Constrained Motion Planning in Discrete State Spaces

Mihail Pivtoraiko and Alonzo Kelly

Robotics Institute, Carnegie Mellon University
mihail@cs.cmu.edu, alonzo@ri.cmu.edu

Summary. We propose a principled method to create a search space for constrained motion planning, which efficiently encodes only feasible motion plans. The space of possible paths is encoded implicitly in the connections between states, but only feasible and only local connections are allowed. Furthermore, we propose a systematic method to generate a near-minimal set of spatially distinct motion alternatives. This set of motion primitives preserves the connectivity of the representation while eliminating redundancy – leading to a very efficient structure for motion planning at the chosen resolution.

Key words: nonholonomic motion planning lattice control set

1 Introduction

Discrete representation of states is a well-established method of reducing the computational complexity of planning at the expense of reducing completeness. However, in motion planning, 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 satisfying 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, using an inverse trajectory generator. 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, it is possible to 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 carefully constructed such that each connection represents a feasible path. 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 small *control set*, a minimal set of primitive paths that, when concatenated, 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 any two configurations in a 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 [3] 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 [10]. 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 [6] 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 convenient 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], [9], [5]. A recent trend appears to favor more deterministic variants of the PRM [11]. In [2], 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" [2]. In this regard,

our approach of using an implicit lattice and searching it by means of a precomputed 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 [4].

3 State Lattice

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

3.1 Inverse Path Generation

Among several approaches discussed in Section 1.1 that allow finding a sequence of controls from a given initial configuration to a final configuration, we evaluated the one described in [10]. This approach allows fast generation of nonholonomic trajectories. The assumed form of the solution path is a curvature polynomial of arbitrary order. The method was shown to provide good results and in principle allows optimization w.r.t. various criteria, e.g. least curvature variation. The continuous specification of paths was 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

Discretization is central to defining the state lattice as a generalization of a grid. Discretization converts the motion planning problem into a sequential decision process. We adopt the typical strategy of assuming that decisions are made only at discrete states. The states are the nodes in the lattice and the motions that connect the states are the edges. While the state vector can certainly have arbitrary dimension, for this paper we have implemented the state lattice in 4 dimensions. Each node of the lattice therefore represents a 4-dimensional *posture* that includes 2D position, heading and curvature.

If the discretization exhibits any degree of regularity, then the spatial relationships between two given states will reoccur often due to the existence of other identically arranged pairs of states. A discretization exhibiting some degree of regularity 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:

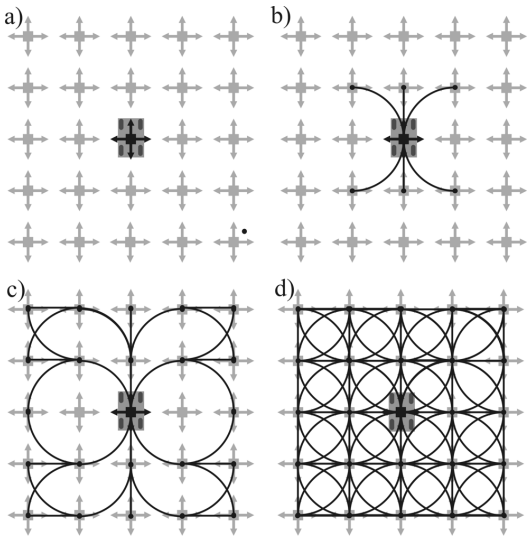

Fig. 1. Constructing the lattice for the Reeds-Shepp Car. In a) we define a discretization in C-space (an (x, y) grid is chosen here, arrows indicate allowed headings), an origin is chosen; b) for 8 neighbor nodes around the origin, feasible paths are found; c) same query is extended outward to 24 neighbors,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

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

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 motions 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. Without loss of generality, we henceforth consider the state lattice to be a valid representation of the system's reachability graph.

Since we discretize state space, it is consistent to consider discretizing paths through space in a similar fashion. 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, defined as a set of configurations within a certain distance δ_e of τ_1, given some metric ρ:

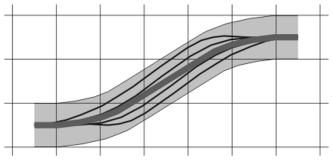

Fig. 2. Path Equivalence. A variety of paths between two configurations (thin black lines) that are contained in the boundary (grey region) are considered to be equivalent and represented by a canonical path (thick gray line).

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

All paths that satisfy this criterion are considered to belong to the same equivalence class (Fig. 2). It is important to note that this definition of path equivalence is consistent with applications to mobile robotics. Typically, there is a certain error of path following for realistic vehicles. By exploiting this error, motion planning can be made more efficient, as presented below.

4 Control Set

Here we present a principled method of choosing the neighborhood of the lattice that both offers practical guarantees of best exploration of the lattice and keeps the neighborhood size small to preserve search efficiency. We strive to obtain the minimal set of controls as motion alternatives that captures local connectivity of the lattice.

4.1 Path Decomposition

With the insights obtained in Section 2.2, we again look at the lattice as a concept derived from a grid. The regularity property of the grid implies that it is possible to isolate a certain representative set of connections which is repeated everywhere in the grid. As is illustrated in Fig. 3a, for the case of a 4-connected rectangular grid, 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 "primitives", paths in the minimal set. 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].

In a similar fashion, if we could identify such a control set for a lattice, we could use it to address the computational issues mentioned above and essentially create a finite representation of the lattice.

By invoking the notion of path equivalence class and some $\delta_e > 0$, we can substitute a path with two other paths such that their concatenation generates

a motion that belongs to the same equivalence class as the original path. We define path decomposition as the problem of finding two such constituents of a path (Fig. 3b).

By definition of path decomposition, the two constituent paths must meet at a lattice 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. Through a simulation study we concluded that it is possible for realistic vehicle parameters (κ_{max} and δ_e) to decompose the entire assumed infinity of motions in the state lattice, of arbitrary length, in this manner. We considered over 2000 different (relatively long) paths in the lattice and showed that all of them could be decomposed into at least two (usually more) smaller paths. Thus, the control set allows us 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).

4.2 Generating the Control Set

Given a method to generate the set of distinct feasible paths to a single state, the control set can be generated by a process of structured elimination. First, paths to all states one unit from the origin are generated, then, paths to all states two units from the origin, etc. When a path is considered, it is tested for passing sufficiently close to an intermediate state, and if so, it is removed from the control set because it can be decomposed into the path to this state from the origin and the path from this state to the end-point. Since we are moving radially outward, any path that can be decomposed may be removed from the control set because its "ingredients" have already been considered.

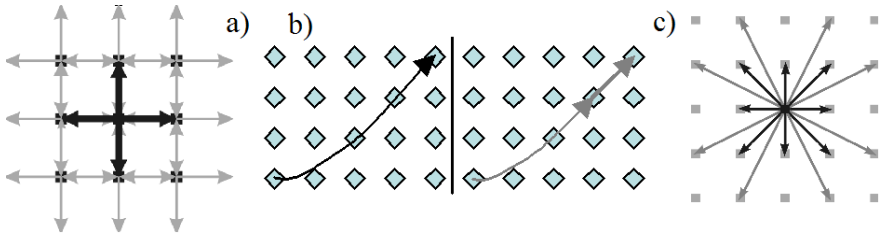

Fig. 3. a) Isolating a minimal set of paths in a 4-connected grid. b) An illustration of path decomposition. Blue diamonds are lattice nodes; the black curve on the left is some arbitrary path (that starts and ends on nodes). On the right, we show that it can be decomposed into two (grey) smaller paths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

Each of them is either in the control set already or does not need to be because it itself is decomposable.

This process terminates at the certain radial distance from the origin when all paths at that distance can be decomposed. Through simulation studies similar to the one mentioned in Section 3.1 we verified that this termination condition is a good heuristic for obtaining a control set that spans the entire state lattice.

Although a uniform discretization of heading is an option, we found it useful to discretize heading in the non-uniform manner. The motivation for this is 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 radius of two nodes around the origin. Certainly, increasing this radius would 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 two. Fig. 3c 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 cubic order:

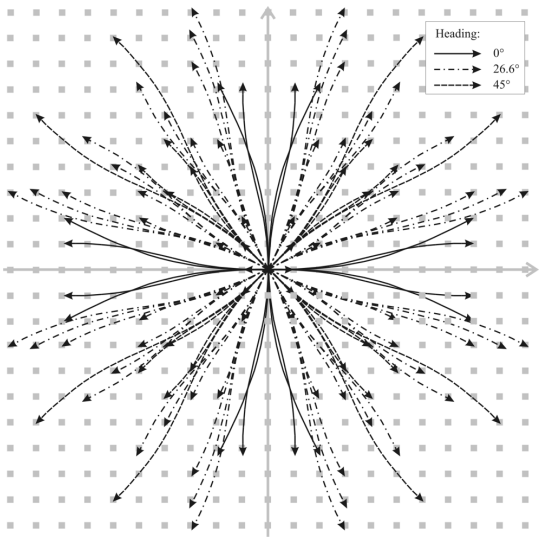

Fig. 4. An Example Control Set. Only three sets of paths with initial heading of 0, 26.6 and 45° are specified; all others are obtained by reflection around x- and y-axes, and the two diagonals.

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

Such functions possess exactly the 5 degrees of freedom required to join any two poses with arbitrary initial and terminal curvatures [10]. An example control set that results is depicted in Fig. 4. Note that the shortest paths (e.g. straight up of length 1) are present in the control set, but not immediately visible.

5 Motion Planning Using Control Sets

The state lattice now possesses the properties necessary to express nonholonomic motion planning as graph search. 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 can be viewed as 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 roadmap, in which the cost of traversal is considered during the search.

5.1 Estimating the Search Heuristic

A key component of heuristic search (e.g. A* implemented for this paper) is calculating the heuristic, an estimate of the cost to travel from any node in the lattice to the goal. We begin with the discussion of the issues involved in estimating the heuristic for nonholonomic vehicles and arrive at a heuristic estimation scheme, based on pre-computing a look-up table, that produces very accurate estimates, thereby improving search efficiency considerably.

The heuristic must not be an over-estimate of the true cost in order for it to be admissible (i.e. in order to guarantee that the search algorithm will find the optimal path). Ideally, we would like it to be exactly equal to true cost, such that the search can be correctly guided toward the goal. Typically, it is impossible to use standard distance metrics, e.g. Euclidean, to estimate the nonholonomic heuristic because depending on change in heading from start to goal and on direction to goal, the vehicle may have to execute an n-point maneuver. Consider re-orienting a car 180°: translation can be negligible in Euclidean sense, but the overall length of the maneuver is significant. Using Euclidean distance for such local plans would result in a gross under-estimate of the cost, such that the behavior of A* would approach that of breadth-first search, with an accompanying performance decrease.

It is important to observe that these issues are primarily relevant for local planning. When the distance to goal is much larger than the minimum turning radius of the vehicle, the under-estimation error percentage of Euclidean distance will become small, thus making this metric a viable heuristic option. We this in mind, we propose a hybrid approach to calculating the heuristic: in the close vicinity of the robot, a local estimation procedure that considers the vehicle's kinematic model is used, whereas in the far range Euclidean distance is sufficiently accurate. Based on experimental studies, we found that a good threshold for switching between local and global heuristics is 10 minimum turning radii.

As we mentioned, it is crucial for the heuristic to be as accurate as possible for high efficiency. However, there is no known closed-form solution for calculating the local heuristic, and obtaining an accurate estimate requires solving the original planning problem. Thus, we defined an off-line pre-processing step to create the heuristic look-up table (LUT). The table is simply a compilation of the costs to travel from the origin to all lattice nodes in its local neighborhood. These costs are determined by running the planner for each possible path endpoints using simply Euclidean distance as heuristic, which is guaranteed to be an under-estimate. LUT generation could be a lengthy process, but it is performed off-line, and the agenda for future work includes developing advanced function approximators to eliminate this pre-processing. The exact values for the path costs provided by the LUT result in the dramatic speed-up of the planner as described in the next section.

5.2 Path Planner Results

In order to quantify the performance of the present path planner, we undertook a simulation study that included performing a statistically significant number of planning experiments, where initial and final path configurations were chosen at random. It was confirmed that the planner built using the control set generated in Section 4.2 for the robots in [4] is very efficient: it performs as efficiently as basic grid search. In fact, for over 90% of path planning queries, our method performs even faster than grid search. The significant result here is that 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 reason for this significant speed-up is twofold. The primitive paths can span multiple grid cells, such that by choosing a primitive, the planner may "jump" ahead, while grid search still considers one cell after another. Besides, the accurate heuristic as provided by the LUT was shown to reduce the number of required search iterations considerably.

In Figure 5 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. The figure shows the results of over 1000 timing experiments for both nonholonomic

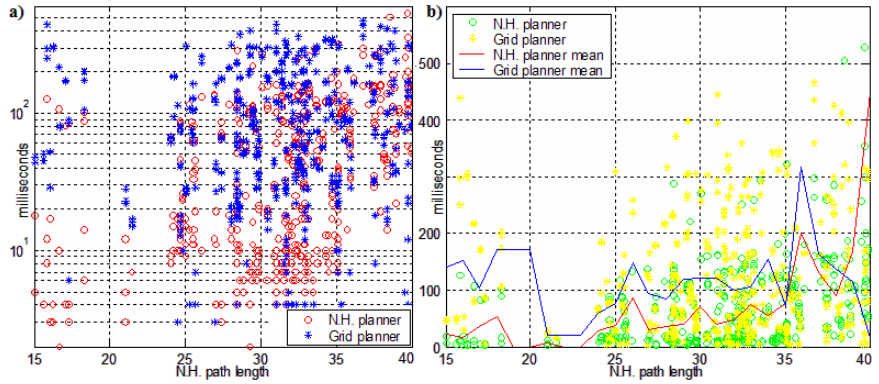

Fig. 5. Run-time results of our nonholonomic (N.H.) path planner (red datapoints) in comparison with basic grid search (blue datapoints). Vertical axis is the time of plan generation, and horizontal axis is the length of the nonholonomic paths. a) Runtimes for both nonholonomic and grid search on semi-log scale. b) Average runtimes superimposed on the same plot on linear scale.

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 increasing nonholonomic planning complexity. Figure 5a shows the runtime versus nonholonomic path length (i.e. "complexity") per planning query, plotted on the semi-log scale. Nonholonomic planner runtimes are denoted with circles, and the grid search runtimes – with stars. Even though the plot looks rather busy, the clustering of circles below the stars is clearly visible, indicating that on average nonholonomic planner ran faster in the same experiment (i.e. a choice of path endpoints). This trend is easier to see in Figure 5, where we superimposed the mean of runtime for both planners. The two solid lines (red for nonholonomic, and blue for grid search) clearly show that nonholonomic planner on average 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. Thus, our path planner is clearly very efficient and can compute most planning queries in less than 100ms, which deems it useful for real-time applications.

6 Applications

We discuss several important mobile robotics applications that could greatly benefit from the constrained motion planning approach presented herein. The

constrained motion planner presented here was successfully implemented on the DARPA PerceptOR program [4]. 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. Fig. 6b depicts an example motion plan that was generated by the vehicle on-line. The grayscale portion in the figure represents the cost-map, pink indicates obstacles, and orange is the area of unknown cost. Yellow line represents the generated plan. With this technology, the PerceptOR vehicles exhibited kilometers of autonomous traverse in very difficult natural terrain (Fig. 6a).

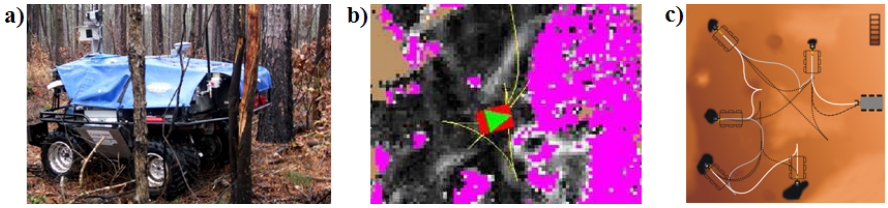

Fig. 6. Example applications: a) Robot navigation in natural cluttered environments (DARPA PerceptOR). b) A nonholonomic path computed in a natural cul-de-sac. c) Planetary rover instrument placement problem. The rover must approach five science objects at specified heading in cluttered environment on the slope of a crater.

Another important application for which the presented motion planner is suited especially 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 (see Figure 6c). 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 [10] 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 continuous curvature functions is the possibility to define velocity planning quite easily. By defining a maxi-

mum 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 scale the results to be applicable to kilometers of traverse.

References

1. Latombe J-C (1991) Robot motion planning. Kluwer, Boston
2. Branicky MS, LaValle S, Olson S, Yang L (2001) Quasi-randomized path planning. In: Proc. of the Int. Conf. on Robotics and Automation
3. Frazzoli E, Dahleh MA, and Feron E (2001) Real-time motion planning for agile autonomous vehicles. In: Proc. of the American Control Conference
4. Kelly A et al. (2004) Toward reliable off-road autonomous vehicle operating in challenging environments. In: Proc. of the Int. Symp. on Experimental Robotics
5. 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
6. Pancanti S et al. (2004) Motion planning through symbols and lattices. In: Proc. of the Int. Conf. on Robotics and Automation
7. 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
8. Wang D, Feng Q (2001) Trajectory planning for a four-wheel-steering vehicle. In: Proc. of the Int. Conf. on Robotics and Automation
9. 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
10. Kelly A and Nagy B (2003) Reactive nonholonomic trajectory generation via parametric optimal control. Int. J. of Robotics Research 22:583-601
11. 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