote the region represented by the vertices of $\mathcal{G}_i$. We will also use the notation $H(\pi^i_{\text{start}}) = \bigcup_{j=0}^{i} H(n_{k_j,p_j})$.

Proposition 3. *Let i be the current iteration number. Suppose that, for all $j = 0, 1, \ldots, i-1$, the MSPP algorithm backtracked only if there were no valid extensions of π^j_{start}. Then, any valid extension of π^i_{start} is fully contained in $\tilde{\mathcal{R}}_i$. Furthermore, if the MSPP algorithm backtracked at iteration i, there is no valid extension of π^i_{start}.*

Proposition 4. *The MSPP algorithm is complete.*

Proof. The MSPP algorithm performs an informed depth-first search on a finite tree $\mathcal{T}_{\text{path}}$, tree of ε-feasible loopless FIP starting from n_{start}, so it visits each branch of the tree $\mathcal{T}_{\text{path}}$ at most once and terminates in finite time. Suppose now that there exists an ε-feasible FIP π from n_{start} to n_{goal}. Suppose, for the sake of contradiction, that π is not found by the MSPP algorithm. It follows that either the MSPP algorithm returns a different ε-feasible FIP π' or it backtracks from n_{start} and reports failure (see Line 15 in Algorithm 1). Suppose that the MSPP algorithm backtracks from n_{start}, say, at iteration i. It then follows that $\pi^i_{\text{start}} = (n_{\text{start}})$. However, every FIP path from n_{start} has n_{start} as its first element, and hence $\pi^i_{\text{start}} \subset \pi$. The last expression implies, however, that π^i_{start} can be extended to n_{goal} using an ε-feasible FIP, namely, π. From Proposition 3 it follows that the algorithm does not backtrack from n_{start}, a contradiction. Hence the algorithm returns the ε-feasible FIP π'. □

B. Complexity

The reduced graph $\mathcal{G}_i$ keeps nodes of size 1 in a sphere of radius α centered at n_{k_i,p_i}, nodes of size 2 in a sphere of radius 2α centered at n_{k_i,p_i}, etc., and nodes of size 2^ℓ in a sphere of size $2^\ell\alpha$ centered at n_{k_i,p_i}. All these spheres contain approximately the same number of nodes S, since

$$S \approx \frac{\text{volume of the sphere of size } 2^k\alpha}{\text{volume of a node at level } k} \approx \frac{\pi^{d/2}}{\Gamma(\frac{d}{2}+1)}\alpha^d$$

is independent of k. Hence, while the search space grows exponentially, as 2^ℓ, the number of nodes of the reduced graph only grows linearly, as ℓS. Since the number of nodes per sphere grows exponentially with the dimension d, it follows that the number of nodes in $\mathcal{G}_i$ is linear in the number of levels of the tree ℓ, and exponential in the number of dimensions d, that is, $|\mathcal{V}_i| = O(\ell 2^d)$. As a comparison, to solve the same problem on a uniform grid, the graph would have $O(2^{\ell d})$ vertices.

1) Finding the reduced graph nodes: The complexity of this step depends on the number of nodes of the tree visited to find all the vertices in the reduced graph. It can be easily shown that the worst case complexity of this step is $O(|\mathcal{V}_i|)$.

2) Finding adjacency: Every pair of vertices is tested for adjacency, so $|\mathcal{V}_i|(|\mathcal{V}_i|-1)/2$ tests are made. Thus the complexity of this step is $O(|\mathcal{V}_i|^2)$.

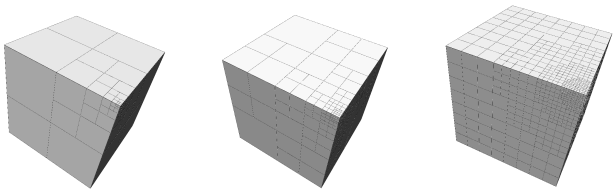
3) Finding the shortest path: If the A* algorithm is used for this step, the complexity is $O(|\mathcal{E}_i| + |\mathcal{V}_i| \log |\mathcal{V}_i|)$ but $|\mathcal{E}_i|$ is bounded by a linear function of $|\mathcal{V}_i|$ because the number of neighbors of a given hypercube is upper bounded since the resolution is finite. Hence the complexity of this step is $O(|\mathcal{V}_i| \log |\mathcal{V}_i|)$.

C. Algorithm Parameter Tuning

There are three parameters that can be tuned in the MSPP algorithm and which affect its performance: the maximum depth of the tree ℓ, the threshold α used to calculate the reduced graph, and the threshold ε.

1) Maximum number of levels of the tree ℓ: This parameter determines at which level of detail the world map is used and, subsequently, the resolution of the smallest nodes of the resulting path.

2) Decomposition parameter α: The path constructed by the algorithm should be a FIP. This is achieved by choosing only the finest information nodes to be part of the path. The following proposition gives a condition on the parameter α ensuring that the neighbors of n_{k_i,p_i} on $\mathcal{G}_i$ are leafs of $\mathcal{T}$.



(a) $\alpha = 1$ 49 nodes (b) $\alpha = 2$ 217 nodes (c) $\alpha = 5$ 2377 nodes

Fig. 3: Reduced graph for different values of α. The original space contains 287,496 nodes.

Proposition 5. *Let $\alpha \geq \sqrt{d}/2$. Then the selected nodes neighboring the current node are finest information nodes, that is, they are leafs of $\mathcal{T}$.*

Figure 3 shows the result of different values for α when the point of interest is the corner of a cube ($d = 3$).

V. Application to a Mobile Robot

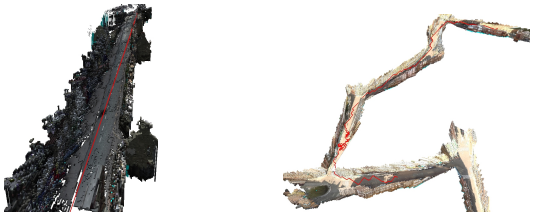
A. Maps Created From a Vision Sensor

In this section, we describe how we build a 3D multi-resolution volumetric occupancy map of an environment from camera images, similarly to [10]. Given a sequence of camera images, we first obtain a camera path and a sparse 3D reconstruction by performing visual SLAM [15], [14]. We model the 3D environment with the data structure described in Section II. However, unlike the traditional occupancy mapping work of [21], [19], we do not assume that each voxel's occupancy $o_{n_{k_j,p_j}}$ is independent of the other voxels. Instead, we form a 3D conditional random field (CRF) [20] over the voxel occupancy states $\{o_{n_{k_j,p_j}}\}$ that enforces spatial regularization over neighboring voxels as follows

$$P(\{o_{n_{k_j,p_j}}\}|\mathcal{D}) = \frac{1}{Z(\mathcal{D})} \prod_j \psi_u(o_{n_{k_j,p_j}}) \prod_{j,m\in\mathcal{N}} \psi_p(o_{n_{k_j,p_j}}, o_{n_{k_m,p_m}}), \quad (3)$$

where $Z(D)$ is the partition function over the observed data $\mathcal{D}$, and ψ_u, ψ_p are unary and pairwise potentials [20] described below. The unary potential $\psi_u(o_{n_{k_j,p_j}})$ is defined over each voxel occupancy state $o_{n_{k_j,p_j}}$. We use the sparse point cloud obtained from the visual SLAM pipeline as range measurements to update the unary terms in the same way as laser range measurements are treated in traditional occupancy mapping framework [19], [21]. In (3) $\psi_p(o_{n_{k_j,p_j}}, o_{n_{k_m,p_m}})$ is the pairwise potential, enforcing label consistency between two neighboring voxels n_{k_j,p_j} and n_{k_m,p_m} falling into a given neighborhood. We use a Potts potential [20] for ψ_p, which takes a higher value when labels are the same, and has a lower value when they are different, thus penalizing dissimilar labels across neighboring voxels. The final occupancy map is obtained by Maximum a Priori (MAP) inference [9] over the CRF in (3).

Figure 4 shows the results of the map creation and the planning at the same time applied to the CamVid [2] and Leuven [11] datasets. The Leuven dataset, specifically, demonstrates some of the potential pitfalls of having a short fine resolution horizon. In particular, the path is not smooth exhibiting several zig-zags. This is owing to the fact that if the vehicle is far from the wall, the wall will not be resolved as an obstacle until the robot comes closer. This may create a path that keeps zigzagging near the wall as the robot tries different alternatives. Nonetheless, the robot eventually finds a path to move forward.



(a) CamVid (b) Leuven

Fig. 4: Results of the planning on the maps reconstructed from the camera images

B. Real-time Application with Unknown Space Exploration

In this section, we present another application of the MSPP algorithm in which the robot is equipped with a laser range sensor and has to reach a known goal while navigating inside an a priori unknown environment. We use a real-time simulation environment which creates strict constraints on runtime. The map is built from the laser sensor measurements using Octomap [7], and it uses an incremental method that is fast enough for real time implementation. The MSPP algorithm is used to plan on the partially unknown map, and replanning is done when obstacles are detected along the current planned path.

We use the Gazebo simulation environment and the Robotic Operating System (ROS) to communicate between the different modules of the simulator. The simulator provides the ground truth representation of the world and integrates the dynamics of the robot based on the received commands. Noisy laser measurements are generated at 2Hz and the pose of the robot is sent at 100Hz. The Octomap server creates the tree $\mathcal{T}$ using the measurements received, and sends the new map to the planner. The planner checks whether the current planned path is ε-feasible. If it is not ε-feasible it replans until it finds one, and sends the solution to the path tracker as a set of waypoints. Finally, the waypoint tracker generates the motor commands from the current robot pose and the computed waypoints.

The world used for the simulation is shown in Figure 5. The robot starts at the center and the goal is the red ball. The path is recalculated when a new goal is assigned, or the current planned path is not ε-feasible any more on the updated map. The computed path is subsequently smoothed and is fed to the trajectory tracking module. Figure 6 shows the result of the path smoothing.

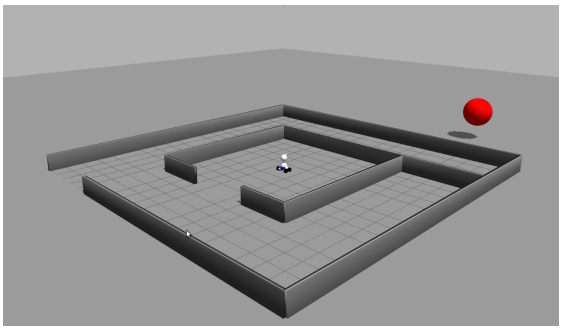
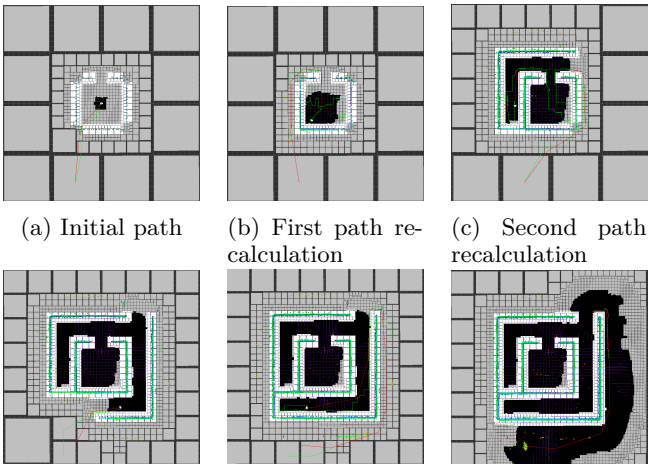


Fig. 5: Maze used for the simulation, the red ball represents the goal. The robot starts at the center of the maze.

Figure 6 depicts some key frames of the results[1]. The benefits of using a multi-scale representation can be seen since the unknown space is represented by nodes encoding large regions of the environment, thus reducing the size of the data kept in memory. The colored cubes correspond to points measured by the laser sensor, and clearly show the walls.



(a) Initial path (b) First path recalculation (c) Second path recalculation

(d) Third path recalculation (e) Fourth path recalculation (f) Goal reached

Fig. 6: Results of the mapping and planning simulation.

VI. Conclusions

In this paper, we present a multi-scale perception and path-planning algorithm. The algorithm is shown to be complete. The complexity of the algorithm is shown to grow only linearly, while the size of the map grows exponentially. This allows the algorithm to be implemented on large maps without excessive execution times. The proposed algorithm seamlessly integrates multi-scale perception with multi-scale path planning. The results of the algorithm are applied on 2D and 3D maps created from a realistic multi-scale perception algorithm involving a mobile robot navigating in an unknown environment.

[1]A video of the results can be found in the multimedia attachments.

References

[1] S. Behnke. Local multiresolution path planning. In *Robocup 2003: Robot Soccer World Cup VII*, pages 332–343. Springer, 2004.

[2] G. Brostow, J. Fauqueur, and R. Cipolla. Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters*, 30(2):88–97, 2009.

[3] R. V. Cowlagi. *Hierarchical Motion Planning for Autonomous Aerial and Terrestrial Vehicles*. PhD thesis, Georgia Institute of Technology - School of Aerospace Engineering, 2011.

[4] D. Ferguson and A. Stentz. Using interpolation to improve path planning: The Field D* algorithm. *Journal of Field Robotics*, 23(2):79–101, 2006.

[5] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[6] R. C. Holte, T. Mkadmi, R. M. Zimmer, and A. J. MacDonald. Speeding up problem solving by abstraction: A graph oriented approach. *Artificial Intelligence*, 85(1):321–361, 1996.

[7] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, 2013.

[8] S. Kambhampati and L. S. Davis. Multiresolution path planning for mobile robots. *IEEE Journal of Robotics and Automation*, 2(3):135–145, 1986.

[9] J. H. Kappes, M. Speth, G. Reinelt, and C. Schnorr. Towards efficient and exact map-inference for large scale discrete computer vision problems via combinatorial optimization. In *Computer Vision and Pattern Recognition*, Portland, OR, USA, June 25–27 2013.

[10] A. Kundu, Y. Li, F. Dellaert, F. Li, and J. M. Rehg. Joint Semantic Segmentation and 3D Reconstruction from Monocular Video. In *European Conference on Computer Vision*, Zurich, Switzerland, September 6–12, 2014.

[11] L. Ladicky, P. Sturgess, C. Russell, S. Sengupta, Y. Bastanlar, W. Clocksin, and P. H. Torr. Joint optimisation for object class segmentation and dense stereo reconstruction. In *British Machine Vision Conference*, Aberystwyth, Wales, August 31 – September 3 2010.

[12] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *International Conference on Machine Learning*, Williamstown, MA, USA, June 28 – July 1 2001.

[13] Y. Lu, Y. Huo, and P. Tsiotras. An accelerated path-finding algorithm using multiscale information. *IEEE Transactions on Automatic Control*, 57(5):1166–1178, May 2012.

[14] R. Newcombe and A. Davison. Live dense reconstruction with a single moving camera. In *Computer Vision and Pattern Recognition*, San Francisco, CA, USA, June 13–18 2010.

[15] D. Nister, O. Naroditsky, and J. Bergen. Visual odometry. In *Computer Vision and Pattern Recognition*, Washington, DC, USA, June 27– July 2 2004.

[16] D. K. Pai and L.-M. Reissell. Multiresolution rough terrain motion planning. *IEEE Transactions on Robotics and Automation*, 14(1):19–33, 1998.

[17] C. Petres, Y. Pailhas, P. Patron, Y. Petillot, J. Evans, and D. Lane. Path planning for autonomous underwater vehicles. *IEEE Transactions on Robotics*, 23(2):331–341, 2007.

[18] S. Sahni. *Data Structures, Algorithms, and Applications in C++*, pages 751–786. New York: WCB McGraw-Hill, 1998.

[19] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2005.

[20] C. Wang, N. Komodakis, and N. Paragios. Markov random field modeling, inference and learning in computer vision and image understanding: A survey. *Computer Vision and Image Understanding*, 117(11):1610 – 1627, 2013.

[21] K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems. In *ICRA 2010 Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation*, Anchorage, Alaska, USA, May 3–8 2010.

[22] M. Yguel, O. Aycard, and C. Laugier. Wavelet occupancy grids: a method for compact map building. In *Field and Service Robotics*, pages 219–230. Springer, 2006.