Proceedings of the 16th International IEEE Annual Conference on Intelligent Transportation Systems (ITSC 2013), The Hague, The Netherlands, October 6-9, 2013

Path Planning on Grid Maps with Unknown Goal Poses

Georg Tanzmeister, Martin Friedl, Dirk Wollherr and Martin Buss

Abstract— **Path planning for robots typically consists of finding a path from a given start state to one or multiple given goal states. However, there are situations in which the pose of the goal state is not explicitly known, e.g. in sensor-based autonomous driving in unknown environments. This paper presents a path planner that is capable of planning feasible paths in the absence of goal poses. The approach combines the advantages of both the focused search of A* and the uniformly-exploring search of Rapidly Exploring Random Trees. With this approach, it is possible to quickly find potential goal states and their corresponding paths and to continue the exploration as processing time allows. Furthermore, it is shown how to cluster paths to extract the main possible directions. Results on simulation and real data are given to demonstrate the utility and efficiency of the proposed approach.**

I. Introduction

Path planning is a well established area of research in robotics. The problem of path planning can be formulated as follows: Given a start state, a goal state, a representation of the robot and a representation of the world, find a collision-free path that connects the start with the goal satisfying the system constraints [1]. Although some problems provide multiple start states and others multiple goal states, these together with the geometric representation are the typical input of a path planner. In some situations, however, the pose of the goal state, i.e. the location and the orientation, might not be known.

Consider for example an autonomous vehicle driving on public streets using a global and a local planner. The global planner uses a given map containing information about lanes for global coarse navigation and the local planner calculates the actually driven trajectory using sensor information to detect static and dynamic objects in the vicinity of the robot. Such a configuration was used by many autonomous vehicles in the DARPA Urban Challenge, e.g. [2], [3]. In road construction sites, however, map data becomes out-of-date and the global path is not valid anymore.

When the road structure changes temporarily, the new road course is usually highly structured and will eventually lead back to regular streets. Additionally, the new road course is commonly shaped to be drivable with a certain speed, which poses additional constraints on the search space, i.e. it is very unlikely that one ought to stop and to steer the maximum possible angle to catch a turn. Similarly to road changes, localization difficulties are also problematic, since then, the orientation and the position in the map are not accurate and even with slight inaccuracies in the orientation, keeping the lane becomes difficult. These observations are used to motivate a planner that provides a path to the local planner in cases where the global map-based planner cannot, enabling the robot to continue, as shown in Fig. 1. Since the new road course is not known a-priori, there is no goal pose to plan towards, which makes traditional path planning algorithms not applicable. The contribution of this paper is best described as a planner that

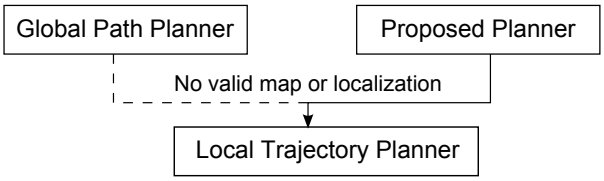


Fig. 1. The planner in the context of autonomous driving.

- is capable of planning feasible paths without knowing the desired goal pose,
- combines the main elements of two well-known classes of path planning algorithms, i.e. A* and Rapidly Exploring Random Trees (RRTs),
- is able to quickly find a path candidate and to continue exploration as time allows,
- can cluster the found path candidates to capture the main outcome and to reduce the complexity.

The paper is structured as follows. Section II gives a review of related planning algorithms. Section III formally presents the problem of planning with unknown goal poses and gives a detailed description of the proposed planner. Based on the output, Section IV then demonstrates how to cluster paths to reduce complexity and finally, Section V shows and discusses simulation results.

II. Related Work

Grid maps are a common way to represent the environment [4]. Graph search algorithms, like A*, can be applied on a grid by connecting grid cells to build a graph structure. Much work has been done to improve the original A*. Incremental search algorithms such as D* [5] are able to quickly re-plan in dynamic environments and any-time planners like Anytime Repairing A* [6] can rapidly find a potentially sub-optimal path and approach the optimal one with more time.

Apart from combinations of both, e.g. Anytime Dynamic A* [7], there are also modifications that tackle the problem of unnatural and suboptimal paths produced by A* and many of its variants. Theta* [8] modifies how the search graph is

G. Tanzmeister and M. Friedl are with BMW Group Research and Technology, D-80992 Munich, Germany (e-mail: georg.tanzmeister@bmw.de; martin.mf.friedl@bmw.de).

D. Wollherr and M. Buss are with the Institute of Automatic Control Engineering of the Technische Universität München, D-80290 Munich, Germany (e-mail: dw@tum.de; mb@tum.de).

978-1-4799-2914-613/$31.00 ©2013 IEEE

generated. Instead of restricting the neighbors of each node in the graph to the neighbors of its grid cell, an edge between two nodes can traverse multiple cells and hence the move direction from one node to another is not limited to the either 4 or 8 neighbors anymore. Field D* [9] also provides a way to represent an any-angle move between two cells but cell costs are associated to the cell corners rather than the cell centers and an any-angle transition is calculated by linearly interpolating the cost of the corresponding cell corners.

Planning in an any-angle fashion is often still not enough to satisfy non-holonomic constraints for e.g. a vehicular robot. Dolgov et al. [10] presented Hybrid-state A*, a variant similar to Field D* but without the limitation of piecewise linear paths. In Hybrid-state A*, every cell of the discretized state space can be associated with an arbitrary continuous state in its discrete range and transitions are done by applying a kinematic model and by using the continuous state. If a transition ends up in a cell already visited and which is on the list of nodes to be processed, the cost is compared and the cheaper one is used. If the state in the cell was already used to plan further, i.e. it is marked as $closed$, the edge is pruned. Hybrid-state A* is not guaranteed to find the minimal-cost path due to the pruning, the discretization of controls and time, and a violation of the Markov property, although the output typically lies in the neighborhood of the global optimum, as the authors point out.

Apart from A*, Rapidly Exploring Random Trees [11] are another popular tool for planning. In the RRT algorithm, a search tree is built that grows towards samples from the search space. A path to the goal is found by sampling the goal state with some, usually low, probability and thus by trying to connect the search tree with the goal. Multiple Extensions have been developed e.g. to incorporate cost information in the node expansion [12] or to plan in an any-time way [13]. Combinations of A* and RRTs typically occur in roadmap based methods, where RRTs are used to construct the roadmap and A* to compute shortest paths [14]. Other techniques apply workspace decomposition and use heuristics similar to A* for the roadmap construction [15], or use the decomposition to run a discrete graph-search algorithm like A*, to obtain leads for a continuous search [16].

All of the above techniques require query pairs of a start and a goal states to calculate a paths. This paper presents a planner capable of planning paths in the absence of goal poses. The approach proposed combines A* and RRTs and employs them both in the exploration phase. A* is used to quickly find possible paths and RRTs build upon the same, partially-explored search tree to uniformly explore it further. In the A* part of the technique, many of its variants can be applied. However, in the investigated scenarios, most utility can be extracted when dealing with non-holonomic constraints and we build upon the work on Hybrid-state A*. In contrast to Hybrid-state A* though, the search tree connections are not restricted to direct grid cell neighbors but allowed to traverse multiple cells, similar to Theta*. Any-time characteristics, i.e improving the solution with increasing time, are integrated in that the more time that is available, the more of the search space is explored and the more diversity in the clustered path candidates exist.

Planning paths in the absence of goal poses is partially related to reachability analysis, which aims at extracting all reachable states from a certain start state within a fixed time frame [17]. Here, however, the main motivation is to find a usually small subset of all reachable states that satisfy certain goal constraints, e.g. whose paths equal a certain length. Any guarantees of finding all reachable states are insignificant. Furthermore, path cost is important and planning characteristics, such as optimality and completeness, are examined. The planning problem and the proposed planner are described in the following sections.

III. Planning with Unknown Goal Poses

This section formally presents the planning problem. It is shown, how a goal-oriented heuristic can be used even in the absence of goal poses, the benefits from combining A* and RRTs are discussed and finally, the implementation of the proposed algorithm together with a detailed description in pseudo-code is presented.

A. Problem Formulation

The problem of path planning with unknown goal poses can be defined as follows. Given an initial state of the robot, q_S, a desired goal path length, l_G, a configuration space, C, and a feasible, collision-free path between two states x and y, $\boldsymbol{p}(x, y)$, find a set of goal states

$$G = \{x \mid x \in C_{free} \wedge \exists \boldsymbol{p}(q_S, x) \wedge ||\boldsymbol{p}(q_S, x)|| = l_G\} \quad (1)$$

There are two desired properties about G. First, G should contain the goal state reached by the minimum cost, collision-free path from q_S, since without application specific knowledge this can be assumed to be the best one. Second, G should exhibit diversity. Diversity in goal states and their corresponding paths is desired, so that the information gain between different elements in G is high, since further processing, e.g. when applying domain knowledge for final path selection, can then choose from a greater variety.

Note that the problem could have been formulated more abstract by using any predicate instead of l_G that must hold and that allows a fair comparison between elements in G. Since this work focuses on online, sensor-based planning in unknown environments, l_G fits perfectly the requirements and is thus integrated directly in the problem.

B. Heuristic-based Search with Unknown Goal Poses

In A*, the search is based on a heuristic that represents the estimated cost from a node to the goal. For the heuristic to be admissible, it must not overestimate the true cost to the goal. Often the path length is directly used as the path cost and the straight-line distance to the goal is used as heuristic.

Even without goal poses, the path length can be used as a heuristic by deriving it with the desired goal path length l_G. In order to represent arbitrary costs though, it needs to be integrated into the path cost in a way that the resulting heuristic is still admissible. Let c_i be the edge cost between

978-1-4799-2914-613/$31.00 ©2013 IEEE

two adjacent nodes, l_i the corresponding path length and n a node in the search tree. Then, by restricting the minimum edge cost between two adjacent nodes to the segment path length, i.e. $c_i \geq l_i$, an admissible heuristic can be defined as

$$h(n) = \max(l_G - l_n, 0), \tag{2}$$

where l_n represents the path length from q_S to n.

As the length calculations might be costly, consider a different view. By setting the length of each path segment to a fixed size, the length of a path can be seen as the depth of the last node in the search tree. The edge cost constraint then simplifies to $c_i \geq 1$ and the heuristic to

$$h(n) = \text{depth}(l_G) - \text{depth}(n) \tag{3}$$

Since the depth of the search tree can be limited, there is no need for the maximum operator anymore to assure that $h(n) \geq 0$. Note, that setting the step size of the node expansion to a fixed value—and not necessarily to 1—is not obligatory for an admissible heuristic and is merely motivated to enable a fair comparison between goal path candidates, as will be shown in the next section.

C. Combining A* with RRTs

Due to its heuristic, A* is strongly focused towards the goal and is thus very efficient. A* is complete, i.e. if a solution exists it finds it, otherwise it reports back that no solution exists. A* is also optimal, i.e. it finds the minimum cost solution. Uniform-sampling RRTs on the other hand are in a way complementary to the focused, goal-oriented search of A*. One of the properties of standard RRTs is that nodes with large Voronoi regions are more likely to be expanded than nodes with small Voronoi regions and hence the tree favors to grow into unexplored areas [11].

Hence, for planning with unknown goal poses this paper proposes to first apply A* to find the minimum-cost path and to then continue the search with uniform-sampling RRTs. A* and RRT are combined such that the RRT builds upon the search results of A*, i.e. it directly uses its search tree for further expansion and thus every state is visited only once. Due to the combination, a planner is achieved that is both complete and uniformly exploring. Note, that in the implementation, Hybrid-state A* is used and thus optimality is given up, although the A* result lies near the optimum.

D. Finding Useful Paths

Without known goal poses and by using the length as goal criteria, it lies in the application-specific design of the planner of whether or not the produced paths are useful, e.g. a path that rotates the robot around its axis might be not. For autonomous vehicles the approach, proofed to be particularly useful. First, non-holonomic constraints greatly reduce the possible state space and thus the possible paths. Second, if a specific velocity is assumed that ought to be drivable, the state space is further dramatically reduced. Third, in structured environments, like roads, incorporating distance to obstacles into the costs greatly increases the path quality. And fourth, it is inherent in the RRT to favor exploration.

E. Implementation

The pseudo-code of the path planner is given in Algorithm 1. The algorithm starts with an A* search and uses the typical $open$ and $closed$ data structures for representing the search tree. There is an additional data structure $candidates$ where goal candidates are stored. Goal candidates are those nodes, whose $depth$ satisfy the desired length. Once a goal candidate is found, it switches to an RRT-based search using the same data structures. The only thing that actually changes is the method with which a node is extracted from $open$:

popA*(): As in conventional A*, the node with the lowest f-value is chosen for expansion.

popRRT(): Randomly pick a sample in the state space and find the closest node in $open$ to expand the tree. If the search tree has grown big enough, it is possible to switch strategy to randomly sample $open$, which is more efficient in terms of running time, when $open$ is large. It should be noted however that randomly picking a node in $open$ can lead to a bias towards areas that contain many $open$-nodes and thus may hinder a uniform exploration.

During node expansion several *move* commands are applied, which implicitly define the neighbors. As described above, the implementation builds upon Hybrid-state A* without the restriction that moves must land in neighboring cells. That way, the resolution of the grid and the resolution of the search tree can be independent, facilitating the trade-off between collision-checking accuracy and search time due to fewer push/pop operations. However, since a move can cross multiple cells, all of these need to be considered during collision-checking and cost calculation.

Anytime-characteristics can be integrated by returning the first path as soon as it has been found and by returning $candidates$ whenever it has grown significantly. For simplicity it was left out in the pseudo-code. In the following section it is shown how the resulting paths can be clustered.

IV. Clustering

Once collision-free paths through the grid have been found, they are clustered to reduce complexity and further processing. This step is motivated by the observation that many paths are similar and can be combined to one class. In further processing, e.g. when applying domain-specific knowledge for final path selection, this can provide a substantial speed-up since only one representative of each class needs to be considered. A natural selection of the representative is to take the one which exhibits the lowest overall cost. In the following the similarity calculation between paths is presented, which is required for any clustering algorithm.

A. Similarity between Paths

Path clustering requires distance calculations between paths. Let $\boldsymbol{p}^i$ and $\boldsymbol{q}^i$ represent the state of the node at the i-th movement step of path $\boldsymbol{p}$ and $\boldsymbol{q}$ respectively, then the distance between $\boldsymbol{p}$ and $\boldsymbol{q}$ can be calculated as

$$d(\boldsymbol{p}, \boldsymbol{q}) = \sum_i \rho(\boldsymbol{p}^i, \boldsymbol{q}^i) \tag{4}$$

978-1-4799-2914-613/$31.00 ©2013 IEEE

Algorithm 1 Planning with Unknown Goal Poses

$open$, $closed$, $candidates \leftarrow \emptyset$
$start.state \leftarrow q_S$
$start.[f, g, depth, parent]$, $i \leftarrow 0$
$open$.push($start$)
while $open \neq \emptyset$ **and** $i < maxIterations$ **do**
  if $candidates = \emptyset$ **then**
    $curr \leftarrow open$.popA*()
  else
    $curr \leftarrow open$.popRRT()
  end if
  $closed$.push($curr$)
  for each $a \in actions$ **do**
    $next.state \leftarrow$ move($curr.state$, a)
    if noCollision($curr.state$, $next.state$) $\wedge$ $next.cell \notin closed$ **then**
      $next.g \leftarrow$ calculateTotalCost($curr$, $next$)
      if $candidates = \emptyset$ **then**
        $h \leftarrow$ calculateHeuristic($next.state$)
        $next.f \leftarrow next.g + h$
      end if
      $next.depth \leftarrow curr.depth + 1$
      $next.parent \leftarrow curr$
      if $next.cell \in (candidates \cup open)$ **then**
        $ref \leftarrow$ getNodeInCell($next.cell$)
        if $next.g < ref.g$ **then**
          exchangeNodes($next$, ref)
        end if
      else if $next.depth \geq goalLength$ **then**
        $candidates$.push($next$)
      else
        $open$.push($next$)
      end if
    end if
  end for
  $i \leftarrow i + 1$
end while

The function ρ defines the distance between two points in some metric space, see [1] for more details. Since different units cannot be compared, e.g. angular quantities and distances in the plane, the robot displacement metric can be used. In the simulation results, the displacement metric is simplified to the Euclidean distance between the reference points in the plane:

$$\rho(\boldsymbol{p}^i, \boldsymbol{q}^i) = ||a(\boldsymbol{p}^i) - a(\boldsymbol{q}^i)||, \tag{5}$$

where $a(x)$ represents the position at state x. Fig. 2 illustrates the distance calculation. Using this simple distance measure between two paths, clusters can be built.

B. Hierarchical Clustering

Path candidates are clustered using agglomerative hierarchical single-linkage clustering as described in [18] for various reasons. First of all, is does not require to specify the number of clusters as opposed to e.g. K-Means. Additionally, it is simple, the number of clusters directly relate to the distance threshold and it does not represent noise, like DBSCAN [19]. Interestingly, especially the last point, which is usually pointed out as one of the advantages of DBSCAN compared to simpler clustering algorithms, turns out to be a disadvantage for our needs. The proposed path planner does not produce any noise, since every path is a valid, feasible path. Hence, if a single path is considered to be an outlier, it is just a different possible path and should thus be treated as separate cluster. Although one could eliminate the noise in DBSCAN by setting the minimum number of points that form a cluster to 1, which would greatly reduce its advantages in general, in hierarchical clustering, it is possible to modify the distance threshold t, used to link up different clusters, during execution. As further processing, like final path selection, often directly scales with the number of elements to process, in Algorithm 2 a strategy is proposed to automatically adapt t with a parameter c to limit the maximum number of clusters. It differs from a naive strategy to stop as soon as the cluster and the distance thresholds are reached and lead to better results in the experiments, which are given in the next section.

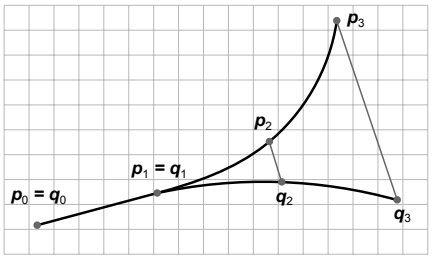


Fig. 2. Distance between two paths, $\boldsymbol{p}$ and $\boldsymbol{q}$. The euclidean metric is used on the equidistant reference points.

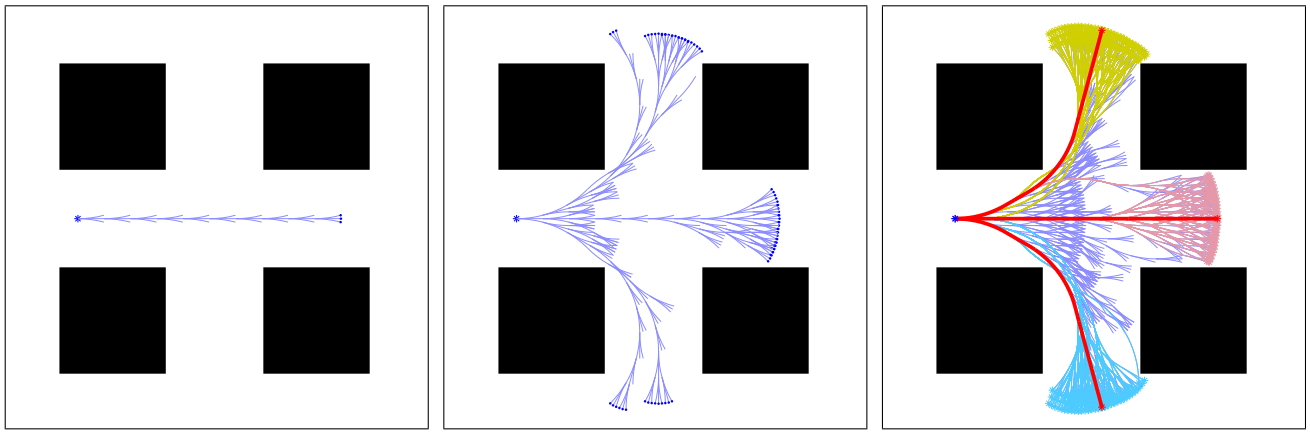
Algorithm 2 Clustering Threshold Adjustment

Require: $t, c > 1$
if $nextMinDist > t$ **then**
  if $numClusters \leq maxClusters$ **then**
    return $clusters$
  end if
  $t \leftarrow ct$
end if
continue clustering

V. Experimental Results

In the following, experimental results on real and simulation data are shown. The robot movement was simulated using a constant-curvature and constant-velocity kinematic bicycle model. The number of actions was restricted to 3, i.e. move straight, move left and move right, for some Δt. The path cost results from two individual costs: A cell traversal cost, which is calculated out of the cost map, and an action cost, which stays constant. The cost map may contain arbitrary cost values. In our case it combines information from the occupancy grid [20], i.e. the higher the occupancy value of a grid cell, the more expensive the cost, and an obstacle distance potential field, in particular the Voronoi field [10]. The parameters used are given in Table I.

978-1-4799-2914-613/$31.00 ©2013 IEEE

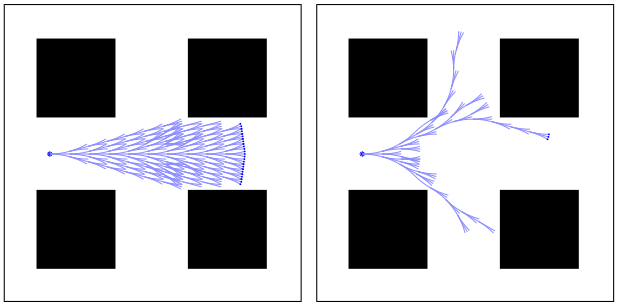


(a) Result after 10 iterations. (b) Result after 100 iterations. (c) Clustered result after 500 iterations.

Fig. 3. Results on simulation data. The blue star to the left represents the starting location and the dots possible found goal states. After 10 iterations the optimal path was found, (a). After 100 iterations the main directions were explored, (b). The final clustered result with class representatives in red, (c).

TABLE I
PARAMETERS USED IN THE TEST SCENARIOS

Parameter	Simulation	Constr. Site	Parking Lot
Grid size		512×512	
Cell size	0.2m	0.2m	0.3m
Path steps	10	10	40
Path length	60m	70m	60m
Max. clusters	5	5	5
Thresh. clustering	30m	5m	5m
Vehicle dim.	1m $\times$ 1m	5m $\times$ 2m	5m $\times$ 2m
Steering angle	$-2°, 0°, 2°$	$-4°, 0°, 4°$	$-30°, 0°, 30°$



(a) A* after 100 iterations. (b) RRT after first hit (42 iterations).

Fig. 4. Comparison to only A* and only RRT.

A. Simulation Scenario

The simulation data consists of a grid representing four quadratically-shaped obstacles, while all other cells are free space. The obstacles induce a strong topology on the scene and demonstrate the usefulness of the clustering. In Fig. 3(a) the result after the A* part is shown. After 10 iterations the first 3 path candidates were found and the algorithm switched to RRT. In Fig. 3(b) another intermediate result after 100 iterations is shown. The tree expanded uniformly into the unexplored regions. In Fig. 3(c) one can see the final result after 500 iterations. Three clusters were identified, which correctly represent the topology of the search space.

In Fig. 4 the proposed planner is compared to an A*-only and an RRT-only implementation. Similar to Fig. 3(b), Fig. 4(a) shows the result of the A*-only version after 100 iterations. One can clearly observe the strong focusing effects of A*. The additional iterations after the first path was found only lead to minor improvements in the explored search space. In contrast, an RRT-only implementation is very likely to need more iterations until the first candidate is found, Fig. 4(b). Apart from that, completeness and optimality is not guaranteed. With the combination of RRT and Hybrid-state A*, the benefits from both algorithms can be exploited.

B. Real-world Scenarios

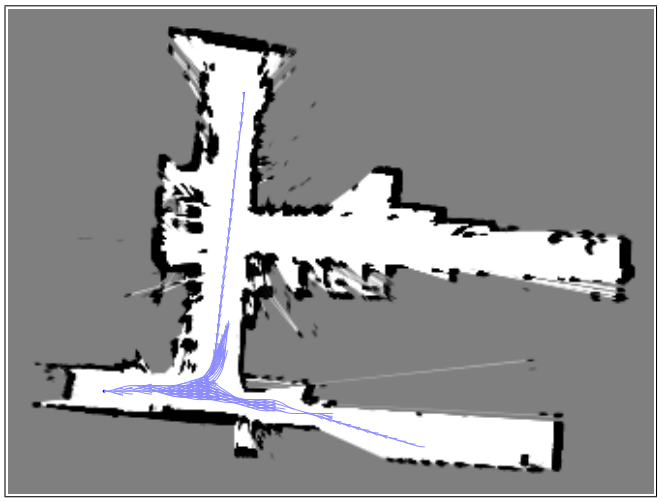
In Fig. 5 the algorithm was applied on occupancy grids built from laser data. Fig. 5(a) and 5(b) show a parking lot and Fig. 5(c) and 5(d) a road construction site. The parking lot scenario shows a dense, cluttered environment consisting of walls and parked vehicles. A* performs well in finding the first path and the search tree also grows towards a second branch. However, it is the combination RRT exploration that quickly finds also the lower-cost third branch.

The construction site scenario exhibits traffic cones that ought to guide the way for the vehicle. Each of the two intended routes were found as separate clusters. Additionally one other feasible, collision-free path cluster was detected, which also represent drivable, though not admissible, paths. Note that no domain-specific knowledge was used, e.g. information about the shape or the structure of the road boundary elements, other than the motion model of a vehicle.

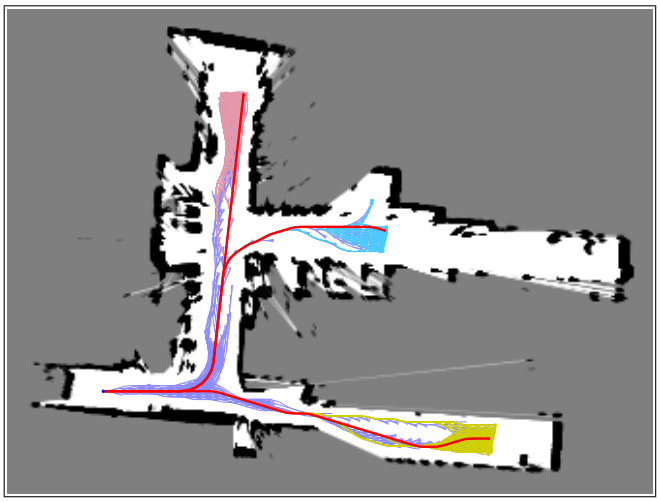
VI. CONCLUSION

This paper presented a path planning algorithm capable of planning feasible paths in the absence of goal poses, i.e. goal locations and orientations. We demonstrated the usefulness of such a planner in—but not restricted to—autonomous driving situations, where map data is not available or is out-of-date, or localization in the global map is not possible or insufficiently accurate. Additionally, we showed how individual paths can be clustered. Path clustering is an essential part of a planner that does not know the pose of the goal, since many path candidates can be reduced to a few principal paths that capture the main reachable topology.

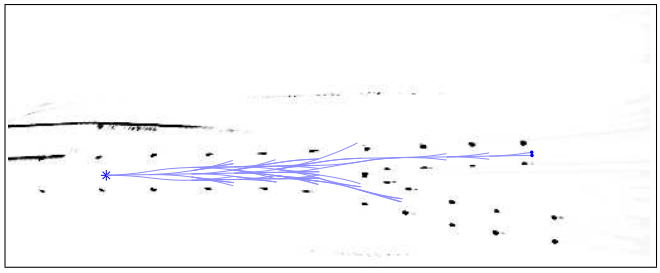
978-1-4799-2914-613/$31.00 ©2013 IEEE



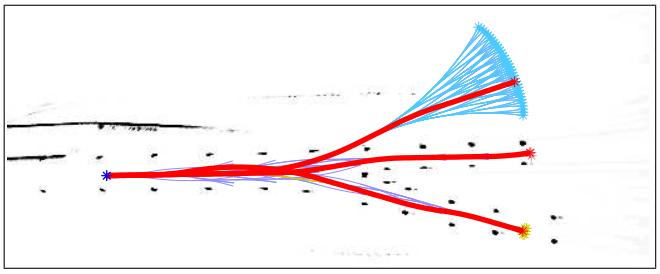
(a) First path after 325 iterations.



(b) Clustered result after 2000 iterations.



(c) First path after 52 iterations.



(d) Clustered result after 200 iterations.

Fig. 5. Results on an occupancy grid captured in a parking lot (top row) and a road construction site (lower row). In (a), (c) the result after A* and in (b), (d) after RRT and clustering are given.

ACKNOWLEDGMENT

This work was partially funded by the German Federal Ministry of Economics and Technology through the research initiative UR:BAN (www.urban-online.org).

REFERENCES

[1] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006.

[2] C. Urmson *et al.*, "Autonomous driving in urban environments: Boss and the urban challenge," *J. Field Robot.*, vol. 25, no. 8, pp. 425–466, 2008.

[3] M. Montemerlo *et al.*, "Junior: The stanford entry in the urban challenge," *J. Field Robot.*, vol. 25, no. 9, pp. 569–597, 2008.

[4] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[5] A. Stentz, "The focussed D* algorithm for real-time replanning," in *Proc. Int. Joint Conf. on Artificial Intelligence*, 1995, pp. 1652–1659.

[6] M. Likhachev, G. Gordon, and S. Thrun, "ARA*: Anytime A* search with provable bounds on sub-optimality," in *Proc. Conf. on Neural Information Processing Systems*, 2003.

[7] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, "Anytime dynamic A*: An anytime, replanning algorithm," in *Proc. Int. Conf. on Automated Planning and Scheduling*, 2005, pp. 262–271.

[8] A. Nash, K. Daniel, S. Koenig, and A. Feiner, "Theta*: Any-angle path planning on grids," in *Proc. AAAI Conf. on Artificial Intelligence*, 2007, pp. 1177–1183.

[9] D. Ferguson and A. Stentz, "Field D*: An interpolation-based path planner and replanner," in *Proc. Int. Symp. on Robotics Research*, 2005, pp. 1926–1931.

[10] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Path planning for autonomous vehicles in unknown semi-structured environments," *Int. J. Robot. Res.*, vol. 29, no. 5, pp. 485–501, 2010.

[11] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Computer Science Dept., Iowa State University, Tech. Rep. TR 98-11, 1998.

[12] C. Urmson and R. Simmons, "Approaches for heuristically biasing RRT growth," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, vol. 2, 2003, pp. 1178–1183.

[13] D. Ferguson and A. Stentz, "Anytime RRTs," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2006, pp. 5369–5375.

[14] K. Bekris, B. Chen, A. Ladd, E. Plaku, and L. Kavraki, "Multiple query probabilistic roadmap planning using single query planning primitives," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, vol. 1, 2003, pp. 656–661.

[15] K. Klasing, D. Wollherr, and M. Buss, "Cell-based probabilistic roadmaps (CPRM) for efficient path planning in large environments," in *Proc. Int. Conf. on Advanced Robotics*, 2007.

[16] E. Plaku, L. Kavraki, and M. Vardi, "Discrete search leading continuous exploration for kinodynamic motion planning," in *Proc. Robotics: Science and Systems*, 2007, pp. 326–333.

[17] M. Althoff, "Reachability analysis and its application to the safety assessment of autonomous cars," Ph.D. dissertation, Technische Universität München, 2010.

[18] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, 2nd ed. New York, NY, USA: John Wiley & Sons, Inc., 2001.

[19] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. Int. Conf. on Knowledge Discovery and Data Mining*, 1996, pp. 226–231.

[20] F. Homm, N. Kaempchen, J. Ota, and D. Burschka, "Efficient occupancy grid computation on the GPU with lidar and radar for road boundary detection," in *IEEE Intelligent Vehicles Symp.*, 2010, pp. 1006–1013.