EFFICIENT CONSTRAINED PATH PLANNING VIA SEARCH IN STATE LATTICES

Mihail Pivtoraiko and Alonzo Kelly

Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213

ABSTRACT

We propose a novel approach to constrained path planning that is based on a special search space which efficiently encodes feasible paths. The paths are encoded implicitly as connections between states, but only feasible and local connections are included. Once this search space is developed, we systematically generate a near-minimal set of spatially distinct path primitives. This set expresses the local connectivity of constrained motions and also eliminates redundancies. The set of primitives is used to define heuristic search, and thereby create a very efficient path planner at the chosen resolution. We also discuss a wide variety of space and terrestrial robotics applications where this motion planner can be especially useful.

Key words: path planning; nonholonomic; differential constraints; lattice; control set; path/motion primitives.

1. INTRODUCTION

Discrete representation of states is a well-established method of reducing the computational complexity of motion planning to manageable at the expense of surrendering strict completeness. Furthermore, such discrete representations complicate the satisfaction of differential constraints which reflect the limited maneuverability of many real vehicles. We propose a mechanism to achieve the computational advantages of discretization while guaranteeing satisfaction of motion constraints.

To this end we introduce a search space, referred to as the *state lattice*, which is the conceptual construct that is used to formulate a nonholonomic motion planning query as graph search. The state lattice is a discretized set of all reachable configurations of the system. It is constructed by discretizing the $\mathcal{C}$-space into a hyperdimensional grid and attempting to connect the origin with every node of the grid using a feasible path, an edge. The lattice in general is also assumed to contain all feasible paths, up to a given resolution, which implies that if it is possible for a vehicle to travel from one node to another node, then the lattice contains a sequence of paths to perform this maneuver. Hence, we conclude that this formulation allows resolution complete planning queries.

Like a grid, the state lattice converts the problem of planning in a continuous function space into one of generating a sequence of decisions chosen from distinct alternatives. Unlike a grid, the state lattice is constructed such that its connections represent feasible paths. A connectivity scheme that intrinsically represents mobility constraints leads to superior motion planning results because no time is wasted either generating, evaluating, or fixing infeasible plans.

To achieve this scheme we attempt to capture *local connectivity* of the state lattice, within a limited neighborhood of any node. We discuss designing a minimal set of primitive paths, which, when placed in a sequence, can re-generate any other path in the lattice. We further show that this formulation lends itself directly to building an efficient search algorithm.

2. PRIOR WORK

The utility of the lattice is hinged on the assumption that it is possible to determine a feasible path between two given configurations in $\mathcal{C}$-space without obstacles. While this is itself a very difficult problem, it has been the objective of much research in the past century. Frazzoli et al. in (6) suggest that there are many cases where efficient, obstacle-free paths may be computed either analytically or numerically by solving an appropriate optimal control problem. A fast nonholonomic trajectory generator was described in (3). It generates polynomial spiral trajectories, such that a path is specified by a continuous control function: curvature as a function of path length.

It was shown in (9) 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, and under most quantizations the vertices of the reachability graph are unfortunately dense in the reachable set. By using an inverse path generator, we can choose a con-

venient discretization in control and state space, one that makes the search more efficient. This also allows us to use continuous control functions that are natural for real systems.

The importance and difficulty of enforcing differential constraints also has a long history (1), (2), (8). A recent trend appears to favor more deterministic variants of the PRM (4). In (5), Quasi-PRM and Lattice Roadmap (LRM) are introduced by using low-discrepancy Halton/Hammersley sequences and a regular lattice, respectively, for sampling. LRM appeared especially attractive due to its properties of optimal dispersion and near-optimal discrepancy.

Also, a "Lazy" variant of these methods was discussed that avoided collision checking during the roadmap construction phase. In this manner the same roadmap could be used in a variety of settings, at the cost of performing collision checking during the search. An even "lazier" version is suggested, in which "the initial graph is not even explicitly represented" (5). In this regard, our approach of using an implicit lattice and searching it by means of a pre-computed control set that only captures local connectivity is very similar to the Lazy LRM. Our contribution is in exploring the conjecture made in that work and successfully applying it to nonholonomic motion planning.

Initial concepts of this work were explored in a successful field implementation of a nonholonomic motion planner built using the state lattice of limited size represented explicitly (7).

3. STATE LATTICE

In this section we describe the state lattice as a generalization of a grid and formulate it as a search space for efficient constrained motion planning as heuristic search.

3.1. Inverse Path Generation

A number of good methods of inverse trajectory generation exist, as this area received considerable attention previously. Since in our case we are mostly interested in generating paths in $\mathcal{C}$-space, the method described in (3) lent itself very well for our purpose. The assumed form of the solution path in that inverse path generator is a curvature polynomial of arbitrary order. The method allows optimization w.r.t. various criteria, e.g. least curvature variation. Its continuous specification of paths is convenient to manipulate and execute in vehicle controllers. The method executes practically in real-time: a query is computed in about 1 millisecond.

3.2. Constructing the State Lattice

The state lattice that we develop here can be viewed as a generalization of a grid. It adds connections to the grid: if there is a feasible path between any two discretized state values (lattice nodes), then they are connected with a lattice edge corresponding to that path. Here state discretization plays an important role. It converts the motion planning problem into a sequential decision process.

While the state vector can have any dimension, we illustrate the discussion by considering path planning for wheeled ground vehicles. Each lattice node therefore represents a 4-dimensional *state* that includes 2D position, heading and curvature.

Typically a chosen state discretization exhibits a high degree of regularity, such that the spatial relationships between two given states will reoccur often due to the existence of other identically arranged pairs of states. Regular state discretization leads to a set of motion options which is similarly regular. For the balance of the paper, we adopt the assumption that the state space discretization is regular in at least the translational coordinates (x, y). Specifically, if the path between two postures:

$$[x_i, y_i, \theta_i, \kappa_i] \rightarrow [x_f, y_f, \theta_f, \kappa_f]$$

is feasible, then so is the path

$$[x_i{+}n\Delta_l, y_i{+}n\Delta_l, \theta_i, \kappa_i] \rightarrow [x_f{+}n\Delta_l, y_f{+}n\Delta_l, \theta_f, \kappa_f]$$

for any integer n and (x, y)-discretization step size Δ_l. While the starting and ending states for two such paths are distinct, the motion itself (perhaps encoded as a steering function) is not.

With these properties in mind we construct the state lattice by using the inverse path generator to find paths between any node in the grid and the arbitrarily chosen origin. Fig. 1 illustrates lattice construction for the Reeds-Shepp car. By regularity, we can copy the resulting set of feasible paths to any node in the lattice. In the limit, as the lattice is built by including all feasible motions from any point, it will approach the reachability graph of the vehicle, up to a chosen resolution. Thus, we come to a conclusion that the state lattice is a valid representation (discretization) of the system's reachability graph.

3.3. Path Equivalence

It is consistent with state discretization to consider discretizing paths through space. We consider two paths (with identical endpoints) which are "sufficiently" close together to be *equivalent*. We define a path τ_1 to be equivalent to τ_2 if τ_1 is contained in a certain region Q around τ_2. Q is defined as a set of configurations within a certain distance δ_e of τ_1, given some distance metric ρ:

$$\forall q \in \tau_1, \forall q' \in \tau_2, Q = \{q' | \rho(q', q) < \delta_e\} \qquad (1)$$

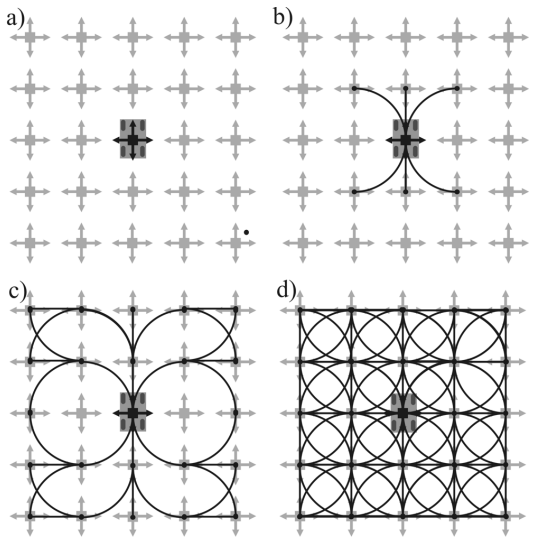

Figure 1. Constructing the lattice for the Reeds-Shepp Car. In a) we define a discretization in C-space (an (x, y) grid is chosen here, arrows indicate allowed headings), an origin is chosen; b) for 8 neighbor nodes around the origin, feasible paths are found; c) same query is extended outward to 24 neighbors, only a few direct paths are shown; d) complete lattice.

All paths that satisfy this criterion are considered to belong to the same equivalence class (Fig. 2). Path equivalence is central to our method because it is utilized to simplify the search space considerably without the loss of path representation. In other words, by this property the search space of paths does not need to contain an infinity of feasible paths between two given states q_i and q_f, but only the non-equivalent paths (a much smaller subset). Moreover, in further discussion we will see that path equivalence can be extended to require only direct paths between q_i and q_f to be included in the lattice, in which case the aforementioned infinity of paths can be substituted with a finite (and small) set of non-equivalent (spatially distinct) paths.

This reasoning is consistent with applications in robotics. Typically, there is a certain error of path following for realistic robotic mechanisms. By utilizing path equivalence to dramatically reduce the search space of paths, we essentially exploit this error to make path planning more efficient. This can be viewed as an approximation that does not cost anything in terms of quality of planning, as it is based on behavior of real vehicles.

4. CONTROL SET

The state lattice that was constructed in the previous section is assumed to be inifinite in extent and to contain all feasible non-equivalent paths. In order to arrive at a tractable search space of paths, we present a method to utilize path equivalence to capture local connectivity of the reachability graph of the system. We present a principled method for choosing the neighborhood of the lattice that both offers practical guarantees of complete exploration of the lattice and keeps the neighborhood size small to preserve search efficiency.

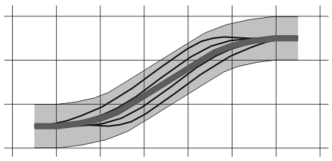

Figure 2. Path Equivalence. A variety of paths between two configurations (thin black lines) that are contained in the boundary (grey region) are considered to be equivalent and represented by a canonical path (thick gray line).

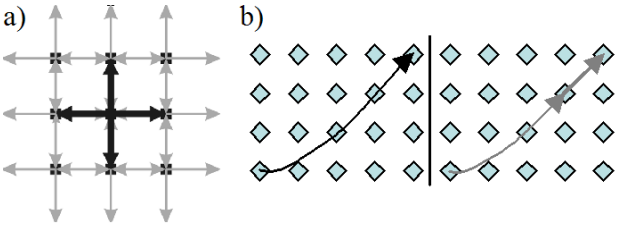

Figure 3. a) Isolating a minimal set of paths in a 4-connected grid. b) An illustration of path decomposition. Blue diamonds are lattice nodes; the black curve on the left is some arbitrary path (that starts and ends on nodes). On the right, we show that it can be decomposed into two (grey) smaller paths that meet at another node.

4.1. Path Decomposition

Path planning using heuristic search in 4- and 8-connected grids is widely accepted as very efficient. For example, in the rover navigation application, long-range path planning is often done using grid search. For motivation and illustration of this section, we look at the details of grid search and particularly focus on the aspects that make it so efficient.

The regularity property of the grid implies that it is possible to isolate a certain representative set of connections. This set can be used to obtain the original grid connectivity by copying the set of connections to every node. Fig. 3a illustrates the case of a 4-connected rectangular grid, where it is easy to identify the minimal set of connections. The grey lines in this figure represent all possible paths in the grid. Four thick black paths in the center constitute the minimal set of paths. Any path through this grid can be decomposed into a sequence of paths from the minimal set, referred to as “primitives”. If we cast the grid in the context of motion planning, we understand that this minimal set enables us to generate arbitrarily long motion plans in the infinite grid. This concept has been used in motion planning for some time (1).

As was mentioned, the concept of the state lattice was derived from the grid. If we could identify such a control

set for the state lattice as we did for the grid, we would effectively crystallize the entire infinity of paths in the lattice into a finite and small set of primitives.

In order to achieve this goal, we develop the concept of *path decomposition* as follows. By invoking the notion of path equivalence and some $\delta_e > 0$, we can replace a path with two other paths such that their concatenation generates another path that is equivalent to the original path. We define path decomposition as the problem of finding two such constituents of a path (Fig. 3b).

The discretization property of the lattice requires that the two constituent paths meet at a lattice node, or in other words, the point of path's decomposition must be a node. Intuitively, the longer a path is, the more lattice nodes it comes "close" to, hence the easier it is to find a decomposition because there are more "opportunities" to do so.

4.2. Generating the Set of Path Primitives

Simply put, the value of generating the primitives set is to eliminate redundancies of the lattice both in terms of the variety of paths between nodes (through the notion of path equivalence), and in terms of generally unlimited path length (path decomposition). In this section we will present a systematic procedure to generate this set and discuss its convergence, including the necessary criteria for it to arrive at a finite set of primitives.

The procedure for primitive set generation is based on structured elimination. First, the lattice nodes with Manhattan distance of 1 (in translational coordinates) are considered. All paths between the origin and these nodes are analyzed. Then, the nodes with the distance of 2 are chosen, etc., thus moving radially outward. When a path between the origin and a lattice node is considered, it is tested for passing sufficiently close to any other node along its way, and such nodes are tested as points of path decomposition. The decomposition test succeeds when two conditions are satisfied: 1) there are feasible paths between the origin and this intermediate node, and between the node and the destination, and 2) the concatenation of the two *subpaths* is equivalent to the original path. As soon as a point of decomposition is found, the original path his abandoned because its subpaths are already present in the set of primitives (if they need to be), since we consider nodes radially outward. If, however, a path cannot be decomposed, it is added to the set of primitives. This process terminates at a certain radial distance from the origin when all paths at that distance can be decomposed. A listing of this algorithm is given in the Figure 4.

On line 08, we verify that a path between the origin and a chosen lattice node is feasible. This could be done either by looking for the corresponding connection in the state lattice or using the path generator (that would have been used for constructing the lattice). The function on line 12 steps along the path in small increments. This way the algorithm looks for candidate points of decomposition, and

```
01 PrimSet = empty
02 R = 0
03 Qi = Lattice Origin
04 do {
05   R += 1
06   FoundAtLeastOnePath = false
07   for each node Qf s.t. ManhattanDistance(Qi, Qf)==R
08     OrigPath = FindInLattice(from Qi to Qf)
09     if(OrigPath found)
10       qs = Qi
11       do {
12       qs = GetNextStateSampleAlongPath(OrigPath)
13         Nn = GetNearestLatticeNode(qs)
14         SubPath1 = FindInLattice(from Qi to Nnearest)
15         SubPath2 = FindInLattice(from Nnearest to Qf)
16         if( (SubPath1 found || SubPath2 found) &&
17             (MainPath == (SubPath1 + SubPath2) )
18           break // goto 07: choose new OrigPath
19       } while (qs != Qf)
20       Add OrigPath to PrimSet // no decomposition
21       FoundAtLeastOnePrimPath = true
22     end if
23   end for
24 } while (FoundAtLeastOnePrimPath)
```

Figure 4. Primitive path set generation algorithm.

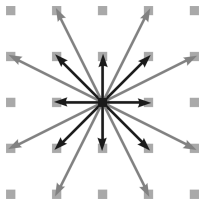

Figure 5. An illustration of special heading discretization that allows considering straight lines as much as possible. Black arrows show 8 equal heading intervals, grey arrows represent finer, non-uniform discretization: 8 additional intervals chosen such that the grey arrows end on lattice nodes as well.

checks them on lines 16-17: if the subpaths are feasible and their concatenation is equivalent to the original path, then the decomposition is successful, and the algorithm moves to the outer loop. FoundAtLeastOnePrimPath is a flag that indicates whether any primitives were added at this value of radius R. If not, then all paths at this radius have been decomposed, and the algorithm terminates.

4.3. Implementation Details

It is worthwhile to note several particularities of implementation that we found useful in verifying the algorithm. Although a uniform discretization of heading is an option, it turned out to be better to discretize heading in a non-uniform manner. The motivation for this was to enable the planner to produce straight-line paths as much as possible. For example, if the goal is a node that is to the left two nodes and up one node from the origin, then a vehicle can get there in a straight line if its heading is $\arctan(1/2) = 26.6°$. Therefore, we define 8 equal heading intervals of 45°, and also 8 non-uniform intervals chosen such that there can be straight paths from the origin to all nodes within the Manhattan radius of three around the origin. Certainly, increasing this radius would

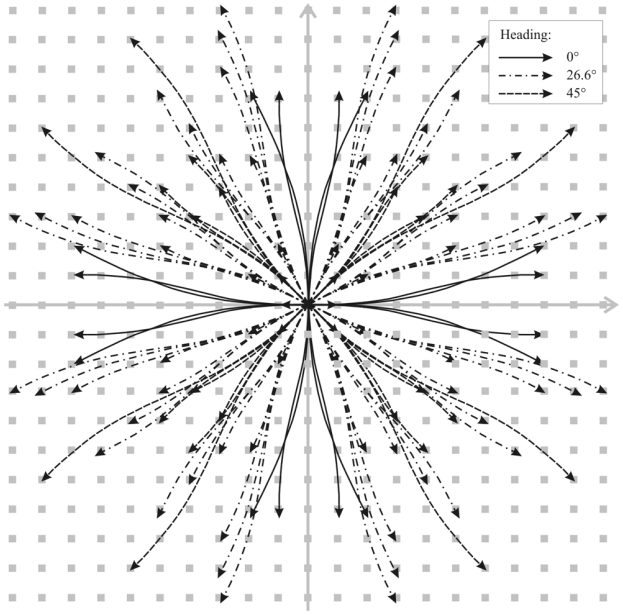

Figure 6. An example set of primitive paths. Only three sets of paths with initial heading of $0°$, $26.6°$ *and* $45°$ *are specified; all others are obtained by reflection around x- and y-axes, and the two diagonals.*

result in finer heading resolution and increase consideration of straight paths. However experimentally we concluded that the added computational cost exceeds the gain of the radii greater than three. Fig. 5 shows our heading discretization: black lines indicate the uniform part, and grey lines show the non-uniform part.

Motion alternatives were expressed as polynomial steering (curvature) functions of path length, of cubic order:

$$\kappa(s) = a + bs + cs^2 + ds^3$$

Such functions possess exactly the 5 degrees of freedom required to join any two poses with arbitrary initial and terminal curvatures (3).

4.4. Results of Primitive Set Generation

An example control set that results is depicted in Fig. 6. Note that the shortest paths (straight up of length 1 and similar) are present in the control set, but not immediately visible.

The primitive set generation algorithm is guaranteed to converge for any distance between the nodes of the lattice, Δ_l, and state discretization, given that the path's minimum turning radius (reciprocal of robot's maximum curvature, mechanically limited or arbitrated for point-turn vehicles based on velocity, wheel slip, etc.) is greater than Δ_l. This is certainly a valid assumption for real robotic vehicles, since it is natural to relate Δ_l to the cell size of the robot's perception map. A geometrical explanation of convergence is based on the observation that, given above assumptions, it is impossible for a path to wind through the lattice nodes and evade every one of them. The path will eventually become decomposable at a finite distance (perhaps large).

The length of the largest primitive in the set is denoted primitive set radius R_{ps}. Another important parameter for characterizing the set is the *outdegree* – the maximum number of primitives emanating at any heading and curvature. It was verified experimentally that for the realistic values of vehicle minimum turning radius and Δ_l (chosen to be equal to typical perception map resolution and modeled after vehicles used in (7)), both R_{ps} and the outdegree are quite small. In the example shown in Fig. 6, R_{ps} was 20, and outdegree was 9. Clearly, this set of primitives is very small, only slightly larger than that for the grid search. Thus, we can define motion planning as heuristic search using this set of primitives and can expect it to be nearly as efficient as grid search, with an important benefit that the resulting paths naturally will comply with whatever kinematic model we chose for generating this primitive set.

5. MOTION PLANNING USING SETS OF PRIMITIVE PATHS

Even though creating a small set of path primitives is central to our method, let us recall that the search space for the path planner is the state lattice. The set of primitives is a finite representation of the lattice (an approximation based on non-zero path following error of real vehicles).

A cost-map can be overlaid on the lattice to represent obstacles or other criteria with respect to which we want to find an optimum obstacle-free motion plan (energy, slope hazard, etc.). We also assume a path sampling procedure that returns the cost of traversal of a path given the cost of cells spanned by the path. The control set is the neighborhood that is expanded when a particular lattice node is considered. Its minimality property that we were seeking is important to making the search in the lattice efficient. We should note that by virtue of containing all feasible motions, the lattice is a cyclic graph. Any standard systematic heuristic graph search algorithm can be applied. In this manner the state lattice can be considered a "lazy" roadmap, in which the cost of traversal is calculated during the search.

5.1. Estimating the Search Heuristic

A key component of heuristic search (e.g. A*) is estimating the heuristic, an estimate of the cost to travel from any node in the lattice to the goal. The result of the search depends heavily on the quality of the heuristic estimate. We first discuss basic methods for estimating the heuristic and arrive at an approach based on pre-computing the exact cost of travel in the vicinity of the origin and storing it in a look-up table (LUT). The cost values are used for

heuristic estimates, and since they are exact values, the search speeds up considerably.

To orient this discussion, let us first consider a simple heuristic estimate. Given initial and final states, $Q_i = (x_i, y_i, \theta_i, \kappa_i)$ and $Q_f = (x_f, y_f, \theta_f, \kappa_f)$, a natural estimate of the path length for a nonholonomic vehice must include the difference in both translational and rotational coordinates. Thus, a heuristic can be a combination of Euclidean distance, L_2, and a weighted heading error.

$$h = L_2((x_i, y_i), (x_f, y_f)) + w_\theta(\mid \theta_f - \theta_i \mid) \quad (2)$$

where w_θ is heading error weight.

In order for the heuristic to be admissible, it must be an under-estimate of the true cost. The drawback of the simple heuristic above is that for some values of w_θ, the heuristic may over-estimate. We propose a LUT-based method of heuristic estimation that is as efficient yet more accurate. It requires a pre-processing step to create the heuristic LUT. The table is simply a compilation of the costs to travel from the origin to all lattice nodes in its vicinity. These costs are determined by running the planner for each possible (Q_i, Q_f) pair using simply Euclidean distance as heuristic, which is guaranteed to be an under-estimate.

Certainly, there are limitations to the size of the LUT. However, the effect of difference in heading and curvature in Q_i and Q_f is significant only in the close vicinity of the vehicle. For path destinations that are beyond several minimum turning radii around the robot, the Euclidean distance is a fairly good heuristic. Thus, in our implementation we limited the size of the heuristic LUT to 3 minimum turning radii. For paths with Q_f beyond that, Euclidean distance was used.

5.2. Path Planner Results

Here we present the timing results of the A*-based motion planner implemented using this approach. We undertook a simulation study of the planner by generating random obstacles and choosing random initial and final configurations. It was confirmed that the planner built using the primitive set generated in Section 4.4 for the robots in (7) performs as efficiently as basic grid search. In fact, for simpler path planning queries, our method performs even better than grid search. This is very exciting, since this method generates optimal nonholonomic paths with no post-processing, yet can perform better than the classical grid search, the archetype of efficiency in path planning. We believe that the primary reason for this significant speed-up is the fact that primitive paths could span multiple grid cells, such that by choosing a primitive, the planner may "jump" ahead, while grid search still considers one cell after another. It is also important to note that this efficiency is due in no small part to the accurate estimate of the heuristic using the pre-computed table. The heuristic guides the search algorithm well and avoids unnecessary branching.

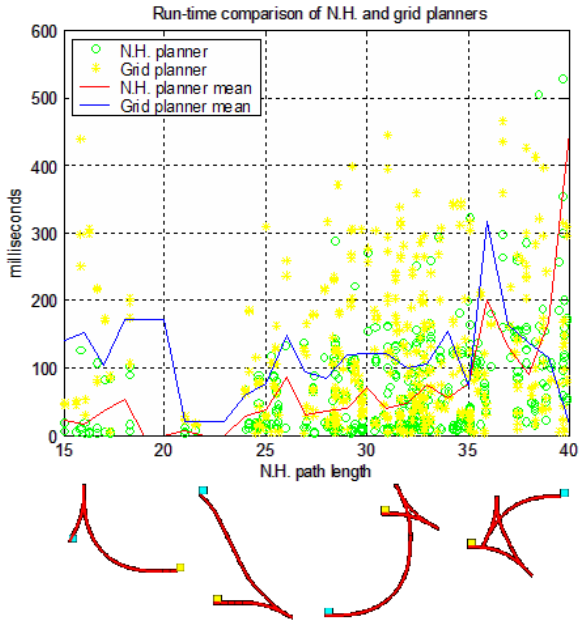

Figure 7. Run-time results of our nonholonomic (N.H.) path planner in comparison with basic grid search. Vertical axis is the time of plan generation, and horizontal axis is the length of the nonholonomic paths. Example paths below illustrate increasing path planning complexity for higher values along horizontal axis.

In Fig. 7 we present the timing results of our planner by considering the toughest local planning scenarios: the final state (goal) is close to the initial state and exhibits significant change in heading and direction to goal. Figure 7 shows both the results of over 1000 timing experiments for both nonholonomic path planner and grid search. For each experiment, a goal Q_f was chosen randomly such that the Euclidean distance between initial state and goal was the same. In this manner, the grid search had roughly the same amount of work to do, whereas nonholonomic path planner's job could vary significantly depending on changes in orientation between initial and final states. The length of the resulting nonholonomic plan is roughly indicative of that complexity, and so the horizontal axis (in units of cell size) is intended to capture this. The increasing complexity from left to right is also illustrated with sample planner solutions (yellow box is initial, and blue box is goal state). Two solid lines in the Fig. 7 are average values of runtime of both planners per particular value of "path complexity". Even though the raw datapoints portion of the plot is very busy, the average runtime lines clearly show that nonholonomic path planner generally takes less time. The balance tilts in favor of grid search only at the right-most end of the horizontal axis, i.e. for highest planning complexity. Nevertheless, our path planner is clearly extremely efficient and can compute most planning queries in less than 100ms, which deems it useful for real-time applications.

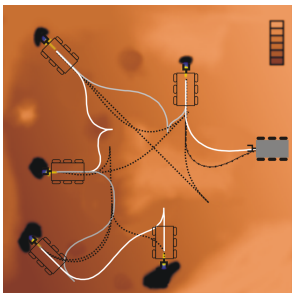

Figure 8. An illustration of applying the presented path planner to a successful solution of the rover instrument placement task.

6. APPLICATIONS

A path planner based on the state lattice concept was successfully implemented on the DARPA PerceptOR program (7). This planner guided a car-like all-terrain vehicle in its exploration of natural, often cluttered, environments. The proposed planner exhibited great performance as a special behavior that was invoked to guide the vehicle out of natural cul-de-sacs.

Another important application for which the presented motion planner is suited very well is rover navigation for space exploration. The rover instrument placement task is known to be a difficult problem both from the standpoint of motion planning and execution. The significant communication time lag is an important consideration prompting quick progress in rover autonomy. Very rough terrain and considerable wheel slip on loose terrain require an approach that can consider the model of rover motion as accurately as possible, as well as take into account the peculiarities of the terrain as it is being discovered.

Our method of motion planning is well suited for this application because it addresses all of the above issues. The inverse trajectory generator used in this approach (3) can use any kinematic rover model whatsoever, and therefore any generated path is inherently executable by the rover under consideration. The flexibility of using any cost-map, overlaid over the state lattice (implicitly represented through using control sets), enables this planner to consider an arbitrary definition of obstacles in terms of the map cost: both binary (e.g. rocks) and variable (e.g. slopes as high-cost, yet traversable). Moreover, dynamics analysis can be made to "label" regions of steep slopes or very loose terrain as untraversable.

An added benefit to specifying paths as curvature functions is the possibility to define velocity planning quite easily. By defining a maximum desirable angular velocity of a vehicle, it is straight-forward to compute the maximum translational velocity as a function of path curvature. In this fashion, our path planner can become a trajectory planner with this simple velocity planning post-processing step.

7. CONCLUSIONS AND FUTURE WORK

This work has proposed a generative formalism for the construction of discrete control sets for constrained motion planning. The inherent encoding of constraints in the resulting representation re-renders the problem of motion planning in terms of unconstrained heuristic search. The encoding of constraints is an offline process that does not affect the efficiency of on-line motion planning.

Ongoing work includes designing a motion planner based on dynamic heuristic search which would allow it to consider arbitrary moving obstacles, the extension of trajectory generation to rough terrain, and hierarchical approaches which would allow efficient constrained path planning for kilometers of traverse.

REFERENCES

[1] Latombe J-C (1991) Robot motion planning. Kluwer, Boston

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

[3] Kelly A and Nagy B (2003) Reactive nonholonomic trajectory generation via parametric optimal control. Int. J. of Robotics Research 22:583-601

[4] LaValle S, Branicky M and Lindemann S (2004) On the relationship between classical grid search and probabilistic roadmaps. Int. J. of Robotics Research 23:673-692

[5] Branicky MS, LaValle S, Olson S, Yang L (2001) Quasi-randomized path planning. In: Proc. of the Int. Conf. on Robotics and Automation

[6] Frazzoli E, Dahleh MA, and Feron E (2001) Real-time motion planning for agile autonomous vehicles. In: Proc. of the American Control Conference

[7] Kelly A et al. (2004) Toward reliable off-road autonomous vehicle operating in challenging environments. In: Proc. of the Int. Symp. on Experimental Robotics

[8] Laumond J-P, Sekhavat S and Lamiraux F (1998) Guidelines in nonholonomic motion planning. In: Laumond J-P (ed) Robot motion planning and control. Springer, New York

[9] Pancanti S et al. (2004) Motion planning through symbols and lattices. In: Proc. of the Int. Conf. on Robotics and Automation

[10] Scheuer A, Laugier Ch (1998) Planning sub-optimal and continuous-curvature paths for car-like robots. In: Proc. of the Int. Conf. on Robotics and Automation

[11] Wang D, Feng Q (2001) Trajectory planning for a four-wheel-steering vehicle. In: Proc. of the Int. Conf. on Robotics and Automation