2008 IEEE/RSJ International Conference on Intelligent Robots and Systems
Acropolis Convention Center
Nice, France, Sept, 22-26, 2008

# Differentially Constrained Motion Replanning Using State Lattices with Graduated Fidelity

Mihail Pivtoraiko and Alonzo Kelly

*Abstract*— **This paper presents an approach to differentially constrained robot motion planning and efficient re-planning. Satisfaction of differential constraints is guaranteed by the *state lattice*, a search space which consists of motions that satisfy the constraints by construction. Any systematic replanning algorithm, e.g. D*, can be utilized to search the state lattice to find a motion plan that satisfies the differential constraints, and to repair it efficiently in the event of a change in the environment. Further efficiency is obtained by varying the fidelity of representation of the planning problem. High fidelity is utilized where it matters most, while it is lowered in the areas that do not affect the quality of the plan significantly. The paper presents a method to modify the fidelity between replans, thereby enabling dynamic flexibility of the search space, while maintaining its compatibility with replanning algorithms. The approach is especially suited for mobile robotics applications in unknown challenging environments. In this setting, we applied the planner successfully to the navigation of research prototype rovers in JPL Mars Yard.**

## I. INTRODUCTION

In recent years there has been a growing interest in efficient motion replanning. Real mobile robot applications face challenges due to scarce and uncertain perception information. In order to facilitate planning robot's motion given such challenges, dynamically replanning algorithms were developed [13] [6]. Such algorithms incorporate updated perception information and modify the motion plan accordingly, while reducing computation time by reusing previous computation.

This work introduces efficient replanning to motion planning under differential constraints that is based on searching a *state lattice*, a directed cyclic graph that encodes the constraints by construction [11]. Substantial computation is performed off-line to determine the connectivity of edges that represents the differential constraints. This allows fast planning (on-line) by utilizing standard search algorithms in this graph, while naturally satisfying the constraints.

In order to satisfy the differential constraints, relatively high dimensionality of the state lattice may be required. Deterministic search in this setting can be computationally costly. This cost is especially significant in outdoor robotics applications, as they pose complicated planning problems, in particular due to complex environments.

This paper addresses this limitation by managing the complexity of the search through modification of the fidelity of representation. The search space consists of one or more arbitrary regions of different fidelities. Lower fidelity of representation results in faster search, but higher fidelity results in better quality solutions. The approach is closely related to multi-resolution planning [8], but we use the term *graduated fidelity* to emphasize that the quality of representation is expressed not only as the resolution of state discretization, but also as the connectivity of edges between the vertices in the state lattice. Each region of the search space can be assigned a fidelity arbitrarily, yet practically this choice is guided by the region's relevance for the planning problem and the availability of the environment information. In particular, it is often beneficial to utilize a high fidelity of representation in the immediate vicinity of the moving robot. Our method meets that need by allowing the regions of different fidelity to move or change shape arbitrarily.

The contribution of this work is an improved state lattice search space that consists of regions of different fidelities of representation and allows the regions to move or change shape between replans. This search space remains compatible with standard search algorithms and is capable of producing motion plans that satisfy differential constraints without any post-processing. The presented method allowed real-time motion planner operation onboard a mobile robot in rough outdoor terrain.

## II. PRIOR WORKS

A planner based on A* search in the state lattice was successfully utilized to guide a car-like robot in challenging natural environments [11]. The graduated fidelity extension to that work, presented herein, is related to the general area of multi-resolution planning: [3], [2], [9] and others. One difference our method has with most multi-resolution predecessors is that regions of different resolution are allowed to move over time, while the search space remains compatible with systematic search. The idea of dividing the search space into regions is related to [14], but our method allows replanning in this search space.

Satisfaction of differential constraints also has received a considerable amount of attention in motion planning research. Powerful probabilistic techniques have been developed [7] [5], however our method is deterministic and under appropriate conditions can offer a number of guarantees provided by standard search algorithms, including optimality and resolution-completeness. A number of other approaches utilize discretization in control space to manage the complexity of the planning problem [1]. However, there are important advantages to using discretization in the state space instead.

This research was conducted at the Robotics Institute of Carnegie Mellon University, sponsored by NASA/JPL as part of the Mars Technology Program.

The authors are with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA {mihail, alonzo}@cs.cmu.edu

In particular, it simplifies reducing dispersion of sampling, in turn allowing a more uniform distribution of samples in the state space [8]. This is beneficial to exploring the state space more efficiently, as the search attempts to find a path from initial to final state. Unfortunately, reducing state space dispersion through control space sampling is difficult. It was shown in [10] that through careful discretization in control space, it is possible to force the resulting reachability graph of a large class of nonholonomic systems to be a lattice, however this is usually difficult to achieve. By using a boundary value problem solver [4], we can choose a convenient discretization in the state space, one that makes the search more efficient, while maintaining the satisfaction of differential constraints.

## III. DIFFERENTIALLY CONSTRAINED PLANNING AS SEARCH IN STATE LATTICES

In this section we develop some nomenclature to define the motion planning problem under differential constraints and to review a method to solve it using search in state lattices [11].

The motion planning problem we consider is a six-tuple $(X, X_{free}, x_{init}, x_{goal}, U, f)$. The robot *state space*, $X \subset \mathbb{R}^n$, is an $n$-dimensional compact differentiable manifold. $X_{free} \subseteq X$ is the set of states that satisfy global constraints (e.g. control bounds, obstacle avoidance, etc.). The boundary conditions for the planning problem are $x_{init} \in X_{free}$ and $x_{goal} \in X_{free}$. The set of robot controls $U$ contains the control inputs that the system accepts. A function $f$ is the system model (equation of motion) and encodes kinematics and dynamics constraints: $\dot{x} = f(x, u)$, where $x \in X, u \in U$. The solution to the motion planning problem is an input function $u_s : [t_0, t_f] \rightarrow U$, where $t_0$ is the starting time and $t_f$ is the final time.

The proposed method to solve the above problem hinges on a particular discretization of the robot state space, the state lattice. It is represented as a directed cyclic graph, where vertices are discrete values of state. An edge between two vertices $x_i, x_j$ is an input function $u_i : [0, 1] \rightarrow U$, such that the corresponding path $\pi_i : [0, 1] \rightarrow X_{free}$ (obtained by integrating $f(x_i, u_i)$) satisfies $\pi_i(0) = x_i$ and $\pi_i(1) = x_j$. Thus, every vertex of the state lattice is connected to one or more other vertices via edges that represent feasible motions. Further, finding a motion plan that satisfies differential constraints is reduced to finding a path (a sequences of vertices and edges) in the state lattice graph. Each edge can be assigned a cost. Since the state lattice is a directed cyclic graph, any standard systematic search algorithm can be utilized to find the shortest path in it, which would correspond to a minimum-cost feasible motion that drives the system from $x_{init}$ to $x_{goal}$.

Generating the input functions (state lattice edges) involves solving the *boundary value problem*, and we rely on a significant amount of prior work in this area [8]. We successfully used a method of trajectory generation for wheeled mobile robots in [4]. Often it is possible to find a feasible path from any state lattice vertex to infinitely many other vertices. The branching factor of such a graph would be unmanageable, and therefore we must limit the connectivity. Finding the best way to arrange the connectivity of vertices in the state lattice is related to the problem of optimal sampling, and a number of approaches have been developed [11].

The principle of limiting the connectivity between lattice vertices to make computation manageable can be viewed as discretization in the space of motions. Similar to any discretization, it has implications on the completeness of the planner. However, the proposed approach maintains the resolution-completeness, if we allow the term "resolution" to apply to both state and motion discretization. It may be helpful to see that planning in state lattices is a generalization of the classical approaches to planning in 2D grids [13] [6]. The principal difference is the connectivity between the vertices. It is trivial in the case of the grid ($N$-nearest neighbors), while the state lattice requires the system model and a boundary value problem solver to compute the connectivity.

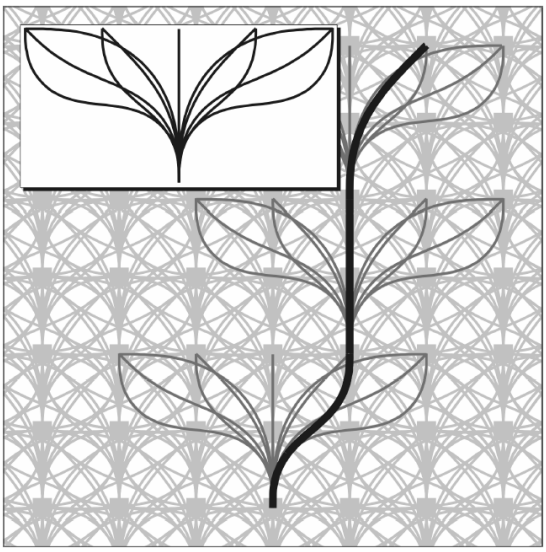

Fig. 1. A repeated and regular pattern of vertices and edges comprises the state lattice. The inset shows the immediate neighbors of a node, including 2D projections of the edges (workspace paths) leading towards the neighbors. The overall motion plan (thick black curve) is a sequence of such edges.

## IV. GRADUATED FIDELITY PLANNING

In this section, we describe an approach to reducing the computational complexity of planning with state lattices at the cost of selectively reducing the quality of representation. Some regions of the workspace, e.g. those that are partially or completely unknown, can be represented at lower fidelity, and the regions that are best known or most relevant for the problem are represented at highest fidelity.

We extend the above definition of the state lattice by assuming that it consists of subgraphs $\mathbf{G_1}, \mathbf{G_2}, \ldots, \mathbf{G_n}$. The arrangement of vertices and edges in each subgraph is assumed to be regular, but this arrangement may be different among subgraphs to reflect the differences in the fidelity of

representation. This composite search space is maintained to remain a directed cyclic graph, so that replanning algorithms can be utilized to reuse previous computation while replanning.

We define *modifying* a subgraph as arbitrarily changing its position (in coordinates that do not affect its connectivity, namely the translational ones) and shape (the extent of its boundary in state space). After a subgraph is modified, some of the vertices near its boundary are set to belong to a different subgraph. Note that the vertices do not move, they simply change ownership from one subgraph to another. For example, as a subgraph changes position, it "gains" the vertices on its leading edge and "loses" vertices on its trailing edge.

After all subgraphs are modified as desired, a replanning procedure needs to be executed to repair the plan. This is the same procedure that repairs the plan due to other changes in the search space (e.g. a perception update). To see why, note that deterministic replanning algorithms reuse computation by storing previously computed costs of vertices in the graph. As a result of modifying the subgraphs, some vertices change their cost due to new connectivity under a different subgraph. It is entirely transparent to the replanning algorithm whether they changed cost due to new edge costs from a perception update, or due to modification of subgraphs. Thus, the only required change to replanning algorithms to enable graduated fidelity is a process to make them aware of the vertices that have new subgraph ownership. This is performed by the function *convert_vertex*, presented in Algorithm 1.

The function *convert_vertex* is executed on each vertex $v_b$ that changes subgraph ownership. If the vertex has not been expanded at all so far, the function returns. Otherwise, we note all vertices that may contain $v_b$ as a predecessor – it is exactly the set of successors of $v_b$ under the edge connectivity of its previous subgraph $\mathbf{G_i}$, denoted as $Succ_i(v_b)$. Further, we remove any back-pointers from these successor vertices $v_s$ to $v_b$ by examining the predecessors of $v_s$, $Pred(v_s)$. Effectively, we undo the effects of a previous expansion of $v_b$. Lastly, if this change resulted in a change of cost of any successor vertex, we insert the affected vertices and $v_b$ itself into the priority queue. D* variants can detect this cost change automatically by recomputing the $rhs$-value. Note that this procedure is likely to cause replanning from the farthest affected vertex to the robot (assuming the search direction from the goal to the robot). Thus, more previous computation is reused if such changes occur closer to the robot.

This procedure is illustrated in Figure 2, using a simplified search space for ease of visualization. In this example, the search space consists of two subgraphs, $\mathbf{G_1}$ (black square vertices) and $\mathbf{G_2}$ (gray circle vertices). Arrows of similar colors are the edges. $\mathbf{G_2}$ is a small subgraph, consisting of six vertices, highlighted with a gray bounding box. The rest of the search space belongs to $\mathbf{G_1}$. In this example, $Succ_1(v_i), \forall v_i \in \mathbf{G_1}$ is 4 nearest neighbors, and $Succ_2(v_j)$, $\forall v_j \in \mathbf{G_2}$ is 8 nearest neighbors. A real implementation of graduated fidelity under differential constraints would utilize the same algorithm, but a more sophisticated connectivity of the subgraphs.

**Input**: graph vertex $v_b$
**if** $v_b$ *was previously expanded* **then**
  **foreach** $v_s \in Succ_i(v_b)$ **do**
    remove $v_b$ from $Pred(v_s)$;
    update_vertex($v_s$);
  update_vertex($v_b$);

**Algorithm 1**: The function *convert_vertex*($v_b$). It enacts a change of subgraph ownership of the vertex $v_b$. After it has been called on all vertices that changed subgraphs, the search space becomes ready for a standard replanning process. The *update_vertex* function is assumed to be a component of the chosen search algorithm that determines whether a vertex needs to be processed during replanning.

The effect of the initial plan is shown in Figure 2a). The search proceeded from the goal vertex $v_g$ to the robot $v_r$. The vertices with white centers were expanded during search, and solid vertices, pointed to by the arrows, remain in the priority queue. The resulting motion plan is highlighted with a thick patterned line.

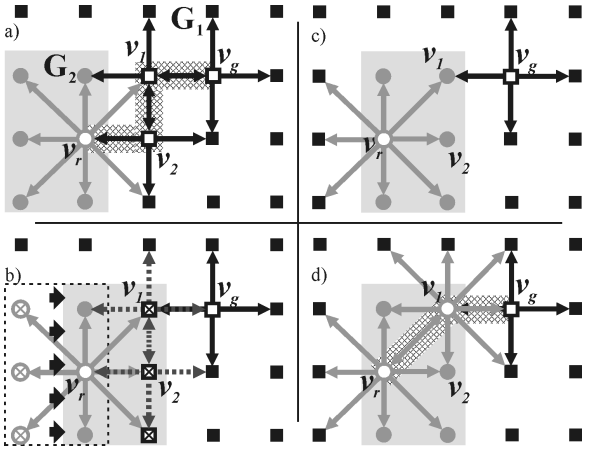


Fig. 2. Maintaining the connectivity between two subgraphs of different fidelities of representation. $\mathbf{G_1}$ is a static subgraph (black square vertices), and $\mathbf{G_2}$ (gray circular vertices) moves w.r.t. the former. Arrows are edges. Hollow vertices have been expanded. Subfigure a) shows the initial plan (thick patterned line), as it originates in $\mathbf{G_1}$ and proceeds into $\mathbf{G_2}$; b) as $\mathbf{G_2}$ moves from left to right, the six crossed out vertices change subgraph ownership, and *convert_vertex* is executed on each of them, which results in undoing the previous expansions of $v_1$ and $v_2$; c) shows the completion of moving $\mathbf{G_2}$: the vertices $v_1$ and $v_2$ now belong to $\mathbf{G_2}$ and are available for re-expansion, if necessary, when the search algorithm performs replanning; lastly, d) shows the result of replanning in the search space from c), where due to re-expansion of $v_1$ under $\mathbf{G_2}$ edge connectivity, a new plan is found.

Next, suppose we move $\mathbf{G_2}$ to the right, as shown in Figure 2b). The six crossed-out vertices change subgraph ownership due to this move, and we execute *convert_vertex* on each of them. The previous expansion of vertices $v_1$ and $v_2$ is undone: their edges (dotted arrows) are removed. In Figure 2c), moving $\mathbf{G_2}$ is completed, and the affected vertices are inserted into the priority queue. When the search

algorithm begins replanning, these vertices will be expanded, using the edge connectivity of $\mathbf{G_2}$, if they are relevant for the problem at hand, as deemed by the heuristic. Figure 2d) completes our example and shows a new motion plan (thick patterned line) due to moving $\mathbf{G_2}$ to the right.

The same algorithm would work with subgraphs of different dimensionalities by using additional "connecting" edges. They are used as part of the expansion of a vertex that lies on the boundary of a subgraph, in order to connect it with the other subgraph. A simple example of connecting a 2D subgraph $\mathbf{G_1}$ to a 3D subgraph $\mathbf{G_2}$ is shown in Figure 3. Additional edges (gray arrows in the figure), denoted by $Succ_{12}(v_b)$ are used in order to generate suitable connectivity from a vertex $v_b$ in subgraph $\mathbf{G_1}$ to $\mathbf{G_2}$. In this case, we have connected this vertex to several vertices in $\mathbf{G_2}$ representing all possible values of the third dimension. Algorithm 1 can be utilized here with a minor modification. In addition to using the vertices in $Succ_i(v_b)$ in the *foreach* loop, we also use the vertices from $Succ_{ij}(v_b)$. This addition certainly increases the branching factor of boundary vertices, however the complexity effect can be insignificant if only a small number of vertices require the extra edges.

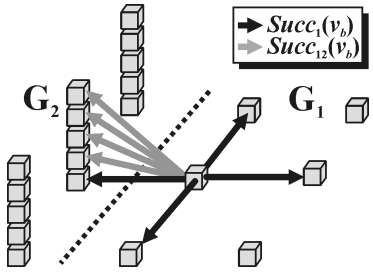


Fig. 3. The 2D subgraph $\mathbf{G_1}$ (4-connected grid) is connected to another subgraph $\mathbf{G_2}$ of a higher dimension. Black arrows are the standard node expansion (4 nearest neighbors), and gray arrows are additional edges that connect the two subgraphs.

## V. IMPLEMENTATION DETAILS

In this section we describe several important details that would be helpful in evaluating the proposed approach. In the next section, we quote the results obtained using the implementation described herein. D* Lite [6] was the search algorithm used in the described implementation, and the details below reflect this choice.

### A. Designing the Connectivity of the State Lattice

By regularity of the state lattice, the set of immediate successors of any vertex does not depend on translational coordinates of the vertex. This is important, because all but translational coordinates of the lattice have finite range. Thus, it is possible to pre-compute and store the *control set*, a finite set of successors for any vertex in the lattice, up to translation.

In Section III, we described the role of designing the connectivity of the state lattice and its importance for the quality of planning. This is a topic of on-going research, and a number of approaches have been developed [11].

### B. Heuristics

There are a number of heuristics that can be used. The simplest one is the Euclidean distance $L_2(x, x_{goal})$, where $x, x_{goal} \in X'_{free}$, a subset of $X_{free}$ consisting of its two translational dimensions. However, Euclidean distance is not a well-informed heuristic, because the length of a differentially constrained path between two points in 2D can be much greater than the Euclidean distance between them. Weighted Euclidean distance metrics in $X$ can be designed to be better heuristics. Other heuristic ideas include distance metrics derived from the Reeds-Shepp result [12] and heuristic look-up tables, containing the pre-computed free-space costs of paths [11]. The latter are especially attractive in state lattice planning with graduated fidelity, since the sizes of the finite subgraphs suggest natural sizes for the corresponding heuristic look-up tables.

### C. Processing Edge Cost Updates

For every change in the cost, $c(x_i, x_j)$, of the directed edge from the vertex $x_i$ to $x_j$, the D* Lite algorithm requires recomputing the priority of $x_j$ and potentially inserting it into the priority queue. Without loss of generality, we assume that environment constraints are represented in a 2D cost map. In order to re-plan due to a change in the cost of a cell $m_{ij} \in \mathbb{N}^2$ in the cost map, the planner needs to know the set of vertices $V_c$ that potentially need to be re-inserted into the priority queue with new priority. Thus, the planner requires a mapping $Y : \mathbb{N}^2 \rightarrow V_c$. This mapping is simple in the case of classical applications of D* in 2D grids, but it is nontrivial in the case of the state lattice.

To develop this mapping, we need to compute the *swath* of a motion, a set of cost map cells $C_s \subset \mathbb{N}^2$ that are covered by the robot as it executes the motion. The cost of an edge that represents this motion is directly dependent on the costs of map cells in $C_s$. Once we pre-compute the control set, it is also often possible to pre-compute the swaths of the edges in it.

Since the mapping between edges and their terminal vertices is trivial, it is easier first to develop the mapping $Y' : \mathbb{N}^2 \rightarrow E_c$, where $E_c$ is the set of edges that are affected by $m_{ij}$ (e.g. pass through it). Determining $Y'$ may still appear as a formidable task, given the high density of edges in the multi-dimensional state lattice. However, we again exploit the regularity of the lattice to simplify the problem. If we have $Y'' : O \rightarrow E_c$, where $O$ is the origin, then $Y' = Y'' + n, \forall n \in \mathbb{N}^2$. In other words, the set of edges, affected by $m_{ij} = O$ is identical for any other cell, up to the translation coordinates. Further, recall that the swath $C_s$ of each edge in $E_c$ is known. In principle, $E_c$ contains all edges $u_c$, such that $m_{ij}$ belongs to $C_s$ of $u_c$. Hence, the mapping $Y''$ is exactly a set of edges, whose swaths pass through the 2D origin. Clearly, it can be precomputed for the same reasons as the control set and can be used readily to incorporate cost map changes in re-planning.

## VI. EXPERIMENTAL RESULTS

In order to evaluate the effect of graduated fidelity in differentially constrained motion planning with state lattices, we performed simulated and field experiments with mobile robots. In all experiments, a planetary rover prototype roughly 0.8m by 1.0m in size was used. It was assumed to be a nonholonomic vehicle capable of moving forward and back with minimum turning radius 0.5m. The vehicle was capable of point-turns, although such maneuvers were considered costly and used as a last resort. The search space consisted of two subgraphs: a high fidelity subgraph that moved with the vehicle and a low fidelity subgraph elsewhere. The $(x, y)$ resolution of both subgraphs featured square cells of 20 cm. The average outdegree of the 4D high fidelity subgraph was 45 (a state lattice). The outdegree of the 2D low fidelity subgraph was 8 (grid connectivity). The dimensions of the state lattice included 2D translational coordinates $(x, y)$ discretized as a grid, heading and curvature. The planner used the Euclidean distance heuristic, and further performance improvement can be expected with better informed heuristics.

The results of two types of experiments are presented below. The first set of experiments attempts to quantify the effects of the size of the high fidelity subgraph around the vehicle on both runtime performance and the quality (cost) of the solution. The second set of simulated and real vehicle experiments demonstrates the performance of the planner in a realistic setting.

The plots in Figure 4 report the results of the first type of experiments. The planner used an environment with random single map cell obstacles independently and identically distributed with probability of 3%. This was a similar environment as in the experiment shown in Figure 5, and an example of it can be seen in the top part of the figure. To avoid confounding the results, only the initial plan statistics are presented (no replanning), and the cost map was fully known *a priori*. For each experiment, the robot was placed at a random location in its environment, and the goal was chosen approximately 100 cells away. Both the runtime (left subfigure) and solution cost (right) measurements were normalized by dividing by the respective quantity as featured by the lattice planner without graduated fidelity. As the plots demonstrate, significant runtime savings were attained (over 10x), while the cost of the solution was not significantly increased. In evaluating other goal selection schemes and different environments, we found that the specifics of the plots in Figure 4 can vary (e.g. the dip in the runtime plot), while the general trends remain unchanged. The graduated fidelity appears to offer performance improvement invariant to the differences in environment. Its benefit was noted to be the greatest in challenging environments.

The second set of experiments demonstrates planner performance in a more realistic setting, where the robot moves in a challenging environment toward a distant target. In the simulated experiment, the robot has a perception region limited to 21x21 cells, centered around the robot. No perception information is available outside this horizon. The size of the high fidelity subgraph was the same size as the perception region. Otherwise, the setup is the same as above. In this setting, we tested planner operation over a 500 meter traverse in the above environment. For clarity, Figure 5 shows a 40-meter subset of the traversed path. Grey cells are obstacles that have not yet come into view of the robot and are unknown to it. Black cells (and red cells in the insets) are obstacles that were seen by the robot. The dark-gray line is the path the robot traveled. Note that it appeared in many cul-de-sacs due to the limited perception, and the planner was effective at guiding the robot out of all of them, while leveraging robot's maneuverability. The replanning due to obstacle discovery and subgraph modification in the search space was occurring continuously.

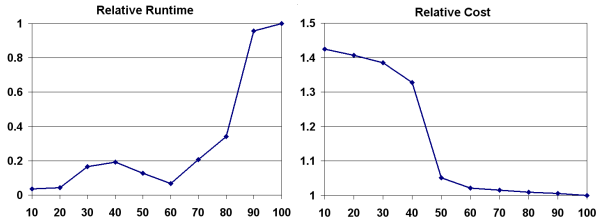


Fig. 4. The results of the initial plan using graduated fidelity. The horizontal axis in both plots is the percentage of the search space (between robot and goal) occupied by the high fidelity subgraph. Left subplot shows relative runtime, expressed as a ratio of graduated fidelity runtime versus full fidelity runtime. Similarly, the right subplot shows the relative cost of the path, the ratio of computed path cost to the best known cost derived by full fidelity planner.

This experiment was performed on a conventional laptop computer with 2GHz CPU and 2GB of RAM. The lower part of Figure 5 shows the runtime of the planner in the experiment above. The $x$-axis of this plot is the count of replan cycles that were necessary to travel along the shown path (total of about 150). The $y$-axis is the runtime (in seconds) of the planner. Notice the two peaks in the middle of the plot: they correspond to two replans #39 and #53, when significant replanning was required due to cul-de-sacs. These situations are highlighted in the top portion of Figure 5.

The graduated fidelity motion planner was integrated with research prototype rovers at JPL. It enabled rovers to navigate without stopping in rough rocky terrain set up in the JPL Mars Yard. Figure 6 shows the results of the FIDO rover running the graduated fidelity state lattice planner on-board to navigate autonomously amid dense rocks. In this experiment, the robot featured a single 1.6GHz CPU and 512MB of RAM, shared among all processes of the rover. The actual memory usage of the planner was less than 100MB. The rover used a perception region (via stereo cameras) and high fidelity region of the same size as above, 21x21 map cells, centered around the rover. The top part of Figure 6 shows the approximate path the rover traveled, and the bottom part shows a semi-log plot of the runtimes of the corresponding replans. As the plot shows, the lattice planner demonstrated replan rates of approximately 10Hz in field experiments.

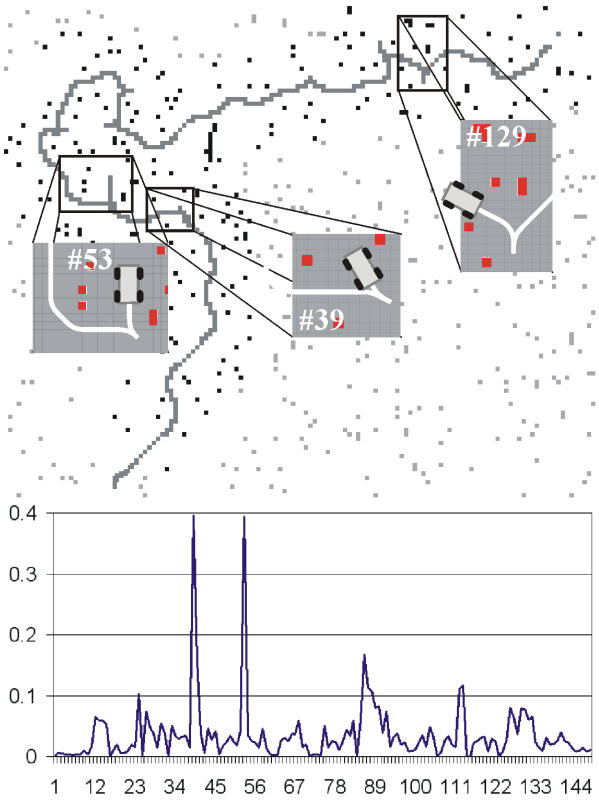


Fig. 5. A simulated experiment of traversing about 500 meters among previously unknown obstacles. Top: the first 40 meters of the path are shown for clarity. Note that all motions generated by the planner were globally feasible, and backing up maneuvers were generated automatically when necessary. Bottom: plot of planner runtime (vertical axis is runtime in seconds, and the horizontal axis is replan cycle number).

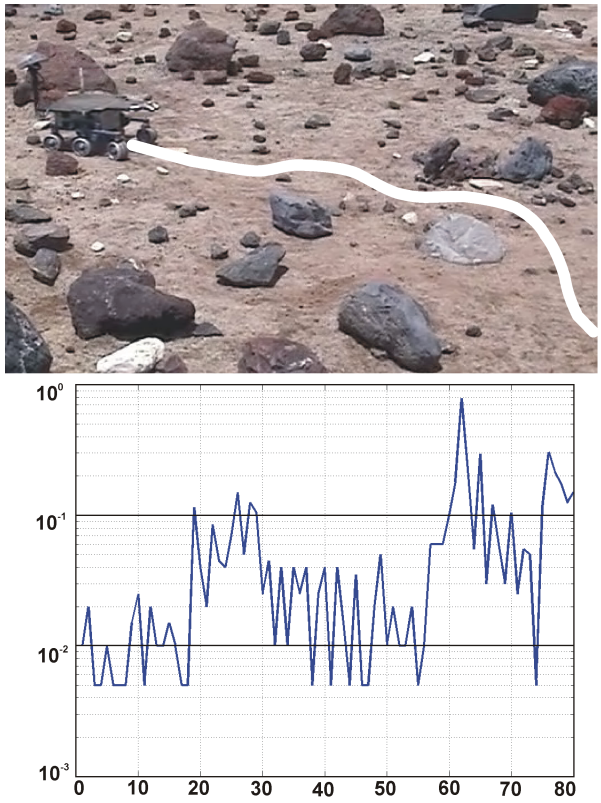


Fig. 6. A field experiment in the JPL Mars Yard. FIDO rover navigated autonomously among previously unknown obstacles, while running the graduated fidelity lattice planner on-board. Top: approximate path the rover followed. Bottom: a semi-log plot of planner runtime (vertical axis is runtime in seconds, and the horizontal axis is replan cycle number).

## VII. CONCLUSIONS AND FUTURE WORK

In this paper we described an approach to improve the efficiency of motion planning and replanning by varying the fidelity of representation of the planning problem. In addition to leveraging dynamic replanning algorithms, this approach enables dynamic and deliberative flexibility in search space topology to boost efficiency. Standard replanning algorithms can be utilized, while the proposed search space design allows both the automatic satisfaction of differential constraints and the adjustment of the search space between replans. The method was successfully demonstrated in simulation and on real robots. Future work includes a further investigation into the state and motion space sampling to further improve planning efficiency.

## REFERENCES

[1] J. Barraquand and J.-C. Latombe. On nonholonomic mobile robots and optimal maneuvering. In *Proc. of the IEEE International Symposium on Intelligent Control*, 1989.

[2] R. Bohlin. Path planning in practice; lazy evaluation on a multi-resolution grid. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2001.

[3] D. Ferguson and A. Stentz. Multi-resolution Field D*. In *Proc. International Conference on Intelligent Autonomous Systems (IAS)*, 2006.

[4] T.M. Howard and A. Kelly. Optimal rough terrain trajectory generation for wheeled mobile robots. *International Journal of Robotics Research*, 26(2):141–166, 2007.

[5] D. Hsu, R. Kindel, and J.-C. Latombe S. Rock. Randomized kinodynamic motion planning with moving obstacles. *International Journal of Robotics Research*, 21(3):233–255, 2002.

[6] S. Koenig and M. Likhachev. D* Lite. In *Proceedings of the AAAI Conference of Artificial Intelligence (AAAI)*, 2002.

[7] S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. *Algorithmic and Computational Robotics: New Directions*, pages 293–308, 2001.

[8] S.M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

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

[10] S. Pancanti, L. Pallottino, and A. Bicchi. Motion planning through symbols and lattices. In *Proc. of the Int. Conf. on Robotics and Automation*, 2004.

[11] M. Pivtoraiko and A. Kelly. Constrained motion planning in discrete state spaces. In *Proc. of the Int. Conf. on Field and Service Robotics*, 2005.

[12] J. A. Reeds and L. A. Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific Journal of Mathematics*, 145(2):367–393, 1990.

[13] A. Stentz. The focussed D* algorithm for real-time replanning. In *Proceedings of the Fourteenth International Joint Conf. on Artificial Intelligence*, August 1995.

[14] R. Szczerba, D. Chen, and J. Uhran. Planning shortest paths among 2D and 3D weighted regions using framed-subspaces. *International Journal of Robotics Research*, 17(5):531–546, 1998.