Fast and Feasible Deliberative Motion Planner for Dynamic Environments

Mihail Pivtoraiko and Alonzo Kelly

Abstract— **We present an approach to the problem of differentially constrained mobile robot motion planning in arbitrary time-varying cost fields. We construct a special search space which is ideally suited to the requirements of dynamic environments including a) feasible motion plans that satisfy differential constraints, b) efficient plan repair at high update rates, and c) deliberative goal-directed behavior on scales well beyond the effective range of perception sensors. The search space contains edges which adapt to the state sampling resolution yet aquire states exactly in order to permit the use of the dynamic programming principle without introducing infeasibility. It is a symmetric lattice based on a repeating unit of controls which permits off-line computation of the planner heuristic, motion simulation, and the swept volumes associated with each motion. For added planning efficiency, the search space features fine resolution near the vehicle and reduced resolution far away. Furthermore, its topology is updated in real-time as the vehicle moves in such a way that the underlying motion planner processes changing topology as an equivalent change in the dynamic environment. The planner was originally developed to cope with the reduced computation available on the Mars rovers. Experimental results with research prototype rovers demonstrate that the planner allows us to exploit the entire envelope of vehicle maneuverability in rough terrain, while featuring real-time performance.**

I. INTRODUCTION

Capable motion planners are important for enabling field robots to perform reliably, efficiently and intelligently. Despite decades of significant research effort, today the majority of field robots still exhibit various failure modes due to motion planning deficiencies. These failure modes range from computational inefficiencies to frequent resort to operator involvement when the autonomous system takes unnecessary risks or fails to make adequate progress. Based on our field robotics experience, we have developed a motion planning method that addresses many drawbacks of leading approaches. It is a deterministic, sampling-based method. It features a sampling of robot state space that lends itself well to utilizing standard graph search techniques, while enabling an array of high-performance planning capabilities. The proposed motion planning method was implemented and successfully validated in field experiments at the California Institute of Technology, Jet Propulsion Laboratory (JPL; Figure 1).

While the planner was originally developed for the Mars rovers, it is equally relevant to dynamic environments. In fact, a variant of this planner was used in CMU's winning entry in the DARPA Urban Challenge [1]. Motion planners intended for use in dynamic environments must be:

- Fast, in order to replan on-the-move when changes in the environment are detected,
- Feasible (meaning satisfying differential constraints) at least in the near field, so that the predicted path to avoid dynamic obstacles is the one actually executed by the vehicle,
- Deliberative, so that effective goal-directed behavior can be maintained despite the path perturbations caused by reacting to changes in the environment such as the motions of dynamic obstacles and the appearance and disappearance of local minima.

In order to satisfy the differential constraints, we propose to encode them in the search space. This allows to shift the constraint satisfaction to the search space design process, which can be accomplished *a priori* and off-line. In particular, we enforce that the edges of the graph that represents the planning problem represent the feasible motions that can be directly executed by the robot. In this manner, the on-line planner can utilize unconstrained, standard search algorithm to find a solution to the motion planning problem, a path in the representation graph.

For most systems featuring differential constraints, relatively high dimensionality of the state space may be required. Deterministic search in this setting can be computationally costly. Planning in complex outdoor environmens can be especially computationally expensive due to any combination of scale, dynamics, and dense obstacles.

Fig. 1. FIDO rover navigates autonomously using the proposed motion planner among dense obstacles in the Mars Yard at the California Institute of Technology, Jet Propulsion Laboratory. The planner manages frequent updates of the limited perception system by replanning continuously at approximately 10Hz. Computed planner motions are smooth in curvature and executable by the robot verbatim.

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

This paper proposes a two-fold solution to this difficulty. First, the search space is designed to be compatible with replanning algorithms [2], [3] that can repair the motion plan efficiently when the representation of the environment changes. This is particularly important in robotics applications in partially known or dynamic environments, where changes in environment information are frequent. The robot must be able to recompute its motion plan quickly. The second manner of alleviating computational complexity of planning is through modification of the fidelity of representation of the motion planning problem. The proposed 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 [4], 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 a representation of motion planning problems under differential constraints that has a number of important advantages:

- Satisfaction of differential constraints is accomplished off-line, to allow fast on-line perfomance,
- Compatibility with standard replanning algorithms allows quick robot reaction to changing environment information,
- Fast planning is enabled through managing the fidelity of representation.

Our concentration on achieving a fast deterministic constraint-compliant planner was originally motivated by the spartan processing available on the Mars rovers. However, our results are equally applicable to terrestrial problems where high planner update rates can be used to respond effectively to changes in dynamic environments as they are predicted or discovered by perception.

In the next section, we relate the proposed approach to prior work. In the following three sections, we further describe each of its principal benefits, as listed above. We will conclude this presentation with experimental results.

II. PRIOR WORK

A planner based on A* search in the state lattice was successfully utilized to guide a car-like robot in challenging natural environments [5]. The graduated fidelity extension to the state lattice is related to the general area of multi-resolution planning: [6], [7], [8] and others. One difference our method has with most multi-resolution predecessors is that we allow regions of different resolution 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 [9], 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 [10] [11], 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 [12]. However, there are important advantages to using discretization in the state space instead. In particular, it simplifies reducing dispersion of sampling, in turn allowing a more uniform distribution of samples in the state space [4]. 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 [16] 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 [14], we are able to choose a convenient discretization in the state space, one that makes the search more efficient, while maintaining the satisfaction of differential constraints.

III. FEASIBLE MOTIONS

Discrete representation of robot state is a well-established method of reducing the computational complexity of motion planning. This reduction comes at the expense of sacrificing *feasibility* and *optimality*, the notions denoting the planner's capacity to compute a motion that satisfies given constraints, and to minimize the cost of the motion, respectively. In computing motions, we seek to satisfy two types of constraints: features of the environment that limit the robot's motion (*obstacles*) and the limitation of the robot's mobility due to the constrained dynamics of its motion (*differential constraints*). Motions that satisfy both types of constraints will be referred to as *feasible* motions.

The proposed method is based on a particular discretization of robot state space, the *state lattice*. It encodes a graph, whose vertices are a discretized set of all reachable states of the system, and whose edges are feasible motions, *controls*, which connect these states exactly. The motions encoded in the edges of the state lattice form a repeating unit that can be copied to every vertex, while preserving the property that each edge joins neighboring vertices exactly. This property of the search space will be denoted *regularity*. The canonical set of repeating edges will be called the *control set*. The number of edges in the control set is exactly the branching factor, *out-degree*, of each vertex in the reachability graph.

A. Sampling State and Motions

Beneficial state sampling policies include regular lattice sampling, where a larger volume of the state space is covered with fewer samples, while minimizing the dispersion or discrepancy [4]. It is natural to extend the concept of regular sampling from individual values of state to sequences of states (i.e. paths). As for state space, the function continuum of feasible motions can also be sampled to make computation tractable. The effective lattice state space sampling, developed in this work, induces a related effective sampling of motions.

Suppose discrete states are arranged in a regular pattern. Besides sampling efficiency benefits, an important advantage of regular sampling of state space is (quantized) translational invariance. Any motion which joins two given states will also join all other pairs of identically arranged states. By extension, the same set of controls emanating from a given state can be applied at every other instance of the repeating unit. Therefore, in this regular lattice arrangement, the information encoding the connectivity of the search space (ignoring obstacles) can be pre-computed, and it can be stored compactly in terms of a canonical set of repeated primitive motions, the control set. Two properties of lattice search spaces that are necessary conditions for satisfying differential constraints are:

1) Enforcing continuity of relevant robot state variables across the vertices,
2) Ensuring that the edges between the vertices of the search space represent feasible motions.

The first condition can be satisfied by adding the relevant dimensions to the search space, in order to represent the continuity of state variables explicitly. For example, if a heading dimension is added to the a 2D (x,y) state space, then $(x, y, North)$ and $(x, y, East)$ become distinct states. In order to satisfy the second condition, we require a method of discretizing the robot control space to force its reachability tree to be a regular lattice in state space. We identify two methods of achieving this:

- *Forward* — for certain systems, there are methods of sampling the control space that result in a state lattice [16], [15],
- *Inverse* — a desired state sampling can be chosen first, and boundary value problem solvers can be used to find the feasible motions (steering functions) that drive the system from one state value to another, e.g. [14], [17].

We prefer the inverse approach because it permits the choice of state discretization to be driven by the application – including the vehicle and the environment. Smaller state spacing is desired for denser obstacles or smaller vehicles. Note that, in the state lattice, if state separations are small relative to the distance required to change vehicle heading by the distance to the next heading sample, the edges in such a structure can span many state separations.

Fortunately, the work of constructing the state lattice can be performed off-line, without affecting planner runtime. Once it is constructed and represented as a directed graph (compactly specified with a control set), the state lattice can be searched with standard algorithms. An example of a simplified state lattice is shown in Figure 2.

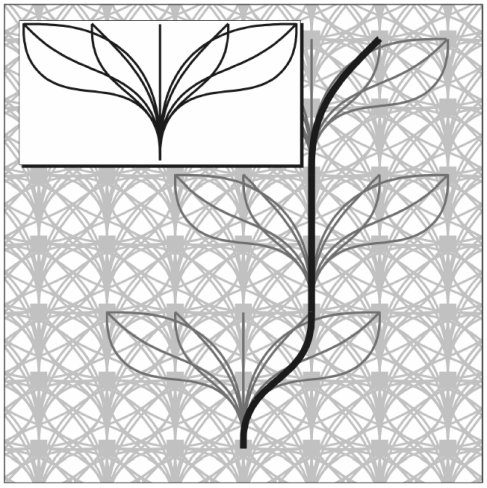

Fig. 2. An Example State Lattice. A repeated and regular pattern of vertices and edges comprises the state lattice. The inset shows the control set, the motions leading to some nearby neighbors of a vertex. The overall motion plan (thick black curve) is simply a sequence of such edges. Reverse motions were omitted for clarity.

Algorithm 1 is a simple inverse method for generating a control set. Referred to as the Shortest Edges algorithm, it may serve as a departing point to evaluate our proposed approach to search space design. To better illustrate the algorithm, in this section we assume a 4D state lattice that consists of 2D translation, heading and curvature. Suppose that Θ and K are user-defined subsets of discrete values of heading and curvature in the state lattice, respectively. By exploiting rotational symmetries in the state lattice, these sets can be desired strict subsets of all possible discrete values of these states variables. The outer for-loop selects the permutations of discrete values of initial and final heading and curvature. The inner for-loop cycles through all discrete value pairs of x and y, such that the maximum norm[1] L_∞ between the origin O and (x_f, y_f) grows from 1 to infinity. For each value of L_∞, if the trajectory generator finds a solution to the boundary value problem, a feasible trajectory u_i, we add it to the control set. At this point we break from the inner for-loop and proceed with another choice of terminal heading and curvature values. The algorithm terminates when a trajectory is generated for every permutation of heading and curvature values.

[1] L_∞ norm of a vector $\mathbf{x} = [x_1, x_2, ..., x_n]$ is $\max_i |x_i|$

Input: State discretization in the state lattice: position, discrete values of heading (Θ) and curvature (K)
Output: A control set, E_x
$E_x = \emptyset$;
foreach $\theta_i, \theta_j \in \Theta$ *and* $\kappa_i, \kappa_j \in K$ **do**
  foreach x_f, y_f *s.t.* $L_\infty(O, [x_f, y_f]) = [1 \ldots \infty)$ **do**
    $u_i =$ trajectory$([0, 0, \theta_i, \kappa_i], [x_f, y_f, \theta_f, \kappa_f])$;
    if $u_i \neq \emptyset$ **then**
      $E_x \leftarrow u_i$;
      break;
    end
  end
end

Algorithm 1: A simple method of generating a control set.

B. Heuristic Cost Estimate

Heuristic estimates of the remaining cost in a partial plan are well-known to have the potential to focus the search enough to eliminate unnecessary computation while preserving the quality of the solution. The Euclidean distance metric is among the simplest options for a heuristic estimate of path length in the state lattice. This function is computationally efficient, and it satisfies the admissibility requirement of A* [18]. However, for differentially constrained planning, it is not a well-informed heuristic and, in the case of short paths, it can vastly underestimate the true path length, resulting in inefficient search. A heuristic for a vehicle with limited turning radius moving in the plane could be derived from the methods of Reeds and Shepp [19]. However, the Reeds-Shepp paths are discontinuous in curvature (i.e. infeasible to execute without stopping), and they do not account for discretization, so even these paths are underestimates. Given ample off-line computational resources, a straight-forward and effective way to predict path lengths is to pre-compute and store the actual cost heuristics that a planner will need, using the planner itself. Such a Heuristic Look-Up Table (HLUT) can be implemented as a database of real-valued query costs. Under this approach, the computation of the heuristic becomes a simple table dereference [20], [21].

IV. REACTIVE REPLANNING

The state lattice search space presented above is compatible with most dynamic programming algorithms. In order to achieve efficient implementation of efficient replanning algorithms (variants of D*), a number of implementation details are presented in this section.

A. Computing Edge Costs

The regularity of the state lattice allows an efficient optimization in evaluation of the cost of graph edges during planning with continuous cost maps, which is roughly equivalent in computational terms to pre-computing C space obstacles. Recall that, in continuous cost field environment models, the cost of a configuration is computed as a cost weighted swept volume (i.e. area in 2D workspace cost fields). That is, the sum of the workspace cell costs occupied by the vehicle volume. We denote the set of map cells occupied by the vehicle volume during execution of a particular motion as the *swath* of this motion. Since lattice edges repeat regularly, so do their associated swaths. Thus, it is possible to pre-compute the swaths for all elements of the control set. When costs change in the workspace cost map, the only computation required to update the cost of an edge (motion) is to add the costs of the cells in the swaths.

The top of Figure 3 depicts a motion of a tractor-trailer vehicle, along with the swath of this motion. In order to evaluate the cost of a motion, the costs of map cells in the swath (reproduced on the bottom of Figure 3) are simply summed up – an operation typically much more efficient than simulating the motion of the system. The simpler alternative of low-pass filtering the workspace cost map by a circular vehicle approximation will be significantly less accurate for systems with elongated shape. The calculation proceeds off-line for a state lattice and we care to satisfy differential constraints, so we use the correct vehicle shape and highly accurate simulation.

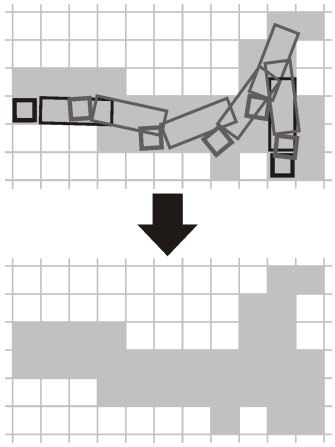

Fig. 3. An example of a pre-computed swath of a path for a tractor-trailer vehicle. Bottom: the swath allows computing the cost of a motion w.r.t. a cost map, without explicitly considering the motion itself (top).

B. Processing Edge Cost Updates in Replanning

D* variants were originally applied to grids [2], [3]. The earliest work on D* used the same resolution for both the cost map and the search space and implicit "edges" which connected states only to their nearest 8 neighbors. In this case, the mapping from a modified map cell to the affected search space edges and vertices is trivial. For a state lattice whose edges may span several map cells, the above historical simplifications of these issues are no longer feasible.

Suppose the replanner uses a priority queue to ensure optimality of the solution. For every change in the cost of the directed edge from the vertex x_i to x_j, $c(x_i, x_j)$, a replanning algorithm requires recomputing the cost of x_j and potentially inserting it into the priority queue. Assuming a map cell $m_{ij} \in \mathbb{N}^2$ changes cost, 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 $Y : \mathbb{N}^2 \rightarrow V_c$.

To develop this mapping, we use the concept of swath, introduced in Section IV-A. More formally, we consider the

swath a set $C_s \subset \mathbb{N}^2$ of cost map cells that are occupied by the robot as it executes a motion. The cost of an edge that represents this motion is directly dependent on the costs of map cells in C_s. Recall that once we pre-compute the control set of a regular lattice, it is 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} (i.e. the set of edges whose swaths pass through the cell). 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 map 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 the set of edges, whose swaths pass through the 2D origin. The Figure 4 illustrates this idea. Like the control set and path swaths, the resulting set of edges can be pre-computed due to the regularity of the state lattice. An example of the V_c for the implementation described in Section VI is shown in the Figure 5.

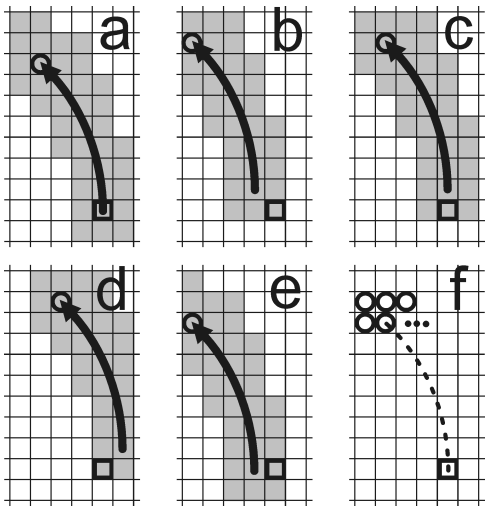

Fig. 4. The first several steps of pre-computing the list of graph vertices that are affected by a change in cost of a map cell. In a), a single element of a control set is chosen for this example. It emanates from the origin of the state lattice, thick square, and connects it to another graph vertex, thick circle. Grey cells are the swath of this motion. Suppose a map cell, located at the origin of the state lattice (thick square), changes cost. We attempt to find all translational versions of the chosen motion, whose swaths are affected by the changed map cell. In the subfigures b) – e), we iterate through several such translational versions of the motion. The resulting (edge end-point) vertices that are considered for insertion to the priority queue are shown in subfigure f). Typically, many more such vertices are processed for each edge (as suggested by ellipsis in subfigure f). The process repeats for all edges in the control set. Pre-computation allows eliminating any redundancy by generating a unique list of such vertices.

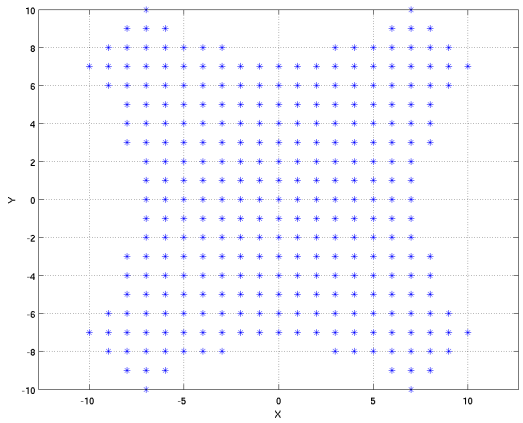

Fig. 5. A 2D projection of an example of V_c, the set of lattice states that are to be re-considered for every updated map cell. The units in the plot are state lattice cells. For the purposes of exposition, here the size of map cells is set to be equal to the size of state lattice (x, y) cells. For each map cell, m_{ij}, that changes cost, we place the set of vertices above in this Figure onto m_{ij} (i.e. the origin of the set of vertices, denoted with coordinates $(0, 0)$, is identified with m_{ij}). Next, we iterate through the depicted list of the vertices and place each one on the priority queue, if it was indeed affected by the cost change of m_{ij}.

V. FAST OPERATION

By virtue of the state lattice's general representation as a directed graph, it can be naturally extended with multi-resolution enhancements. Significant planning runtime improvement was achieved in the literature via a judicious use of the quality of representation of the planning problem, e.g. [22], [7], [8] among others. In field robotics, it is frequently beneficial to utilize a high fidelity of representation in the immediate vicinity of the robot (perhaps within its sensor range), and reduce the fidelity in the areas that are either less known or less relevant for the planning problem. Lower fidelity of representation is designed to increase search speed, while higher fidelity provides better quality solutions. Since grids have traditionally been utilized in replanning, the notion of varying the quality of problem representation has been identified with varying the resolution of the grid. However, our method varies the discretization of both the state and motions. We refer to managing the fidelity of state lattice representation as *graduated fidelity*.

In designing the connectivity of search space regions of different fidelities, care must be taken to ensure that all regions consist of motions that are feasible with respect to the robot's mobility model. If this rule is violated, mission failures become possible due to the differences in the representation of vehicle mobility. Figure 6 illustrates this situation using a simple example. Suppose a search space is used in which a high fidelity region of finite size surrounds and moves with the vehicle, and a disjoint lower fidelity grid is used beyond that. Suppose the A* algorithm is used to plan paths in this hybrid graph. A car-like robot attempts to travel to a goal on the other side of a collection of obstacles that forms a narrow corridor. As long as the low-fidelity region includes the corridor (black line), the planner will find a solution in the graph. However, the 90

degree turn in the path is actually infeasible, since the car-like robot cannot turn in place. As the vehicle moves, the high fidelity region will eventually include the turn in the corridor and the planner will then fail to find a solution. The only viable alternative will be to back up, thereby moving the corridor to the low fidelity region once again. Since the original state of the scenario has now been achieved, it is easy to see that this behavior will repeat forever. In order to avoid such difficulties, it is necessary to ascertain that all levels of fidelity include feasible motions. In particular, the connectivity of low fidelity regions must be a subset of that of the higher fidelity regions.

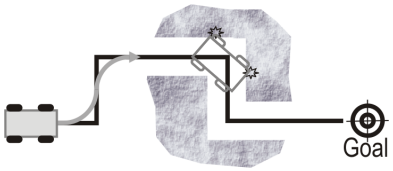

Fig. 6. A simple example of a motion planning problem, where a car-like robot that attempts to follow the infeasible path (black line) experiences a failure.

To implement graduated fidelity planning, the above design requires only a minor modification. Once the state lattice graph is separated into subgraphs of different fidelities as desired, each subgraph uses its own control set to achieve the chosen fidelity. Each control set defines the successors of a vertex being expanded during search. Care must be taken to design the control sets such that they adequately span the boundaries between the subgraphs. Note that control set design is the sole procedure needed to enable graduated fidelity. Replanning algorithms require no changes and will achieve the desired effects automatically.

It can be useful to enable a high fidelity subgraph to move along with the mobile robot as described in the example above. As shown in [23], such flexibility can be accomplished by undoing the effects of previous expansions of the vertices on the perimeter of the moving subgraph. Accomplishing this once again requires no change to the actual replanning algorithm. The change of graph connectivity that occurs between replans is presented to the planning algorithm as a change in cost of the affected graph vertices. Such topology based cost changes appear to replanning algorithms to be identical in nature to perception based cost changes. If the vertex expansion step is considered to be part of an external search space module, the planner actually cannot tell that the graph topology is changing.

More generally, it is straight-forward to extend the concept of graduated fidelity to allow multiple subgraphs of different fidelity to move or change shape between replans. Such flexibility results in a *dynamic search space*, which complements dynamic replanning algorithms to improve planning efficiency. Thus, the graduated fidelity extension of state lattice planning is conceptually simple and straight-forward to implement, and it can be designed to result in significant savings in runtime and memory usage in replanning.

VI. EXPERIMENTAL RESULTS

A differentially constrained motion planner, *lattice planner*, was implemented based on the state lattice and tested in a variety of scenarios, including in simulation and on real robots. The planner was ported to the VxWorks™hard real-time operating system that controls the JPL rover FIDO that was used in field experients. Figure 7 shows the results of a typical experiment with the FIDO running the lattice planner on-board to navigate autonomously amid dense rocks. In this experiment, the rover was given a command to drive to a goal 15 meters directly in front of it, as shown by the black line in the top of the Figure. This motion was infeasible due to large rock formations. However, the rover, under guidance of the lattice planner, negotiated this maze-like and previously unknown environment, and found a feasible path (white dots) to accomplish its mission, despite a very limited perception horizon of 3 meters and $\pm 40^\circ$ field of view.

We have not had a chance to optimize memory usage of our planner implementation; nevertheless, the peak memory usage of the lattice planner over all our experiments with the FIDO rover was less than 100MB. The bottom part of Figure 7 shows the semilog plot of the on-board lattice planner runtime per replan cycle, averaging at approximately 10Hz. This plot serves well to illustrate two points regarding typical planner runtime on-board FIDO: the computation time per replanning operation can vary greatly (depending on the difficulty of the problem at hand), and the replanning runtime was frequently lower than the time-resolution of the rover's operating system (5ms), which is observed via the bottom-limited segments of the plot.

Rover mobility was characterized by a minimum turning radius of 0.5m and a capacity of point turns, which had a high cost due to the time and energy required for reorienting wheels. Both cost map cells and (x, y)-cells of the state lattice were square with 20cm side length; both types of cells coincided in position. The rover used a single 1.6GHz CPU and 512MB of RAM, shared among all processes of the rover, including state estimation, stereo vision perception and communication systems.

The lattice planner utilized two fidelity regions. High fidelity region was square 21x21 mapcells (L_∞-radius of 2 meters), centered around the rover. It utilized a lattice control set with average outdegree 12. Its state space consisted of 2D position and heading (x, y, θ). It was generated using Algorithm 1. A trajectory generator in [17] was used to generate the motions between the given values of robot state. Motions were parameterized as cubic polynomial curvature, κ, functions of path length s, $\kappa(s)$. Low fidelity was represented as eight-connected grid.

In this experiment, the rover traversed approximately 30 meters and achieved the goal successfully (only the first two-thirds of the rover path are shown in the photograph due to the limited field of view of the external camera). No path tracking was used, and the rover executed verbatim the smooth and feasible motion computed by the lattice planner.

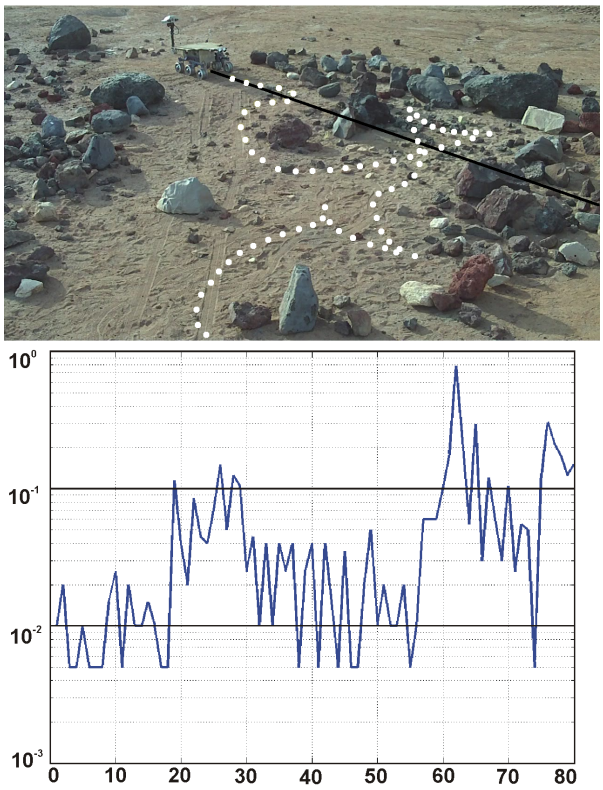

Fig. 7. A field experiment in the JPL Mars Yard. Top: the FIDO rover was commanded to go straight 15 meters (black line). The rover navigated autonomously among previously unknown maze-like obstacles, while running the graduated fidelity lattice planner on-board. White dotted line is the path traversed by FIDO. The rover encountered multiple difficult planning scenarios due to the very limited perception. It traveled approximately 30 meters in order to achieve its goal. Bottom: throughout numerous field experiments, lattice planner on-board FIDO averaged replanning frequency of approximately 10Hz.

VII. CONCLUSIONS AND FUTURE WORKS

In this paper we described an effective approach to planning robot motions that satisfy differential constraints. In addition to leveraging dynamic replanning algorithms, this approach enables dynamic and deliberate changes in search space connectivity 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] D. Ferguson, T. Howard, and M. Likhachev, "Motion planning in urban environments: Part II," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Nice, France, September 2008, pp. 1070–1076.

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

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

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

[5] M. Pivtoraiko and A. Kelly, "Constrained motion planning in discrete state spaces," in *Field and Service Robotics*, vol. 25. Berlin / Heidelberg: Springer, July 2005, pp. 269–280.

[6] 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.

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

[8] D. Pai and L.-M. Reissell, "Multiresolution rough terrain motion planning," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 1, pp. 19–33, 1998.

[9] R. Szczerba, D. Chen, and J. Uhran, "Planning shortest paths among 2D and 3D weighted regions using framed-subspaces," *International Journal of Robotics Research*, vol. 17, no. 5, pp. 531–546, 1998.

[10] D. Hsu, R. Kindel, and J.-C. L. S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *International Journal of Robotics Research*, vol. 21, no. 3, pp. 233–255, 2002.

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

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

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

[14] T. Howard and A. Kelly, "Optimal rough terrain trajectory generation for wheeled mobile robots," *International Journal of Robotics Research*, vol. 26, no. 2, pp. 141–166, 2007.

[15] E. Frazzoli, M. Dahleh, and E. Feron, "Real-time motion planning for agile autonomous vehicles," in *Proc. of the American Control Conference*, 2001.

[16] A. Bicchi, A. Marigo, and B. Piccoli, "On the reachability of quantized control systems," *IEEE Transactions on Automatic Control*, vol. 47, no. 4, pp. 546–563, 2002.

[17] A. Kelly and B. Nagy, "Reactive nonholonomic trajectory generation via parametric optimal control," *International Journal of Robotics Research*, vol. 22, no. 7/8, pp. 583–601, 2002.

[18] J. Pearl, *Heuristics: intelligent search strategies for computer problem solving*. Boston, MA: Addison-Wesley Longman Publishing Co., 1984.

[19] J. A. Reeds and L. A. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pacific Journal of Mathematics*, vol. 145, no. 2, pp. 367–393, 1990.

[20] M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Optimal, smooth, nonholonomic mobile robot motion planning in state lattices," Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-07-15, May 2007.

[21] R. Knepper and A. Kelly, "High performance state lattice planning using heuristic look-up tables," in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, 2006.

[22] R. Bohlin, "Path planning in practice; lazy evaluation on a multi-resolution grid," *Proc. of the IEEE/RSJ International Conference on Intelligent Robots & Systems*, 2001.

[23] M. Pivtoraiko and A. Kelly, "Differentially constrained motion replanning using state lattices with graduated fidelity," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2008.