2013 IEEE Intelligent Vehicles Symposium (IV)
June 23-26, 2013, Gold Coast, Australia

Anytime Path Planning in Graduated State Space*

Haojie Zhang, Guangming Xiong, Bo Su, Jianwei Gong,Yan Jiang, Huiyan Chen and Wei Lan

***Abstract*—Complex robotic systems often have to operate in large environments. At the same time, their dynamic is complex enough that path planning algorithms need to reason about the kinodynamic constraints of these systems. On the other hand, such roboticsystems are typically expected to operate with speed that is commensurate with that of humans. This poses stringent limitation on available planning time. These will result in a contradiction between planning efficiency and the dimensions of the state space determined by the kinodynamic constraints. In this paper we present an anytime path planning algorithm to solving this problem. First, a graduated state space which consists of state lattices and grids is constructed for planning. Then, ARA* algorithm is utilized to search the graduated state space to find a path that satisfies the kinodynamic constraints and available runtime of planning.**

I. INTRODUCTION

As one of the most important technologies in the field of navigation for robots, path planning is the basis for the robots to achieve fully autonomous driving. Due to the differential kinodynamic constraints, it is necessary to consider these constraints in path planning, especially when the robot is operating in cluttered, complex environments. However, this will cause path planning procedure into searching a path with high-dimensional state space. As a result, the planning efficiency will yield a dramatic decrease. On the other hand, such robotic systems are typically expected to operate with speed that is commensurate with that of humans. Thus, the planning efficiency should be guaranteed to avoid collision or accident when the robot is operating in dynamic and time-varying environments.

Currently, there are two main approaches to address the contradiction between planning efficiency and the dimensions of the state space determined by the differential kinodynamic constraints. One method is to perform lowdimensional planning over the whole state space ignoring the robot's kinodynamic constraints, such as two dimensions(x,y) planning. Then, a separate local controller or local planner is designed to perform a higher dimensional plan on a small local region around the robot [1, 2, 3]. However, these approaches are either suboptimal [4, 5] or are limited in the size of the area they can handle [6]. The other one is to perform high dimensional planning over the whole state space. Theseapproaches aredemonstrated and validated in small size of environments. However, the planning efficiency of these approaches will yield a dramatic decrease as the size of the environments is increasing. This will make it hard to generate a plan during the available planning time.

Based on the approach we proposed in [7], this paper addresses the problem by searching in a graduated state space. The proposed approach relies on the notion that only the area closest to the robot needs to be searched carefully to guarantee an executable path, areas further from the robot can be searched more coarsely. This results in substantial speedups and lower memory requirements while introducing the robot's kinodynamic constraints into planning. We show that the algorithm is complete with respect to the state-space discretization and demonstrate the planning efficiency in simulation.

II. GRADUATEDSTATESPACE

In order to reduce the computational complexity of planning and guarantee the feasibility of the executable path, a graduated state space is constructed in this section. It consists of high dimensional subgraphG_H and low dimensional subgraphG_L. The kinodynamicconstraints of the robot is reflected in subgraphG_H. The arrangement of vertices and edges in G_H is assumed to be state lattices suggested by M. Pivtoraiko[8], while grids in G_L, as shown in Fig.1.

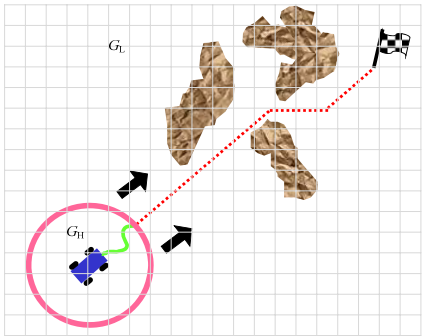

Figure 1. The diagram of graduated state space

*Resrach supported by the National Natural Science Foundation of China (Grant No. 91120010&91120015).

Haojie Zhang is with the Intelligent Vehicle Research Center, Beijing Institute of Technology and Key Laboratory of Biomimetic Robots and Systems, Ministry of Educationof China and China North Vehicle Research Institute, China North Industries Group Corporation.(86-10-68918652; fax: 86-10-68918290; e-mail: haojie.bit@gmail.com).

Guangming Xiong is the corresponding author. He is with the Intelligent Vehicle Research Center, Beijing Institute of Technology and Key Laboratory of Biomimetic Robots and Systems, Ministry of Educationof China (86-10-68918652; fax: 86-10-68918290; e-mail:xiongguangming@bit.edu.cn).

Bo Su is with the China North Vehicle Research Institute, China North Industries Group Corporation.(86-10-83808104; fax: 86-10-83808104; e-mail: bosu@noveri.com.cn).

Jianwei Gong is with the Intelligent Vehicle Research Center, Beijing Institute of Technology and Key Laboratory of Biomimetic Robots and Systems, Ministry of Educationof China (e-mail:gongjianwei@bit.edu.cn).

Yan Jiang is with the Intelligent Vehicle Research Center, Beijing Institute of Technology and Key Laboratory of Biomimetic Robots and Systems, Ministry of Educationof China (e-mail: piuzuzu@gmail.com).

Huiyan Chen is with the Intelligent Vehicle Research Center, Beijing Institute of Technologyand Key Laboratory of Biomimetic Robots and Systems, Ministry of Educationof China. (e-mail: chen_h_y@263.net).

Wei Lan is with the China North Vehicle Research Institute, China North Industries Group Corporation.(86-10-83809650; fax: 86-10-83809650; e-mail: wl1976@163.com).

The subgraphG_H is moving following the robot, denoted as G_{H1} before moving and G_{H2} after moving, as shown in Fig.2. After G_H is moving, some of the vertices near its boundary are set to belong to a different subgraph. In other words, the vertices near the boundary will change ownership from one subgraph to another, but they do not move. For example, as a subgraph changes position, it "gains" the vertices on its leading edge and "loses" vertices on its trailing edge. Thus, the cost between these vertices will be changed due to new connectivity under a different subgraph. In order to search in the graduated state space, it is necessary to make the replanning algorithm aware of the vertices that have new subgraph ownership. This is performed by the function *convert_vertex*, presented in Algorithm 1.The function *convert_vertex* is executed on each vertex s that changes subgraphownership. The updateSetMembership function in Algorithm 1 is set to update the list of states for expansion.

Algorithm 1 convert_vertex

1: for $\forall s \in G_{H1} \vee G_{H2}$
2:$g(s)=\infty$, $v(s)=\infty$
3:updateSetMembership(s)
4: for $\forall s \in G_F (s \neq s_{\text{start}})$
5:$g(s)=\infty$
6:$bs(s)=\arg\min_{s' \in \text{succs}(s)}(v(s')+c(s', s))$
7:$g(s)=v(bs(s))+c(bs(s), s)$
8:updateSetMembership(s)
9: for $s \in G_F$ and $v(s) \neq \infty$
10:for each $s' \in \text{preds}(s)$
11:**if**$s' \in G_{H1} \vee G_{H2}$**then**
12:$g(s')=\infty$
13:$bs(s')=\text{argmin}_{s'' \in \text{succs}(s')}(v(s'')+c(s'', s'))$
14:$g(s')=v(bs(s'))+c(bs(s'), s')$
15:updateSetMembership(s')
16: **end if**

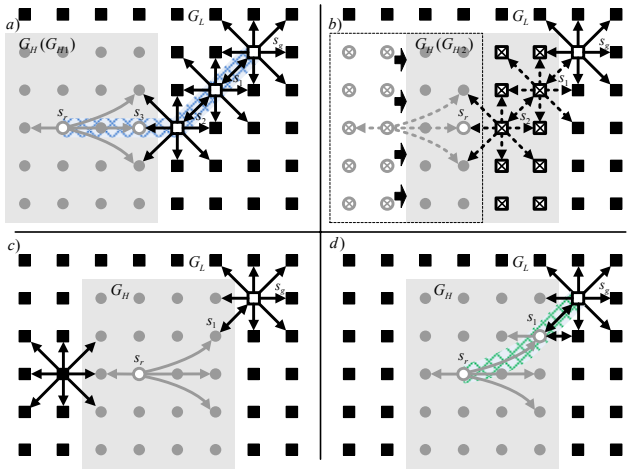

Figure 2. Rebuild the connectivity between two subgraphs of different dimensions. G_L is thelow dimensional subgraphand G_H is thehighdimensional subgraph.

This procedure of rebuilding the connectivity after the subgraph is moving is illustrated in Fig.2, using a simplified search space for ease of visualization. In this example, the graduated state space consists of two subgraphs, G_H(gray circle vertices) and G_L(black square vertices). Arrows of similar colors are the edges. G_H is a small subgraph, consisting of twenty vertices, highlighted with a gray bounding box. In this example, the rest of the graduated state space belongs to G_L.The eight neighbor grids connection is chosen in G_L, such as for $\forall s_i \in G_L$, $preds(s_i)$denote the 8 nearest neighbors of s_i. While state lattice is chosen in G_H, for $\forall s_j \in G_H$, $preds(s_j)$ are state lattices. The effect of the initial plan is shown in Fig.2a from start state s_r to goal state s_g. The vertices with white centers were expanded during search, and solid vertices that were pointed to by the arrows, remain in the priority queue. The resulting path is highlighted with a thick blue pattern line. Next, assume we move G_H to the right, as shown in Fig.2b. There will be twenty crossed-out vertices changing subgraph ownership due to this moving. For these vertices, we execute *convert_vertex* on each of them. The previous expansion of these vertices is undone and their edges(dotted arrows) built are removed (line 2-3, Algorithm 1). At the same time, the affected vertices are inserted into the priority queue. They will be expanded in the changed subgraph again when a new replanning procedure begins, as shown in Fig.2c. After replanning, Fig.2d completes our example and a new path (thick green pattern line) is generated due to moving G_H to the right.

Additional connecting edges are used on the boundary of different subgraphs. In this way, a vertex that lies on the boundary of a subgraph, can be expanded to connect with other vertices in a different subgraph. A simple example of connecting a 2D (x,y) subgraph G_L to a 3D (x,y,θ) subgraph $G_H$is shown in Fig.3. The gray arrows in Fig.3 are used to generate extra connectivity from the vertex s_0 in subgraph G_L to subgraph G_H. The predecessorsof s_0 are 8 nearest neighbors in subgraph G_L. It has three predecessors in subgraph G_H. In this case, we have connected this vertex with several other vertices(stacking cubes) in G_H representing all possible edges of the third dimension, such as different heading θin this example. This addition certainly increases the branching factor of boundary vertices and builds the connectivity between vertices on the boundary of different subgraphs.

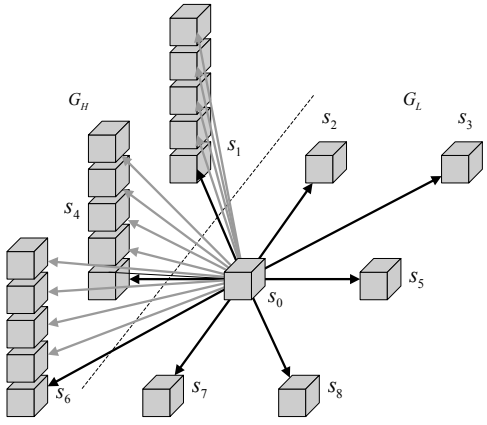

Figure 3.The 2D(x,y) subgraph G_L(8-connected grid) is connected to another 3D(x,y,θ) subgraph G_H. Black arrows are regular edges in G_L and gray arrows are additional edges that connect the two subgraphs.

III. ALGORITHM

In this section we will give a detailed description about the proposed approach. Anytime Repairing A*(ARA*) was the search algorithm used in the description implementation.

A. ARA* Algorithm

ARA* algorithm is an A*-based anytime search algorithm that produces significantly better solutions than current approaches, while also providing suboptimal bounds on the quality of the solution at any point in time. Unlike A* search algorithm, ARA* uses an updated cost function (usually denoted $f(s)$), which is expressed as

$$f(s)=g(s)+\varepsilon h(s) \quad (1)$$

Where $g(s)$ is always the cost of the best path found so far from s_{start} to s, $h(s)$ is the estimate cost from s to the goal and $\varepsilon$is the inflation factor.

ARA* algorithm can drastically reduce the number of states A* has to examine before it can produce a solution through inflating the heuristic by some constant $\varepsilon>1$. An additional nice property of inflating heuristic is that the cost of the solution found for an inflation factor $\varepsilon$is no larger than εtimes the cost of an optimal solution.

ARA* maintains three lists (*OPEN*, *CLOSED* and *INCONS*) during the search. *OPEN*list contains only the consistent states that have not yet been expanded. However, we need to track of all inconsistent states since they will be used to initialize *OPEN* in future searches. ARA* does this by maintaining a set *INCONS* of all the inconsistent states that are not in *OPEN*. Thus, the union of *INCONS* and *OPEN* is exactly the set of all inconsistent states, and can be used as a starting point for the inconsistency propagation before each new search. In successive executions of this algorithm, the search only expands the states that are inconsistent and tries to make them consistent. Thus, ARA* algorithm is guaranteed to be anytime search algorithm by increasing or decreasing the inflation factorεin one single search. More details about ARA* algorithm can be found in [9].

B. Description

The graduated searchalgorithm is searching with the graduated state space constructed in Section II. It starts out by planning in highdimensional state space around the robot and in 2dimensional state space everywhere else. Then, as the robot starts executing the path, the planner will leave a permanent highdimensional region at the current location whenever the new path is different from the previous path or it recognizes the potential for oscillations to occur. In our case, the 2-dimension of our algorithm is used to represent the planar position (x,y). The highdimensional state space is typically position and orientation (x,y,θ).

C. Notations and Assumptions

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

Definition 1($\lambda(\cdot)$): Function returning the projection of a state into a highdimensional state. There also exists the inverse λ^{-1} which maps from a highdimensional state to a lowdimensional state.

$$\lambda(s^l)=s^h \quad (2)$$

$$\lambda(s^h)=s^l \quad (3)$$

Definition 2(Q): Q is the queue that stores the position of all high dimensional regions that have been introduced by the planner.

Definition 3(LQ, HQ): LQ is the queue that stores the position where the infeasible path is generated. HQ is the queue that stores the lowdimensional state s^l and its lower bound if $\lambda(s^l)$ belongs to highdimensional state space.

Definition 4(π^i): The i^{th} plan returned from Computepath function. It consists of the ordered list of states $\{\pi^i(0), \pi^i(1), \cdots, \pi^i(k)\}$ where $\pi^i(0)=s_{curr}$, $\pi^i(k)=s_{goal}$.

D. Graph Search

The graduated search algorithm is based on ARA* algorithm and detailed in Algorithm 2. The variables specific to this algorithm are initialized on line 1 prior to the first search. Computepath function on line 5 takes the current position and goal and run backward ARA* algorithm with the graduated state space (G_H,G_L). It returns the path and the state's g value both in highdimensional state space and lowdimensional state space. Also, the C1 is a returned variable to denote whether the path generated meets the "Condition 1". In our case, the condition 1 we use is a change in homotopy class[10]. If the "Condition 1" is satisfied (as detected by line 23 in Algorithm 2), then the current location is added to Q which is a queue of high dimensional regions. This is done because "Condition 1" should signal when new information, such as the inclusion of the higher dimensions in the search, has shown the current trajectory to not be optimal. By placing the current position into list of highdimensional regions, this region will be remembered as highdimensional state space during the planning cycles.

For these states on the current path, if there exists both a state s'in highdimensional state space and its best successor $bs(s')$ in lowdimensional state space and a state s'' in lowdimensional state space and its best successor $bs(s'')$ in highdimensional state space, then the state $bs(s')$ will be added to the list of LQ which indicates the potential oscillation(line 26-28, Algorithm 2). States in the low dimensional state space are on the current path to the goal and now have a lower g-value than the g_{low} stored for that state are also inserted into the list of high dimensional queue Q (lines 29-31, Algorithm 2).This is done in order to prevent oscillations as a drop in a g-value indicates a fundamental change in the path taken from that location to the goal, even if "Condition 1" was not true. As an example, consider a long, narrow hallway with a robot incapable of turning in place, a high penalty for traveling backwards and a goal state behind the robot, as shown in Fig.4. In Fig.4, the circles represent the high dimensional state spaces and rectangles denote the footprint of the robot. If the robot cannot sense the end of the hallway, it may choose to travel forward to the unseen area and turn around there (following the deep red dotted line, Fig.4). However, once reading the end of the hallway, the robot will determine that it is unable to turn-around, and will travel in reverse back to the goal state (following the green dotted line, Fig.4). As it does

this, it is undesirable for the robot to return to the end of the hallway once it leaves the sensor range. However, we can detect this phenomenon by the lowering of the *g*-values of the states in the lowdimensional state space and place highdimensional regions along the hallway. In this way, the robot is guaranteed to not oscillate.

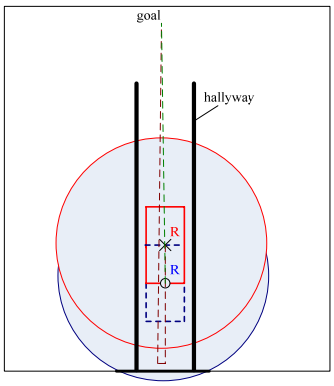

Figure 4. A situation of robot traveling in a hallway. Circle regions are high-dimensional subgraph G_H.

Algorithm 2SEARCH(G_H, G_L, s_{start}, s_{goal})

1: for $\forall s$, $g_{low}(s)= \infty$, Q=Ø, LQ= Ø, HQ=Ø
2:**while**$s_{curr} \neq s_{goal}$**do**
3:update map with sensor data
4: $i=i+1$
5: $[\pi^i\ g_H^i\ g_L^i$ C1]=Computepath(s_{curr},s_{goal},G_H,G_L)
6:**if**$LQ \neq$Ø **then**
7:for$\forall s \in LQ$
8:**if**$\exists \lambda(s) \in G_H$ **then**
9:S=$\lambda(s)$
10:**if**$\exists\ \hat{s} \in$S and $\hat{s}$ is expanded**then**
11:for $\forall\ \hat{s} \in$S
12:$g_{low}(\lambda^{-1}(s))=\min(g_{low}(\lambda^{-1}(s)), g_H^i(\hat{s}))$
13: **end if**
14: otherwise
15: for $\forall\ \hat{s} \in$S
16:$g_{low}(\lambda^{-1}(s))=\min(g_{low}(\lambda^{-1}(s)), g_H^i(s_{curr}) - h(\hat{s}))$
17: **end if**
18:$g_{low}(\lambda^{-1}(s))=\max(g_{low}(\lambda^{-1}(s)), g_L^{i-1}(\lambda^{-1}(s)))$
19: insert/update s and $g_{low}(\lambda^{-1}(s))$ into HQ
20: **end if**
21: LQ=Ø
22: $cp(s_{curr})$=argmin $_{\pi^{i-1}(m) \in \pi^{i-1}}$euclidean ($\pi^{i-1}$(m),$s_{curr}$)
23: **if** C1 is satisfied **then**
24: insert s_{curr} in Q
25: **end if**
26: **if**$\exists s', s'' \in \pi^i$:$(s' \in G_H) \wedge (bs(s') \in G_L) \wedge (s'' \in G_L) \wedge (bs(s'') \in G_H)$ **then**
27: insert $bs(s')$ into LQ
28: **end if**
29: **if** $(bs(s') \in HQ) \wedge (g_L^{i-1}(bs(s')) \leq g_{low}(bs(s')))$**then**
30: insert/update $bs(s')$ into Q
31: **end if**
32: $s_{curr}=\pi^i(1)$
33: convert_vertex
34: **end while**

E. *Theoretical Analysis*

The algorithm is guaranteed complete due to the construction of the graduated state space. An intuition understanding of the proof of the theoretical properties is presented here.

Assuming a solution exists on the full high dimensional graph G_H, then given sufficient time a solution will be found in the graduated state space. To see this, recall that the robot will continue along the same homotopic path unless the high dimensional region encounters an obstacle that is not navigable. At that point, a high dimensional region is added to the graduated state space for all future plans. Since there are a finite number of states in the graduated state space and a finite number of times we can discover that a previous low dimensional plan is infeasible, and thus the algorithm will eventually find a feasible solution.

IV. EXPERIMENTRESULTS

In order to test the efficiency of the proposed anytime path planning algorithm, we randomly generated 40 outdoor maps of four different sizes, such as 500×500, 1000×1000, 1500×1500 and 2000×2000 cells with a resolution of 0.02m. The outdoor environments are very open, with randomly placed circle obstacles (representing trees, rocks etc) that occupy roughly 30% of the map, as shown in Fig.5. The robot's footprint occupied 4×10 grids. For these maps, the start and goal states were placed in diagonally opposite corners with a heading of 0°.

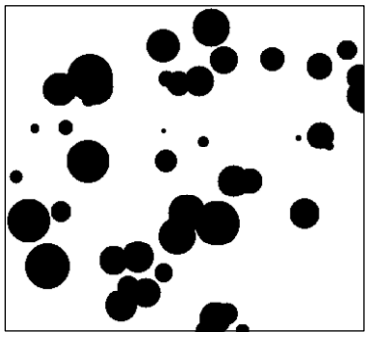

Figure 5. An example outdoor map used for our experiments

In the simulation experiments, the environments that we generated above are totally known for planning. We set a constant inflated factor ε(ε=3.0,2.0,1.0) separately for ARA* algorithm to search with highdimensional state space 3D(x,y,θ) and proposed anytime planning algorithm to search with graduated state space (3D(x,y,θ)\2D(x,y)). The experimental results are shown in Table 1. All the simulation experiments were run on an Intel (R) Core (TM) i7 CPU running at 2.93 GHz, under Ubuntu operating system.

As can be seen from table 1, our algorithm had comparable path costs to the optimal search for the same constant inflated factor ε, just a little more than the costs of optimal path. However, the planning times are significantly decreased on average for each map size. The speed of planning is improved nearly1-95times on average for each map size.

V. CONCLUSION

In this paper, we have described an approach to improve the efficiency of planning with kinodynamic constraints. The proposed graduated state space consists of high dimensional state space and low dimensional state space. The high dimensional state space was designed to satisfy the kinodynamic constraints of the robot. An anytime planning algorithm based on ARA* was presented to search with the graduated state space. This new planning algorithm was successfully demonstrated in simulation. Future work includes implementing it on real robots and revising the graduated state space to further improve planning efficiency.

TABLE 1.SIMULATIONRESULTS

			ε=3.0		ε=2.0		ε=1.0	
	Map size	# of Maps	Ave Path Cost	Ave Plan Time(s)	Ave Path Cost	Ave Plan Time(s)	Ave Path Cost	Ave Plan Time(s)
New Anytime algorithm	500×500	10	57750	0.0037	45768	0.0065	43591	0.0652
	1000×1000	10	107442	0.0115	77049	0.0322	71130	0.1462
	1500×1500	10	124610	0.0059	114430	0.0144	102699	0.1313
	2000×2000	10	165600	0.0085	149060	0.0183	133640	0.1485
ARA* algorithm	500×500	10	53760	0.0353	45390	0.0535	37876	0.0793
	1000×1000	10	101940	0.1904	76991	0.1707	63375	0.2472
	1500×1500	10	122690	0.3920	103890	0.4095	84594	0.4938
	2000×2000	10	161870	0.6964	147784	0.6966	122040	0.8869

REFERENCES

[1] O. Brock and O. Khatib, "High-speed navigation using the global dynamic window approach" in *Proc. 1999 IEEE Int. Conf. on Robotics and Automation*. vol. 1, 1999, pp. 341–346.

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

[3] R. Philippsen and R. Siegwart, "Smooth and efficient obstacle avoidance for a tour guide robot" in *Proc. 2003 IEEE Int. Conf. on Robotics and Automation*. Vol. 1, 2003, pp. 446-451.

[4] K. Bekris and L. Kavraki, "Greedy but safe replanning under kinodynamic constraints," in *Proc. 2007 IEEE Int. Conf. on Robotics and Automation*. vol. 1, 2007, pp. 704-710.

[5] S.Petti and T. Fraichard, "Safe motion planning in dynamic environments," in Proc. 2005 *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*.August, 2005, pp. 2210-2215.

[6] M. Likhachev and D. Ferguson, "Planning long dynamically feasible maneuvers for autonomous vehicles," *Int. Journal of Robotics Research*, vol. 28, pp. 933-945.

[7] H. Zhang, J. Butzke and M. Likhachev, "Combining global and local planning with guarantees on completeness", in *Proc. 2012 IEEE Int. Conf. on Robotics and Automation*. vol.1, 2012, pp. 4500-4506.

[8] M. Pivtoraiko, R. A. Knepper and A. Kelly, "Differentially constrained mobile robot motion planning in state lattices,"*Int. Journal of Field Robotics*, 26(3): 308-333, 2009.

[9] M. Likhachev, D. Ferguson, G. Gordon et.al. "Anytime search in dynamic graphs,"*Journal of Artificial Intelligence*, 172(14): 1613-1643, 2008.

[10] S. Bhattacharya, V. Kumar and M. Likhachev, "Search-based path planning with homotopy class constraints", in *Proc. 2010 National Conf. on Artificial Intelligence*.