2012 IEEE International Conference on Robotics and Automation
RiverCentre, Saint Paul, Minnesota, USA
May 14-18, 2012

Combining Global and Local Planning with Guarantees on Completeness

Haojie Zhang[†], Jonathan Butzke[‡], Maxim Likhachev[‡]

Abstract— **Planning with kinodynamic constraints is often required for mobile robots operating in cluttered, complex environments. A common approach is to use a two-dimensional (2-D) global planner for long range planning, and a short range higher dimensional planner or controller capable of satisfying all of the constraints on motion. However, this approach is incomplete and can result in oscillations and the inability to find a path to the goal. In this paper we present an approach to solving this problem by combining the global and local path planning problem into a single search using a combined 2-D and higher dimensional state-space.**

I. INTRODUCTION

Mobile robots often have to operate in large complex environments. As such, path planning for these systems needs to account for the various kinodynamic constraints of the platform and the environment potentially resulting in a high dimensional state-space. Unfortunately, this high dimensionality often leads to a dramatic increase in the time and memory required to find a path as the environment size increases. For a sufficiently large outdoor environment it can become computationally intractable for the robot's onboard processing capability. Planning only in two dimensions (2-D), such as planar position (x, y), does not suffice as the resulting path may not be feasible due to motion constraints or asymmetric platform shapes. As a result, planning in a high dimensional (high-D) state-space is often necessary in order to guarantee executable paths. For example the 2-D (x, y) path shown in Fig. 1(a) is difficult for a non-holonomic robot to follow due to the instantaneous change in the orientation of the robot. On the contrary the 3-D (x, y, θ) path shown in Fig. 1(b) is relatively easy to execute.

Current approaches to full-dimensional planning for navigation are either suboptimal [1], [2] or are limited in the size of the area they can handle [3]. For very large environments, a common alternative is to perform a global plan in 2-D and have a separate local controller or local planner perform a higher dimensional plan on a small local region around the robot [4], [5], [6]. While effectively ignoring a subset of the dimensions for the global plan can make these large environments tractable and improve planning times, they are incomplete in that the robot may get stuck and never reach the goal even though there is a feasible path. The main reason for this is the inconsistency between the assumptions that local and global planners make.

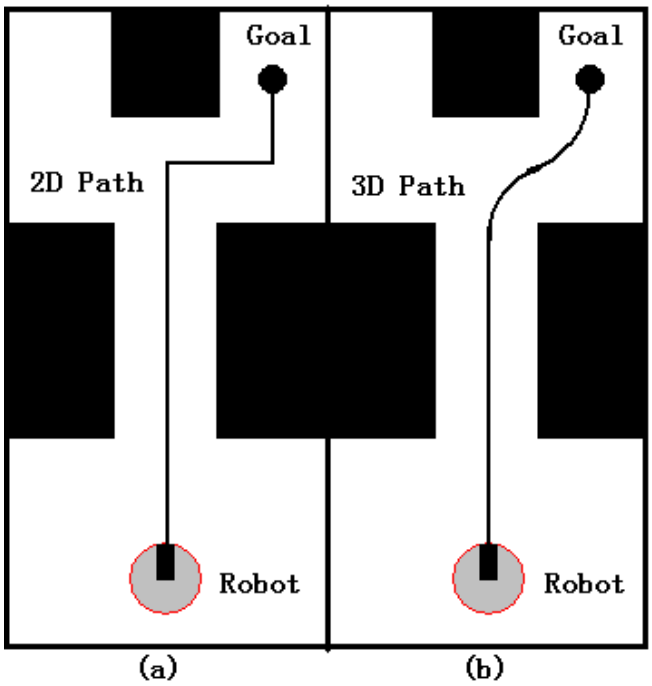


Fig. 1: (a) Unfeasible 2D path for a non-holonomic robot, (b) feasible 3D path for a non-holonomic robot

As an example, Fig. 2 depicts a scenario that many planners are incapable of successfully dealing with. In this scenario, a short path to the goal is available but due to the sharp turn initially being located outside of the range of the local high dimensional planner, it is infeasible unbeknownst to the robot. As the robot approaches the impassable area, the sharp turn enters the range of the local controller. At this point the system recognizes that the current path is no longer executable, replans, and produces a path that goes down the right hand route. However, once the robot leaves the vicinity of the obstruction such that the problem area is no longer within the planning area of the local planner, any replan by the global planner will once again attempt to route the robot down the left hand side. For some environments numerous such routes may exist resulting in a multi-state oscillation where the robot in turn tries each of them without ever producing a plan through the one correct route. A similar situation may occur in outdoor environments where states may be feasible but have high cost depending on which

This work was partially supported by the Department of Defense (DoD) through the National Defense Science & Engineering Graduate Fellowship (NDSEG) Program. This research has also been in part supported by the ONR DR-IRIS MURI project grant N00014-09-1-1052 and ONR ANTIDOTE MURI project grant N00014-09-1-1031.

[†]Intelligent Vehicle Research Center, School of Mechanical Engineering, Beijing Institute of Technology, Beijing 100081, CHINA haojie.bit@gmail.com

[‡]Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA jbutzke@andrew.cmu.edu, maxim@cs.cmu.edu

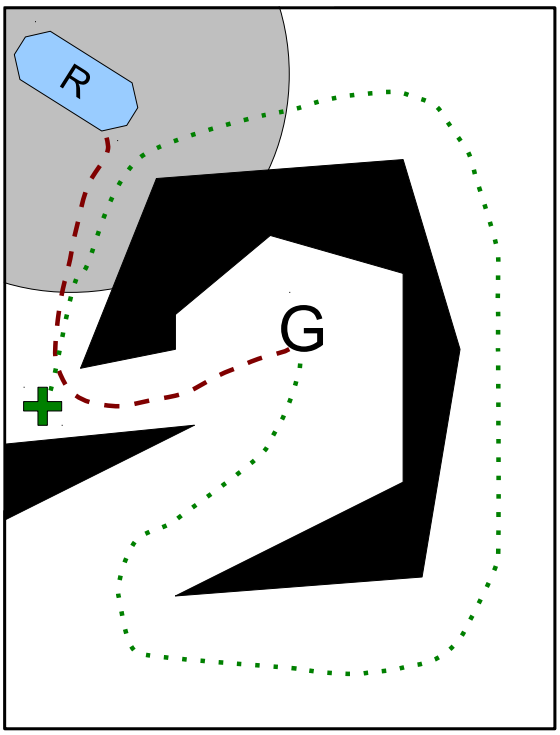


Fig. 2: Example of environment that could be incomplete for systems with separate local and global planners. Robot R is at the start location in the upper left. The local planner operates within the grey circle surrounding the robot. The global planner initially generates red dashed path to the goal G. However the robots size prevents it from making the turn marked by the green "+". At this point the global planner returns the green dotted path. Once the robot returns to the vicinity of the start location, any replan by the global planner would return a path similar to the red dashed path, potentially resulting in the robot oscillating between the two points.

direction the robot is oriented. An example would be a steep slope. While traversing along the side of the slope the cost would be relatively low, however going straight up or down would be considerably more expensive. The naïve approach to solving these problems by blocking off the offending path also results in an incomplete planning solution as the robot may need to partially enter the area in order to maneuver for the correct path, or in the slope example, may be required to pass through a position with a certain orientation.

In this paper we show how to extend the concept of the local and global planners to a single graph in such a way that we can guarantee completeness while being efficient enough for use as an online planner. Using ideas from adaptive dimensionality reduction [7] we can iteratively combine the 2-D global graph along with one or more higher dimensional areas. By selecting the areas in the vicinity of the robot and at key points in the environment for inclusion as high-dimensional areas we can improve upon the separate global and local planner paradigm. This approach relies on the notion that only certain areas need or even benefit from the higher dimensional planning while the vast majority of states only need to be searched in the lower dimension. This results in substantial speedups and lower memory requirements while preventing oscillations. We show that the algorithm is complete with respect to the state-space discretization and can provably guarantee to find a solution, if one exists, provided its actions are reversible (e.g. no one way streets). We also demonstrate the efficiency of the algorithm in simulation and applied to a real-world significantly asymmetric differential drive robot.

II. ALGORITHM

A. Description

The algorithm functions by forming a graph of the search space with a mix of 2-D and high-D states. It starts out by planning in high-D around the robot and everywhere else in 2-D (just as a standard combination of global and local planning). Then as the robot starts executing the trajectory, every time it encounters a situation where the re-planning finds a path that differs from the previous path or it recognizes the potential for oscillations to occur, the planner leaves a permanent high-D region at the current location. For our purposes we use 2-D synonymously with the low dimensional component of our algorithm, specifically using it to represent the planar position (x, y). The high-D state-space is typically translation and orientation (x, y, z, θ) although velocity v or other dimensions could be used as well.

B. Notations and Assumptions

We assume that the planning problem is represented as searching a directed graph $\mathcal{G} = (\mathcal{S}_d, \mathcal{E}_d)$ where $\mathcal{S}_d$ is a discretized finite state-space with dimensionality d, consisting of vertices $s = (x_1, x_2, \ldots, x_d)$ and $\mathcal{E}_d$ is the set of directed edges in the graph. The transition between vertices $e(s_i, s_j)$ is associated with a cost $c(s_i, s_j)$. The objective of the search is to find a least-cost path in $\mathcal{G}$ from start state s_{start} to goal state s_{goal}. We use the notation $\pi(s_i, s_j)$ to denote a path in graph $\mathcal{G}$ from state s_i to state s_j and $\pi^\star(s_i, s_j)$ to denote the least-cost path.

Definition 1 ($\mathcal{S}_H$, $\mathcal{S}_L$): We define two auxiliary state-spaces that we use to construct our search state-space, a high dimensional state-space $\mathcal{S}_H$ with dimensionality h and a low dimensional state-space $\mathcal{S}_L$ with dimensionality l such that $h > l$.

Definition 2 ($\lambda(\cdot)$): Function returning the projection of a state into a lower-dimensional subspace. There also exists the inverse λ^{-1} which maps from a lower to a higher-dimensional space:

$$\lambda(s^h) = s^l \tag{1}$$

$$\lambda^{-1}(s^l) = \{s^h\} \tag{2}$$

$$\lambda(\lambda^{-1}(s^l)) = s^l \tag{3}$$

$$\lambda^{-1}(\lambda(s^h)) \neq s^h \tag{4}$$

Notice that λ is a many to one mapping and λ^{-1} is a one to many mapping. Also note that the left-hand side of (4) results in a set of states not the single state provided as input.

Definition 3 ($\mathbf{Q_c}$, $\mathbf{Q_1}$): $\mathbf{Q_c}$ is the queue that stores the position of all high dimensional regions that have been introduced by the algorithm. Each high-D region can be represented by a single state. For example, a high-D sphere where the center state is stored in $\mathbf{Q_c}$. $\mathbf{Q_1}$ is a queue that stores potential centers of high-D regions and their corresponding lower bounds on cost-to-goal (g_{low}).

Definition 4 (Priority): For every high-dimensional state s, the priority of s according to the i^{th} replan is:

$$P^i(s) = \begin{cases} 2 & \text{if } s \in \mathbf{Q_c} \\ 1 & \text{if } s \in \mathbf{Q_1} \\ 0 & \text{otherwise} \end{cases}$$

Definition 5 (π^i): The i^{th} plan returned from COMPUTEPATH. It consists of the ordered list of states $\{s_0^i, s_1^i, s_2^i, \ldots, s_n^i\}$ where $s_n^i = s_{goal}$ and $s_0^i = s_{curr}$ at the time when COMPUTEPATH was executed. We assume COMPUTEPATH executes a backward $A^\star$ search or one of its variants and as such it computes g-values which are the cost-to-goal for some states in the state-space. It is guaranteed to have been computed for all states on the returned path. The search is performed from the goal to the start state so $g^i(s_j^i) > g^i(s_{j+1}^i) \quad \forall j \colon 0 \ldots n-1$.

Definition 6 (Condition 1 (C1)): is a planner specific test for inserting high-D regions into the hybrid graph. For our implementation we use a change in homotopic class (Def. 7) to signal this condition. Our planner is guaranteed to be complete independent of what "Condition 1" stipulates or even if it is absent.

Definition 7 (Homotopic Class): Two paths connecting the same start and goal coordinate are in the same homotopy class if they can be smoothly deformed into one another without intersecting any obstacle in the environment, otherwise they are in different homotopy class. The homotopic class of a path is dependent on which obstacles it went around and in which direction. In this way paths that take fundamentally different routes to the goal can be identified[1] (see [8] for in-depth mathematical treatment).

Assumption 1: The cost of transition between different high dimensional states is at least the cost of transition between the corresponding low dimensional states. For every pair of high-dimensional state s_i and s_j, it will hold that $c(s_i, s_j) \geq c^*(\lambda(s_i), \lambda(s_j))$. We further stipulate that all transitions are dynamically feasible. By only using feasible edges, the resulting trajectory through the graph is guaranteed to be executable without additional smoothing or post-processing. For 3-D navigation typical motion primitive sets

[1]Homotopic classes can be thought of like a road system. If one were to take Highway 1 from A $\rightarrow$ B, your path may vary depending on which lane you travel in, however, your homotopic class would remain the same since all lanes would travel on the same side and in the same direction around obstacles. If you were to take Highway 2 instead, your homotopic class would be different since it would not pass the obstacles in the same way.

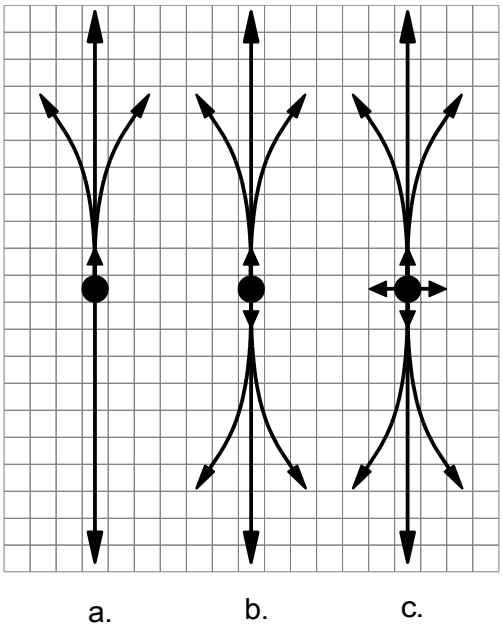


Fig. 3: Examples of motion primitives used to determine allowable edges in the search graph. (a) Non-holonomic robot capable of turning only while moving forwards. (b) A non-holonomic robot capable of turning in both directions. (c) An omni-directional robot capable of strafing sideways.

are depicted in Fig. 3

Assumption 2: The map only changes a finite number of times. This assumption is required for completeness, as a map could be modified in an adversarial way to ensure no plan ever successfully reaches the goal state.

Assumption 3: $\forall s \colon e(s_{curr}, s) \in \mathcal{E}_{HL} \Longrightarrow s \in \mathcal{S}_H$.

To maintain the completeness property we assume that the high-D region around the robot is larger than the length of the longest high-D motion primitive. This ensures that the immediate successor of a robot state, s_{curr}, is a high-D state. For the sake of efficiency in execution we assume that the cost of turning through an angle of α is no greater than the cost to drive along an arc of length $r\alpha$, where r is the radius of the high-D circle.

C. Search Graph Construction

We construct a hybrid dimensional state space $\mathcal{S}_{HL}$ as our search state-space by combining the high dimensional state-space $\mathcal{S}_H$ and low dimensional state-space $\mathcal{S}_L$. The high dimensional state space in $\mathcal{S}_{HL}$ consists of $1 \ldots N$ small circular regions R_n, centered at key points (x_n, y_n) with one of these points always being the current robot position. For each high dimensional region, $\mathcal{S}_{HL}$ is modified by removing all low dimensional states s^l inside the high dimensional region R_n, replacing them with the set of high dimensional projections $\lambda^{-1}(s^l)$. Thus, $\mathcal{S}_{HL}$ contains both low dimensional and high dimensional states. Also, by construction

$$s^h \in \mathcal{S}_{HL} \Leftrightarrow \lambda(s^h) \notin \mathcal{S}_{HL} \tag{5}$$

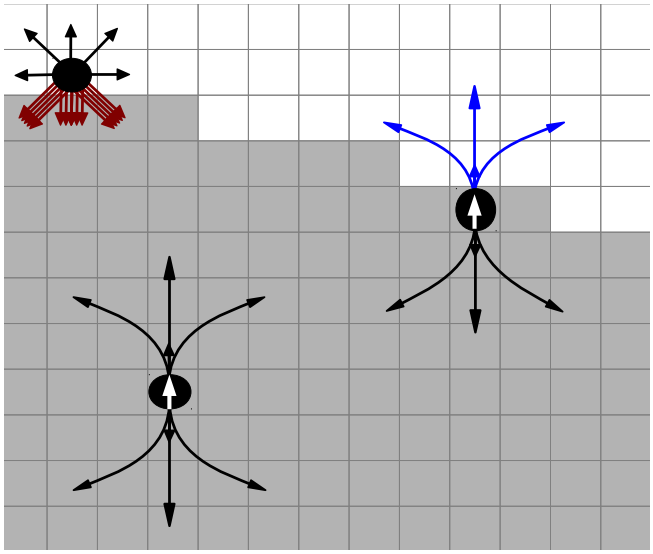


Fig. 4: Example of $\mathcal{E}_{HL}$ construction for a 3-D/2-D environment. The states in the light grey area are 3-D while the white states are 2-D. The state in the lower left is completely in the 3-D region, and thus all of its possible edges are part of $\mathcal{E}_{HL}$. The 2-D state in the upper left contributes its 2-D edges to other 2-D states, and an edge to each element of the set $\lambda^{-1}(s_j^l)$ depicted in red. Similarly, the 3-D state on the right has edges to the 2-D state $\lambda(s_j^h)$ shown in blue.

a high dimensional state s^h and its corresponding low dimensional projection $\lambda(s^h)$ are never simultaneously members of $\mathcal{S}_{HL}$ although one of them always is.

In order to transition between the high dimensional regions and the low dimensional regions, the hybrid transition set $\mathcal{E}_{HL}$ is defined for $\mathcal{S}_{HL}$. The hybrid transition set is constructed as the union of the following edges:

$$\forall i,j : e(s_i^h, s_j^h) \in \mathcal{E}_H, s_i^h \in \mathcal{S}_{HL} \wedge s_j^h \in \mathcal{S}_{HL} \Rightarrow e(s_i^h, s_j^h) \in \mathcal{E}_{HL} \quad (6)$$

$$\forall i,j : e(s_i^l, s_j^l) \in \mathcal{E}_L, s_i^l \in \mathcal{S}_{HL} \wedge s_j^l \in \mathcal{S}_{HL} \Rightarrow e(s_i^l, s_j^l) \in \mathcal{E}_{HL} \quad (7)$$

$$\forall i,j : e(s_i^h, s_j^h) \in \mathcal{E}_H, s_i^h \in \mathcal{S}_{HL} \wedge \lambda(s_j^h) \in \mathcal{S}_{HL} \Rightarrow e(s_i^h, \lambda(s_j^h)) \in \mathcal{E}_{HL} \quad (8)$$

$$\forall i,j : e(s_i^l, s_j^l) \in \mathcal{E}_L, s_i^l \in \mathcal{S}_{HL} \wedge s_j^h \in \mathcal{S}_H \wedge s_j^h \in \lambda^{-1}(s_j^l) \Rightarrow e(s_i^l, s_j^h) \in \mathcal{E}_{HL} \quad (9)$$

Notice that the above definition of $\mathcal{E}_{HL}$ allows for transitions between states of different dimensionalities. Fig. 4 illustrates the set of transitions in the $\mathcal{G}_{HL}$ graph in the case of 3D (x, y, θ) path planning and Algorithm 2 in Section II-D below details the pseudo-code.

Algorithm 1 SEARCH($\mathcal{G}, s_{start}, s_{goal}$)

1: $\mathbf{Q_c} = \{\varnothing\}, \mathbf{Q_1} = \{\varnothing\}$, $g_{low}(s) = 0, g^0(s) = \infty \text{for} \forall s$, $s_{prev} = s_{curr} = s_{start}, i = 0$
2: **while** $s_{curr} \neq s_{goal}$ **do**
3: update map with sensor data
4: $i = i + 1$
5: $[\pi^i, g^i, \text{C1}] = \text{COMPUTEPATH}(s_{curr}, s_{goal}, \text{CONSTRUCT}(\mathcal{G}, \{\mathbf{Q_c} \cap s_{curr}\}))$
6: **if** C1 is satisfied **then**
7: insert s_{curr} in $\mathbf{Q_c}$
8: **end if**
9: **if** $\exists s\colon (s \in \pi^i) \wedge (s \in \mathbf{Q_1}) \wedge (g^i(s) < g_{low}(s))$ **then**
10: move s from $\mathbf{Q_1}$ to $\mathbf{Q_c}$
11: **end if**
12: **if** $g^i(s_{curr}) \geq g^{i-1}(s_{prev})$ **then**
13: $g_{low}(s_{curr})$=max$(g^i(s_{curr}), g_{low}(s_{curr}))$
14: insert/update s_{curr} in $\mathbf{Q_1}$ with $g_{low}(s_{curr})$
15: **end if**
16: $s_{prev} = s_{curr}$
17: $s_{curr} = \pi^i(1)$
18: **end while**

D. Graph Search

The variable dimensional search algorithm is detailed in Algorithm 1. The variables specific to this algorithm are initialized on line 1 prior to the first search. COMPUTEPATH on line 5 takes the current position and goal along with the set of all high dimensional regions and runs backward $A^\star$ or one of its variants on the hybrid graph to determine the optimal path to the goal. It returns this path, as well as updated g-values[2] for all states on the returned path. The third return value is a variable used to denote whether the trajectory returned meets the conditions of "Condition 1" (Def. 6). Note that the current position is always passed into COMPUTEPATH as a location for a high-D region.

The hybrid graph consisting of the hybrid vertex and edge elements, $\mathcal{S}_{HL}$ and $\mathcal{E}_{HL}$ is constructed as detailed in Section II-C and shown in Algorithm 2 and passed into the COMPUTEPATH function (line 5 of Algorithm 1). It initially sets the hybrid state-space to be equal to the low dimensional state-space. For each of the high-D regions it adds the appropriate high-D states and corresponding edges to the graph (lines 5-7). The REGION function on line 5 is also responsible for removing the corresponding low dimensional states from $\mathcal{S}_{HL}$. The ADDEDGES function generates $\mathcal{E}_{HL}$ as described in section Section II-C above.

In the event that "Condition 1" is satisfied (as detected by line 6 in Algorithm 1) then the current location is added to the set of high-D states. This is done because "Condition 1" should signal when new information, such as the inclusion of the higher dimensions in the search, has shown the current trajectory to not be optimal. By placing the current

[2]Note: only the i and $i-1$ g-values must be stored, older ones can be discarded.

Algorithm 2 CONSTRUCT($\mathcal{G}, \mathbf{Q}$)

1: $\mathcal{S}_{HL} = \mathcal{S}_L$
2: $\mathcal{E}_{HL}$ =NULL
3: $\mathcal{G}_{HL} = (\mathcal{S}_{HL}, \mathcal{E}_{HL})$
4: **for all** $\langle s^l \rangle \in \mathbf{Q}$ **do**
5: add REGION($\langle s^l \rangle$) to $\mathcal{S}_{HL}$
6: **end for**
7: $\mathcal{E}_{HL}$ =ADDEDGES($\mathcal{S}_{HL}$)
8: **return** $(\mathcal{S}_{HL}, \mathcal{E}_{HL})$

location into the list of high-D states, that information will be "remembered" during future planning cycles.

States that had previously been priority 1 and are on the current trajectory to the goal and now have a lower g-value than the g_{low} stored for that state are also added to the list of high-D states (lines 9-11). This is done in order to prevent oscillations as a drop in a g-value indicates a fundamental change in the path taken from that location to the goal, even if "Condition 1" was not true. As an example, consider a long, narrow hallway with a robot incapable of turning in place, a high penalty for traveling backwards and a goal state behind the robot. If the robot cannot sense the end of the hallway, it may choose to travel forward to the unseen area and turn around there. However, once reaching the end of the hallway, the robot will determine that it is unable to turn-around, and will travel in reverse back to the goal state. As it does this, it is undesirable for the robot to return to the end of the hallway once it leaves the sensor range. However, we can detect this phenomenon by the lowering of the g-values during the back-tracking portion and place high-D regions along the hallway. In this way, the robot is guaranteed to not oscillate.

The check on line 12 is to flag potential oscillation points along the path. As these points are elevated to priority 1, they become eligible for inclusion in the high-D areas as described in the above paragraph. This check is true when the g-value of the current robot pose is higher than the previously computed g-value of its previous pose. Since the search is made outwards from the goal, the values should be monotonically decreasing, thus any increase signals a potential oscillatory region.

Finally the plan is passed to the drive control subsystem to conduct the actual move. After a (possibly zero) delay, the position is updated and the loop repeats (line 17).

E. Theoretical Analysis

The algorithm holds several theoretical properties due to the construction of the search graph. In the following we assume N is the number of high-D states in the original search graph, $\mathcal{G}_H$.

Theorem 1: At the conclusion of every iteration i of lines 5-15 for $i > 1$, one of the following conditions holds prior to the execution of line 16:

- **case i** $g^{i-1}(s_{prev}) > g^i(s_{curr})$
- **case ii** $\exists$ high-dimensional state s such that $P^{i-1}(s) < P^i(s)$
- **case iii** $\exists j < i$ such that $\|\mathbf{Q_c}^j\| < \|\mathbf{Q_c}^i\|$ and $s^j_{curr} = s_{curr}$and for$\forall k = j+1, \ldots, i-1, s^k_{curr} \neq s_{curr}$, where s^j_{curr} denotes the value of s_{curr} right before line 16 is executed during the j^{th} iteration.

Proof sketch: Assume **case i** does not hold then $g^{i-1}(s_{prev}) \leq g^i(s_{curr})$.
If $P^{i-1}(s_{curr}) = 0$, by lines 12-15, s_{curr} will be moved to Q_1. As a result, $P^i(s_{curr}) = 1 > P^{i-1}(s_{curr})$ and **case ii** will hold.
If $P^{i-1}(s_{curr}) = 1$, it indicates s_{curr} has been the start state at least once before since only start states are inserted into $\mathbf{Q_1}$ (line 14). Let j be the most recent iteration that started from s_{curr} and let $g^j(s_{curr})$ be the corresponding g-value. By line 13 $g_{low}(s_{curr}) \geq g^j(s_{curr})$. Between the j^{th} and i^{th} plan, one of two cases must be true: $\exists$ s_i such that s_i is inserted in $\mathbf{Q_c}$ or not.

- If s_i is inserted in $\mathbf{Q_c}$, **case iii** will hold since the algorithm never removes a state from $\mathbf{Q_c}$.
- If $\nexists s_i$ inserted in $\mathbf{Q_c}$ the hybrid graph did not change and $g^i(s_{curr}) = g^j(s_{curr})$. Since $g^i(s_{curr}) \geq g^{i-1}(s_{prev}) > g^{i-1}(s_{curr})$ because $s_{prev} = s_0^{i-1}, s_{curr} = s_1^{i-1}$ then due to Assumption 1 $g^{i-1}(s_0^{i-1}) > g^{i-1}(s_1^{i-1})$ from Def. 5, then $g_{low}(s_{curr}) > g^{i-1}(s_{curr})$ because $g_{low}(s_{curr}) \geq g^j(s_{curr}) = g^i(s_{curr})$. In which case line 9 will be true (prior to g_{low} being updated on line 13) and s_{curr} will be moved from $\mathbf{Q_1}$ to $\mathbf{Q_c}$ which is a contradiction to assuming $\nexists s_i$ inserted in $\mathbf{Q_c}$.

If $P^{i-1}(s_{curr}) = 2$, it indicates that s_{curr} is already the center of the high dimensional space. Also $g^{i-1}(s_{prev}) > g^{i-1}(s_{curr})$ because of Def. 5. Between line 5 of the $i-1^{th}$ and the i^{th} iteration either a state s was added to $\mathbf{Q_c}$ or $\mathbf{Q_1}$ in which case $P^{i-1}(s) < P^i(s)$ and **case ii** holds or not. If not, then the COMPUTEPATH function was called with $\{\mathbf{Q_c} \cap s_{prev}\}$ on the $i-1^{th}$ iteration and $\{\mathbf{Q_c}\}$ on the i^{th} (since s_{curr} is already in $\mathbf{Q_c}$). Since there are no additional high dimensional states being searched (and possibly less) $g^{i-1}(s_{prev}) > g^{i-1}(s_{curr}) \geq g^i(s_{curr})$, which contradicts the assumption that **case i** does not hold.

Lemma 1: On a finite graph, **case i** can only occur a maximum $O(N)$ times before **case ii** or **case iii** happen.

Proof sketch: Since the g-values are finite, and, without loss of generality, decrease by integral amounts along the path, this leaves a limited number of moves before the robot reaches its goal or **case ii** or **case iii** happen.

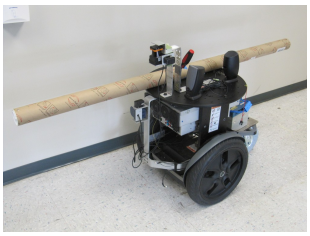
Lemma 2: On a finite graph, **case ii** can only occur a maximum $O(2N)$ times

Proof sketch: When **case ii** occurs at least one state whose priority is increasing. Since each state can increase its priority twice and the priority never decreases, **case ii** can only occur at most $O(2N)$ times.

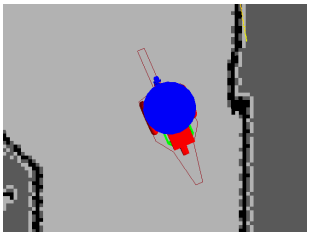
Lemma 3: On a finite graph, **case iii** can only occur a maximum $O(N)$ times

Proof sketch: When **case iii** occurs at least one state has been added to $\mathbf{Q_c}$ which can happen N times.

Theorem 2: On a finite graph, the robot is guaranteed to



(a) Melvin the Segbot



(b) Footprint

Fig. 5: a) Photo of the robot used for the experiments showing asymmetric footprint. b) The planners representation of the robot as seen from above.

reach its goal if any state reachable from the start pose has a feasible path to the goal pose.

Proof sketch: Since for every iteration one of **case i** , **case ii** , **case iii** will occur, there is a bounded number of times **case i** may occur prior to either **case ii** or **case iii** occurring and **case ii** and **case iii** are both themselves bounded in their number of occurrences, then the algorithm will terminate on a finite graph with a solution if one exists. Therefore, the algorithm is complete on finite graphs.

III. EXPERIMENTAL RESULTS

We initially tested our algorithm using a simulation environment similar to that shown in a very simplified form in Fig. 6. The robot model was a differentially driven robot with a significantly asymmetric footprint (Fig. 5b). The map was provided to the robot at the start of the simulation, and the robot planned in 3-D (x, y, θ) for the high dimensional state-space. 50 maps of three different sizes and two types were randomly generated. The indoor maps consisted of many narrow hallways with the occasional larger room. In addition there were several thousand single point obstacles randomly placed (Fig. 7). The other type of map was more open with many rectangular obstacles and single point obstacles placed randomly (Fig. 8). For both map types, the start and goal states were placed in diagonally opposite corners with a heading of 0°. Our algorithm and planning on a uniform 3-D state-space (both using optimal $A^\star$ were run on each of the maps and the results are presented in Table I and Table II. Since the maps were randomly generated, there were a few maps in each category that had no feasible solution, so the number of maps actually completed is presented and all statistics are based on only those maps.

As can be seen our algorithm had comparable path costs to the optimal search ($< 110\%$ of the optimal cost on average for each map size and type), with significantly lower planning times (42-67 times faster on average for each map size and type). In the outdoor environments with their "more convex" obstacles and less dead-ends the algorithm put very few high-D regions into the search graph in comparison with the more complex indoor maps.

After verifying that the algorithm performed satisfactorily in the simulation environment, we ran it on our differential drive robot. In order to make it a fair comparison with full 3-D $A^\star$ type searches we added a long boom on the robot so

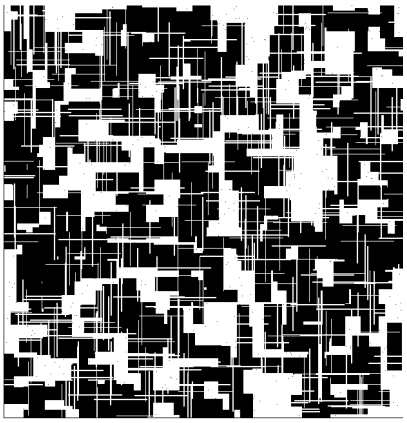


Fig. 7: Example 2000×2000 indoor map for the simulation environment. Black areas are obstacles, in addition there are several thousand single cell obstacles randomly spread across the map. Start is in the upper left corner and the goal is in the lower right corner.

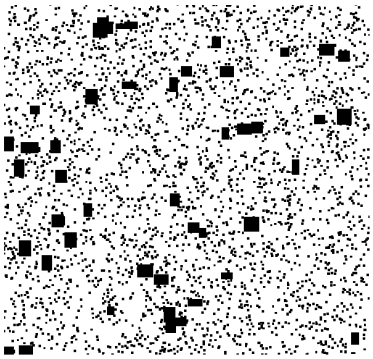


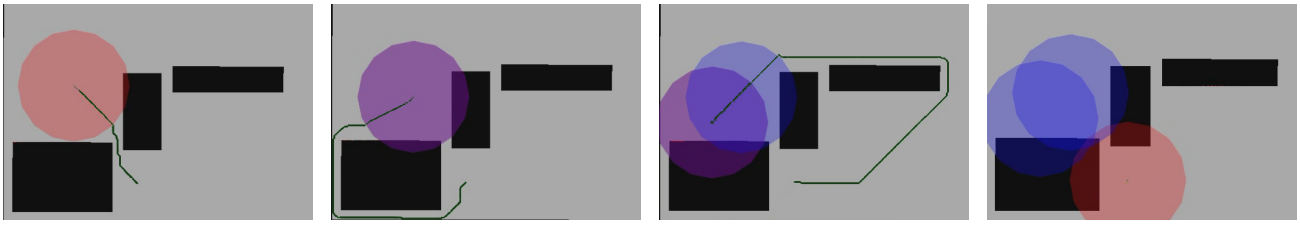
Fig. 8: Example 2000×2000 outdoor map for the simulation environment. Coloring and positions are the same as for the indoor map.

that its orientation played a key roll in whether a particular area of the map was traversable as shown in Fig. 5a. We ran a series of test runs an example of which is shown in Fig. 9. During each of the test runs the robot replans every meter of distance traveled and used a five meter radius for the 3-D circles. For this particular example (the full motion video is included in the accompanying video submission[3]), the robot was attempting to go around the corner near some obstacles in the middle of the hallway. The spacing of the obstacles was enough for the robot to fit through, if the robot did not have the boom attached, it would have been able to make the corner. Once the 3-D circle encompasses the obstacles, the planner finds a better path going around the central core of the map. A 3-D circle was placed at this point due to the homotopic class changing so that as the robot replanned it retained the knowledge that the upper right corner was not traversable.

REFERENCES

[1] K. Bekris and L. Kavraki, "Greedy but safe replanning under kinodynamic constraints," in Robotics and Automation, 2007 IEEE

[3] http://www.youtube.com/watch?v=Y1TZMQTEahs



(a) (b) (c) (d)

Fig. 6: Simple environment, black areas are obstacles, robot R with active 3-D search areas depicted by the colored circles. (a) Robot at start (b) Robot reaches obstacles and determines it cannot fit, replans and leaves permanent 3-D area (in purple) (c) Robot determines it also cannot traverse the far left gap, replans, and leaves 3-D area (in purple, previous circle in blue) (d) Robot reaches goal

TABLE I: Combined Graph Algorithm Simulation Results

Map Type	Map Size (cells)	# of Maps	Initial Plan Time (s)	Avg Plan Time (s)	# of 2-D Expands	# of 3-D Expands	# of Circles	Final Path Cost
Outdoor	2000×2000	43	0.84	0.14	64062	7241	2.19	275823
	3000×3000	45	1.68	0.33	149268	7734	5.09	419057
	4000×4000	48	2.42	0.59	234310	7209	7.40	493062
Indoor	2000×2000	50	1.95	0.83	319069	7718	32	653451
	3000×3000	49	5.13	2.84	850135	9095	64	1076430
	4000×4000	45	10.18	5.15	1426658	8051	63	1325369

TABLE II: $A^\star$ Comparison Results

Map Type	Map Size (cells)	# of Maps	Initial Plan Time (s)	# of Expands	Final Path Cost
Outdoor	2000×2000	43	35.56	2328883	257108
	3000×3000	45	93.36	5280830	388813
	4000×4000	48	162.14	8506620	458966
Indoor	2000×2000	50	108.14	7292373	610800
	3000×3000	49	339.23	18380536	978386
	4000×4000	45	594.39	29472991	1233831

International Conference on, april 2007, pp. 704–710.

[2] S. Petti and T. Fraichard, "Safe motion planning in dynamic environments," in Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on, aug. 2005, pp. 2210–2215.

[3] M. Likhachev and D. Ferguson, "Planning long dynamically feasible maneuvers for autonomous vehicles," Int. J. Rob. Res., vol. 28, pp. 933–945, August 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1577179.1577184

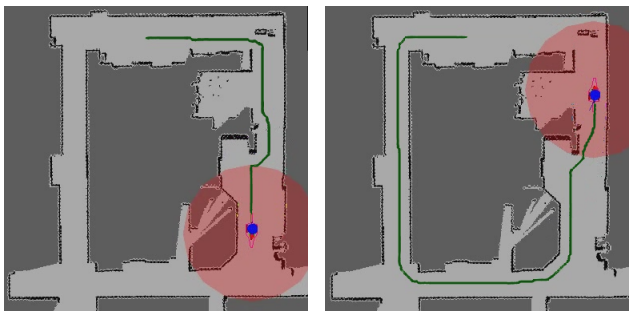
[4] O. Brock and O. Khatib, "High-speed navigation using the global dynamic window approach," in Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on, vol. 1, 1999, pp. 341–346 vol.1.

[5] A. Kelly, "An intelligent predictive control approach to the high-speed cross-country autonomous navigation problem," Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-95-33, September 1995.

[6] R. Philippsen and R. Siegwart, "Smooth and efficient obstacle avoidance for a tour guide robot," in Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on, vol. 1, sept. 2003, pp. 446–451.

[7] K. Gochev, B. J. Cohen, J. Butzke, A. Safonova, and M. Likhachev, "Path planning with adaptive dimensionality." in SOCS, D. Borrajo, M. Likhachev, and C. L. Lpez, Eds. AAAI Press, 2011.

[8] S. Bhattacharya, V. Kumar, and M. Likhachev, "Search-based path planning with homotopy class constraints," in Third Annual Symposium on Combinatorial Search, 2010.



(a) Start (b) Replan

Fig. 9: (a) the robot initially plans a path through the upper right corner to the goal. (b) once the robot reaches the corner it discovers that it cannot maneuver past the obstacles and plans a path around the loop, leaving a 3-D circle at its present location.