2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

HRA*: Hybrid randomized path planning for complex 3D environments

Ernesto H. Teniente and Juan Andrade-Cetto

Abstract— **We propose HRA*, a new randomized path planner for complex 3D environments. The method is a modified A* algorithm that uses a hybrid node expansion technique that combines a random exploration of the action space meeting vehicle kinematic constraints with a cost to goal metric that considers only kinematically feasible paths to the goal. The method includes also a series of heuristics to accelerate the search time. These include a cost penalty near obstacles, and a filter to prevent revisiting configurations. The performance of the method is compared against A*, RRT and RRT* in a series of challenging 3D outdoor datasets. HRA* is shown to outperform all of them in computation time, and delivering shorter paths than A* and RRT.**

I. INTRODUCTION

In mobile robotics, state-of-the art dense mapping techniques are capable of producing impressively rich 3D environment models [1], [2]. Yet, producing efficient trajectories on large complex outdoor environments for off-road robots is still a challenging task. The main reasons being the irregularity of the surface and the need to account for the nonholonomic constraints of the vehicles.

In this paper we present HRA*, a method to plan safe trajectories in rich complex 3D environments that guarantees reachability at a desired robot pose with significantly lower computation time than competing alternatives. Our path planner incrementally builds a tree using the A* algorithm. However, it includes a hybrid cost policy to efficiently expand the search tree, we combine random sampling from the continuous space of kinematically feasible motion commands with a cost to goal metric that also takes into account the vehicle nonholonomic constraints.

To speed up node search, our method also includes heuristics to penalize node expansion near obstacles, with a penalty proportional to the inverse distance to collision; and to limit the number of explored nodes. The method book-keeps visited cells in the configuration space, and disallows node expansion at those configurations in the first full iteration of the algorithm.

Our experiments show that HRA* compares favorably against A*, RRT and RRT*, in terms of computation time, and generates significantly shorter paths than A* and RRT.

This work has been partially supported by the Mexican Council of Science and Technology with a PhD Scholarship to Ernesto Teniente, by the Spanish Ministry of Science and Innovation under project DPI-2011-27510 and by the EU project ARCAS FP7-287617.

The authors are with the Institut de Robòtica i Informàtica Industrial CSIC-UPC, Llorens Artigas 4-6, 08028 Barcelona. {ehomar,cetto}@iri.upc.edu.

II. RELATED WORK

Grid-based path planning algorithms, such as the methods based on the A* [3], connect cells of a discretized configuration space from a start configuration to a goal configuration. To jump from cell to cell, they typically explore the action space with a deterministic minimal set of control parameters. For instance, a skid steer vehicle moving on the plane could have an action set of the form $\{-v_{max}, v_{max}\} \times \{-\omega_{max}, 0, \omega_{max}\}$, with a time interval according to the size of cells in the configuration space [4]. A number of heuristics can be used to modify the resolution search. The time interval for which the kinematic model is unrolled can be related to the amount of clutter near the explored node, for instance, with trajectory lengths proportional to the sum of the distance to the nearest obstacle plus the distance to the nearest edge in a Voronoi diagram of obstacles [5]. Besides modifying the integration time, one can also condition the search speed according to a cost to goal heuristic. This is typical of best-first algorithms such as A* in which the priority queue is sorted according to some cost to goal heuristic [6].

Our method combines these two ideas. We propose a heuristic to modify the resolution search depending on the amount of clutter near the explored node, but instead of generating a Voronoi diagram as in [4], we add to the policy cost a penalty proportional to the inverse distance to a collision. In this way, configurations near obstacles will be sampled sparsely, whereas configurations in open space will have more chances of being tested. Furthermore, we include a measure of the distance to the goal to our cost policy, the same way as in [6], computed from Dubins paths [7], which give the shortest length path between the current configuration and the goal configuration.

Randomized sampling algorithms for path planning such as PRM or RRT [7] explore the action space stochastically. By randomly exploring a continuous action space, RRT has the property of being probabilistically complete, although not asymptotically optimal [8]. RRT* [9], [10] solves this problem by triggering a rewire process each time a node is added to the tree. The rewiring process searches for the nearest neighbor in configuration space and decides whether it is less costly to move to the new node from the explored parent or from such closest neighbor. In very cluttered environments however, RRT* may behave poorly since it spends too much time deciding whether to rewire or not. In our method, we also sample actions randomly from a continuous action space, but in the search for neighbors within the tree, we trigger a rewiring test only when these actions reach a

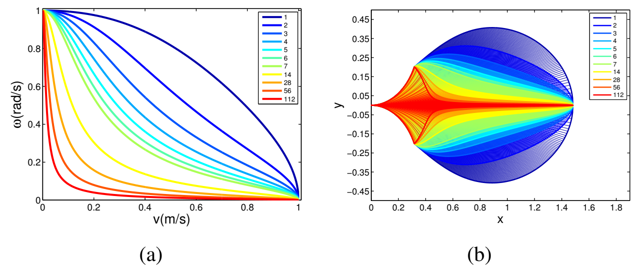

Fig. 1. Reachable curvature sets. a) Normalized turning radius and velocity correlation for different values of l. b) Integrated trajectories for different values of l and $\Delta t = 1$)

previously visited cell in an auxiliary maintained discretized configuration space.

A more recent randomized motion planning strategy is cross entropy motion planning [11]. The method samples in the space of parameterized trajectories building a probability distribution over the set of feasible paths and searching for the optimal trajectory through importance sampling. Cross entropy motion planning is only applicable to problems for which multiple paths to the goal can be computed during algorithm execution.

III. Hybrid Randomized Path Planning

The task is to find a feasible path $\tau(t) : [t_0, ..., t_n] \rightarrow \mathcal{C}_{\text{free}}$, if it exist, in a configuration space free of collisions $\mathcal{C}_{\text{free}}$, from an initial robot configuration $\tau(t_0) = x_I$ to a goal robot configuration $\tau(t_n) = x_G$.

Ideally, we would like to compute the minimum cost path

$$\tau^* = \arg\min(c(\tau) : \tau) \tag{1}$$

but will be content with obtaining a low cost path with a reasonable computation effort. To that end, a number of heuristics aimed at pruning the search space and to bias the exploration towards the goal will be devised.

In our planning context we are faced with nonholonomic motion constraints for the vehicle, and it is through a control sequence $u(t) : [t_0, ..., t_n] \rightarrow \mathcal{U}$, $u(t_i) = (v_i, \omega_i, \Delta t_i)$, that we can move from one robot configuration to another.

A. Steering Functions

We use two motion control policies. The first is the forward kinematics of the vehicle and the second one is an optimal control policy that guarantees goal reach. Our mobile robot is modelled as a Dubins vehicle:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \omega \end{bmatrix}, \qquad |\omega| \le v/\rho \tag{2}$$

where (x, y) is the vehicle position, θ is the vehicle heading, v is the translational velocity, ω is the angular velocity, and ρ is the vehicle minimum turning radius constraint.

1) Robot Forward Kinematics: To compute a new robot position for search tree expansion, we solve Eq. 2 using the second order Runge-Kutta method. This derivation computes the new robot position more precisely than the Euler based solution.

$$\begin{aligned} x_{i+1} &= x_i + v_i \Delta t_i \cos\left(\theta_i + \frac{\omega_i \Delta t_i}{2}\right) \\ y_{i+1} &= y_i + v_i \Delta t_i \sin\left(\theta_i + \frac{\omega_i \Delta t_i}{2}\right) \\ \theta_{i+1} &= \theta_i + \omega_i \Delta t_i \end{aligned} \tag{3}$$

The robot motion commands (v_i, ω_i) are subject to minimum and maximum speed constraints $v_i \in [v_{\text{min}}, v_{\text{max}}]$ and $\omega \in [-\omega_{\text{max}}, \omega_{\text{max}}] | (\omega = 0 \Rightarrow v_{\text{max}}), (|\omega| = \omega_{\text{max}} \Rightarrow v_{\text{min}})$. This definition imposes the maximum speed constraint $v^2 + \omega^2 \le 1$ [7].

We want to compute random velocity commands subject to this constraint. To avoid significant rotation at high speeds we introduce a parameter $l \ge 1$ designed to control the relation between the translational and rotational velocities. The mechanism to sample motion commands is as follows. We draw random translational velocities from the fixed interval $v_{\text{rand}} = [0, 1]$ and compute its corresponding rotational velocity $\omega(v_{\text{rand}}) = \sin(\arccos(v_{\text{rand}}))$. We then scale them with

$$\begin{aligned} v_l &= l\, v_{\text{rand}}\,, \qquad & y &= 1 - v_{\text{rand}} \\ r &= \sqrt{v_l^2 + y^2}\,, \qquad & \omega_l &= \sin(\arccos(v_l/r)) \end{aligned} \tag{4}$$

In this way, we have a gauge to modify the robot behavior. See Fig.1. The original velocity commands on the sphere occur at $l = 1$. Finally, we scale these values to the range given by the translational and rotational velocity

$$v_i = v_{\text{min}} + v_l(v_{\text{max}} - v_{\text{min}}) \tag{5}$$

$$\omega_i = \omega_{\text{max}} \omega_l s\,, \quad s \sim \text{rand}(-1, 1) \tag{6}$$

Finally, the sampling period Δt_i is randomly selected from a small interval $[t_{\text{min}}, t_{\text{max}}]$, and the new robot pose x_{i+1} and trajectory $\tau(i + 1)$ are computed by integrating Eq. 3 over small time steps $\delta t << \Delta t_i$.

The strategy essentially limits large curvature values at high translational speeds, producing smoother trajectories.

2) Locally Optimal Control Policy: Now that we have a mechanism to sample configurations from a search node x_i, we still need a way to measure the cost to reach goal completion. To that end, we compute the length of a Dubins curve from the computed configuration x_{i+1} all the way to the x_G.

Dubins curves [7] are optimal paths made up of circular arcs connected by tangential line segments under the maximum curvature constrain $\kappa = 1/\rho$. In our application we restrict to only use the CLC type and they are specified by a combination of left, straight, or right steering inputs, leading to only only four types of paths (RSL, LSR, RSR, LSL).This will give us a locally optimal control policy and a locally optimal path $\tau(t)$.

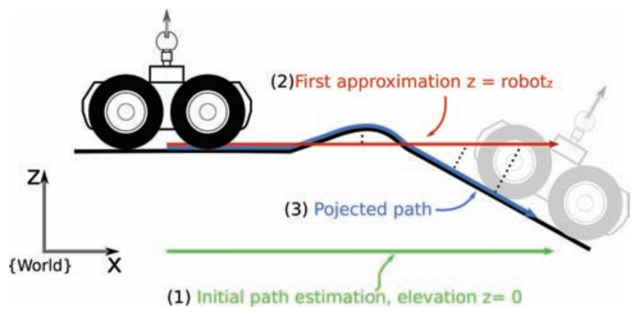
(2)First approximation z = robotz
Z
{World}
X
(3) Pojected path
(1) Initial path estimation, elevation z= 0

Fig. 2. Proposed path projection step, used to estimate the local path and the final robot orientation.

B. Cost Estimation

Our cost unit is time, and as in [12], we split the cost into accumulated cost c_{acc}, the time we have integrated the kinematic model so far to go from x_I to x_{i+i}, and a cost to the goal c_{goal}, the optimistic bound given by the Dubins curve from x_{i+i} to x_G, assuming the remaining path is free of obstacles. Similar to [13], we also use upper and lower bounds, but in a slightly different manner. The upper bound c^+ is used during node selection to improve exploration adding heuristics that bound the node search. The lower bound c^- is used to determine the best path from the solutions set, i.e, c^- will be the cost to minimize in Eq. 1.

The accumulated cost is computed with

$$c_{\text{acc}} = \sum_{k=1}^{i+1} \Delta t_k \tag{7}$$

and the upper and lower bounds are computed with,

$$c^- = c_{\text{acc}} + d_{\text{goal}}/v_{\text{max}} \tag{8}$$

$$c^+ = c_{\text{acc}} + d_{\text{goal}}/v_{\text{max}} + \sum_{j=1}^{n} h_j \tag{9}$$

where d_{goal} is the optimal path length of the Dubins curve to the goal, and h_j are each of the heuristics we use to penalize a path.

We can use as many as desired penalization heuristics h_j to make some configurations less attractive to the planner. Our first heuristic is a local penalty inversely proportional to the distance to the nearest obstacle in front of the robot. To compute it, we raytrace the environment at the robot's heading and measure the distance to the nearest obstacle in front of the robot $d_{\mathcal{C}_{\text{obs}}}$. We define the penalty $h_1 = \kappa_1/d_{\mathcal{C}_{\text{obs}}}$, where κ_1 is a user selected factor to weight the contribution of this heuristic in c^+.

Another penalty h_2 occurs when $\tau_{\text{goal}} \subset \mathcal{C}_{\text{free}}$, i.e. the Dubins path has a clear path to the goal. Then, $h_2 = \infty$ and $h_2 = 0$ otherwise. This way we avoid to explore that node again. This is a termination condition because τ_{goal} is locally optimal so it is not possible to find a better solution from that configuration to the goal.

C. Path projection

The presented control policies compute segment paths parallel to a plane. These paths could be either on a local planar patch, or with respect to a global coordinate frame. These paths however, contain no information about their elevation or orientation with respect to the 3D surface. In [14] the authors use Lagrangian interpolation to project 2D roadmap streamlines to the 3D terrain model, and a feasibility test for each 3D candidate path is computed. In [15], the robot is assumed to move at small distances to little planar patches, and robot transitions between patches is assumed with constant velocity. To compute the 3D path, we propose also a projection of the 2D local paths to the 3D surface, but in contrast to [14], we not only validate feasibility, but an iterative nearest neighbor search is used to refine the local path.

Local paths are computed parallel to the world xy plane. The elevation value for each new configuration in the path is initialized with that of its parent node. Then, a NN search is performed to find its closest traversable point in the 3D map, substituting the z coordinate with that of the NN. If the difference between any two neighbor configurations in the 3D local path is larger than a threshold (i.e., if the hill is deemed too steep), we enter in a loop searching for a new NN assignment. See Fig. 2. Finally, as in [16], we use local planar information to compute the robot orientation for the last configuration in the path.

D. Collision Detection

Once the local 3D path is computed, we need to check again whether it remains collision free. If $\tau(t)$ is found to be in collision, we keep the path segment that lies in $\mathcal{C}_{\text{free}}$, i.e., $\tau(t) = [t_0, ..., t_m] \subset \mathcal{C}_{\text{free}}$, and the configuration $x_{m-\lambda}$ is added to the search tree and we update the path and the commands. At that point, a new exploration step is triggered. The parameter λ is introduced to avoid configurations blocked by obstacles. Notice that if $(t_m - t_{m-\lambda} - \Delta t_m) \leq 0$, the algorithm would return to a collision.

E. Tree Rewire

The rewiring procedure proposed in this work is different from that in [9], [10]. Instead of using vicinity information around a volume or the k-nearest neighbors for each new added node, we use a similar approach to that in [5]. We maintain an auxiliary grid of the configuration space $\mathcal{C}_\tau$, and mark visited cells on it. If a cell was previously visited, we compare the cost of the new and the old paths. In contrast to [5], we do allow rewiring even if the valid configuration at that cell has children. Alg. 1 overviews HRA*, our hybrid randomized path planning method.

IV. Experiments

A. Experimental Setup

We show the performance of HRA* in three different outdoor datasets of uneven challenging terrain. Two of them were gathered using a custom built 3D laser with a Hokuyo UTM-30LX scanner mounted in a slip-ring. The first dataset

```
HybridRandomizedPathPlanner($x_I, x_G, I, K, \mathbf{X}$)
    Input:
        $x_I$: Initial configuration.
        $x_G$: Goal configuration.
        $I$: Maximum number of iterations.
        $K$: Number of motion samples per iteration.
        $\mathbf{X}$: 3D map.
    Output:
        $\tau^*$: Path.
        $\mathbf{u}^*$: Commands.

1: $\mathcal{T}.init(x_I, \mathbf{X})$
2: $\mathcal{C}_{\text{free}} \leftarrow$ ComputeDiscretizedCspace($\mathbf{X}$)
3: $\mathcal{C}_\tau \leftarrow$ UpdateAuxiliarCspace($\emptyset, \tau_0$)
4: $solution = 0$
5: for $i = 1$ to $I$ do
6:     $q_{\text{best}} \leftarrow$ GetBestNode($\mathcal{T}$)
7:     $u \leftarrow$ GenerateRandomCommands($K$)
8:     for $k = 1$ to $K$ do
9:         $(x_k, \tau_k) \leftarrow$ ForwardKinematics($\mathcal{T}, u_k, q_{\text{best}}$)
10:        $m \leftarrow$ CollisonCheck($\tau_k, \mathcal{C}_{\text{free}}, \mathbf{X}$)
11:        if $m$ then
12:            $(x_k, \tau_k, u_k) \leftarrow$ UpdateData($m, x_k, \tau_k, u_k$)
13:        end if
14:        if $\sim solution$ then
15:            $visited \leftarrow$ AreVisitedCells($\tau_k$)
16:        end if
17:        if $(\sim m) \wedge (\sim visited)$ then
18:            $q_{old} \leftarrow$ GetNodeInCell($x_k, \mathcal{C}_{\text{free}}$)
19:            if $q_{old}$ then
20:                $\mathcal{T} \leftarrow$ TreeRewire($\mathcal{T}, q_{\text{best}}, x_k, q_{old}$)
21:            else
22:                $\mathcal{C}_\tau \leftarrow$ UpdateAuxiliarCspace($\mathcal{C}_\tau, \tau_k$)
23:                $c_{\text{acc}} \leftarrow$ IncrementCost($x_k, \mathcal{T}$)
24:                $(c_{\text{goal}}, \tau_{\text{goal}}) \leftarrow$ DubinsToGoal($x_k, x_G$)
25:                $m \leftarrow$ CollisonCheck($\tau_{\text{goal}}, \mathbf{X}$)
26:                $\mathcal{H} \leftarrow$ ComputePenalties
27:                $(c^+, c^-) \leftarrow$ ComputeBounds($c_{\text{acc}}, c_{\text{goal}}, \mathcal{H}$)
28:                $\mathcal{T} \leftarrow$ TreeUpdate($\mathcal{T}, x_k, \tau_k, u_k, c^+, c^-$)
29:            end if
30:            if $\sim m$ then
31:                $(\mathbf{u}^*, \tau^*) \leftarrow$ SelectShortestPath($\mathcal{T}$)
32:                $solution = 1$
33:            end if
34:        end if
35:    end for
36: end for
37: Return $(\mathbf{u}^*, \tau^*)$
```

Algorithm 1: HRA*: Hybrid Randomized Path Planning

was acquired in the Facultat de Matemàtiques i Estadística (FME), located at the Campus Sud at the Universitat Politècnica de Catalunya (UPC). The sensor is mounted atop a Segway RPM400 mobile robot called TEO. The dataset includes 39 dense 3D scans collected on different traversed surfaces (soil, grass and gravel), TEO was driven over bumps of about 10-15 cm height. In the second data set the sensor was mounted atop a Pioneer 3AT platform. The Pioneer gathered 400 points clouds. This is the Barcelona Robot Lab dataset (BRL) [17]. The BRL is a permanent area for mobile robots experiments in the UPC Campus Nord in Barcelona and covers 10000 m^2. The third dataset is part of the Canadian Planetary Emulation Terrain 3D Mapping Dataset [18]. It includes five subsets that emulate planetary terrain. We use the boxmet (BM) subset which has 112 scans and covers 7200 m^2. To validate and compare our method we extracted six different scenarios from these datasets: four from the FME, and one from each of the BRL and BM datasets. See Fig. 3. We analyzed different situations in real static environments such as: narrow passages, cluttered obstacles, bay-like situations, ramps, forks and more.

B. Experimental Conditions

For the RRT and RRT* algorithms, we run 50 Monte Carlo simulations with a maximum number of iterations $I = 20,000$ for each of the five scenarios.

For the A* algorithm, we used 7 motion primitives as described in Sec. III-A. In this case, experiments were run only once since the A* solutions are unique. We set $I = 10,000$ for the FME scenarios and $I = 20,000$ for the BRL and BM scenarios. To allow for a more fare comparison, rewiring is allowed as described in Sec. III-E.

For HRA*, we run two sets of 50 Monte Carlo simulations each, $I = 10,000$ iterations, and we enable the heuristics h_1, h_2, and rewiring. In the first set, HRA*1, cell bookkeeping is disabled, whereas in the second set, HRA*2, bookkeeping is enabled. The parameter l limiting the ratio between translational and angular velocities was selected empirically, by testing HRA*2 for different values of l. A value of $l = 4$ is a good compromise between path quality improvement and execution time. The other parameters were chosen as $\kappa_1 = 0.1m$ and $d_{\text{goal}} = 0.3m$.

When using cell bookkeeping we consider that a path is new if the ration between new cells and visited cells is larger than 1%. Incredibly, such as small percentage of new cells, was enough to obtain a significant boost in computation speed. The discretization of the configurations space was set to $(\Delta x, \Delta y, \Delta\theta) = (0.3m, 0.3m, 5deg)$.

Experiments were run in MATLAB. Some functions where implemented in C++ and embedded in mex files. We use the ANN library [19] for the nearest neighbor search. All experiments are executed on an Intel Core i7-2720 system @ 2.20 GHZ, with 8 GB of RAM memory running Ubuntu 12.04 64 bits.

C. Results

Table I shows the first and best computed path lengths for all methods. The reported algorithms's first computed path length corresponds to the mean value of the first length obtained for each of the Monte Carlo runs. As for the best path lengths reported, these correspond to the minimum path lengths obtained over all Monte Carlo runs. The table shows that the two versions of the proposed approach HRA*1, and HRA*2 are able to compute path lengths comparable to A* with rewiring in all datasets, and significantly shorter than RRT.

Note how cell bookkeeping (HRA*2) improves the best solution on almost all scenarios when compared to (HRA*1) at the expense of a possible larger first solution. The reason is that ell bookkeeping enforces sparsity in exploration gaining speed in finding the first solution. Our method is able to compute the first solution faster in all but one of the scenarios. This is shown in Table II in which we report mean first and best solution computation times for all Monte Carlo runs. Note also how RRT is the fastest method in computing its best solution, whereas RRT* is an order of magnitude slower when compared to all the other methods. The reason is because, each reconnection step involves a sum of costly operations: computing Dubins paths, reprojection,

	Scenario											
	FME1		FME2		FME3		FME4		BRL		BM	
Method	First	Best	First	Best	First	Best	First	Best	First	Best	First	Best
A* with rewiring	18.12	17.63	33.84	33.93	18.61	17.84	36.97	36.47	14.77	14.77	**30.97**	30.83
HRA*1	16.52	16.34	34.54	32.92	18.29	17.39	36.48	35.98	15.64	14.46	31.03	30.50
HRA*2	16.40	15.73	35.42	32.28	18.50	17.28	36.59	35.50	15.79	14.50	31.25	30.59
RRT*	**15.58**	**14.73**	**33.48**	**30.55**	**18.10**	**16.57**	**36.35**	**33.55**	**14.76**	**13.73**	31.58	**29.60**
RRT	19.57	15.89	46.17	35.66	23.29	17.38	44.09	37.46	21.18	15.92	41.16	32.25

TABLE I

FINAL PATH LENGTH IN METERS COMPUTED FOR ALL METHODS. FIRST AND BEST SOLUTIONS.

	Scenario											
	FME1		FME2		FME3		FME4		BRL		BM	
Method	First	Best	First	Best	First	Best	First	Best	First	Best	First	Best
A* w/ rewiring	2.01	95.75	10.04	**10.05**	51.77	83.64	20.62	**21.93**	184.90	184.90	425.82	799.41
HRA*1	2.37	132.92	16.88	62.10	18.31	47.59	14.01	76.65	24.89	36.84	308.43	**371.68**
HRA*2	**1.20**	342.02	13.55	396.63	**6.05**	169.37	**10.51**	139.66	**4.70**	191.02	**47.14**	724.43
RRT*	113.18	2555.42	387.82	4164.13	239.80	2074.17	655.41	3538.28	271.45	1516.01	1311.05	3840.55
RRT	1.89	**25.06**	**8.58**	33.20	20.85	**41.89**	14.71	40.82	10.42	**21.54**	415.37	611.86

TABLE II

COMPUTATION TIMES IN SECONDS FOR ALL METHODS. FIRST AND BEST SOLUTIONS.

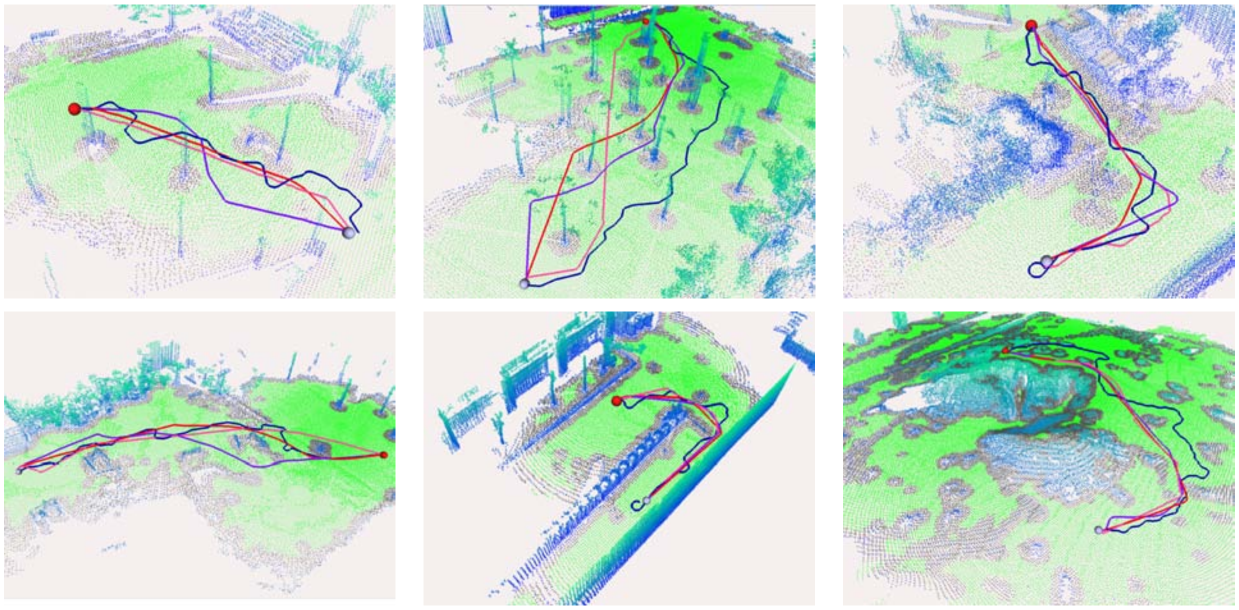

Fig. 3. Path planning scenarios and computed paths. The scenarios, from upper left to bottom right, correspond to the FME1, FME2, FME3, FM4, BRL and BM subsets. The green dots indicate the traversable areas and the degraded blue and gray dots represent non-traversable regions. The start and goal positions are indicated by red and gray spheres, respectively. The resulting paths are: A* with rewiring (magenta), HRA* (red), RRT* (pink) and RRT (dark blue).

and collision detection, which for RRT* is computed for all neighbors within a radius, a significantly larger number of times than in our approach, in which bookkeeping is performed.

Fig. 3 shows the computed paths for the six scenarios and all of the above-mentioned methods, but HRA*1. Notice that even when RRT* has smaller paths than HRA*, the paths computed by our method are smoother. Note also that the proposed approach cannot be as good as RRT* in finding the shortest path, because it uses discretized configuration space for bookkeeping and rewiring, whereas RRT* explores the entire nearest neighbor set.

Finally, an interesting metric to compare path planning methods is the length vs time plot. This is, how quickly can any given method compute a solution with some given quality, say a fixed path length. This computational complexity per performance improvement plots are given in Fig. 4 for all the scenarios explored. The plots nicely show how RRT* is the most expensive algorithm able to compute shortest paths at computation times an order of magnitude larger than the rest of the methods. On the other side of the scale, RRT is the fastes of them all, but the path lengths it computes are in general larger than the rest of the methods. Furthermore, the plots also show that increased computational time does not necessarily mean better solutions with respect to path length for the case of RRT. Our proposed strategies HRA*1 and

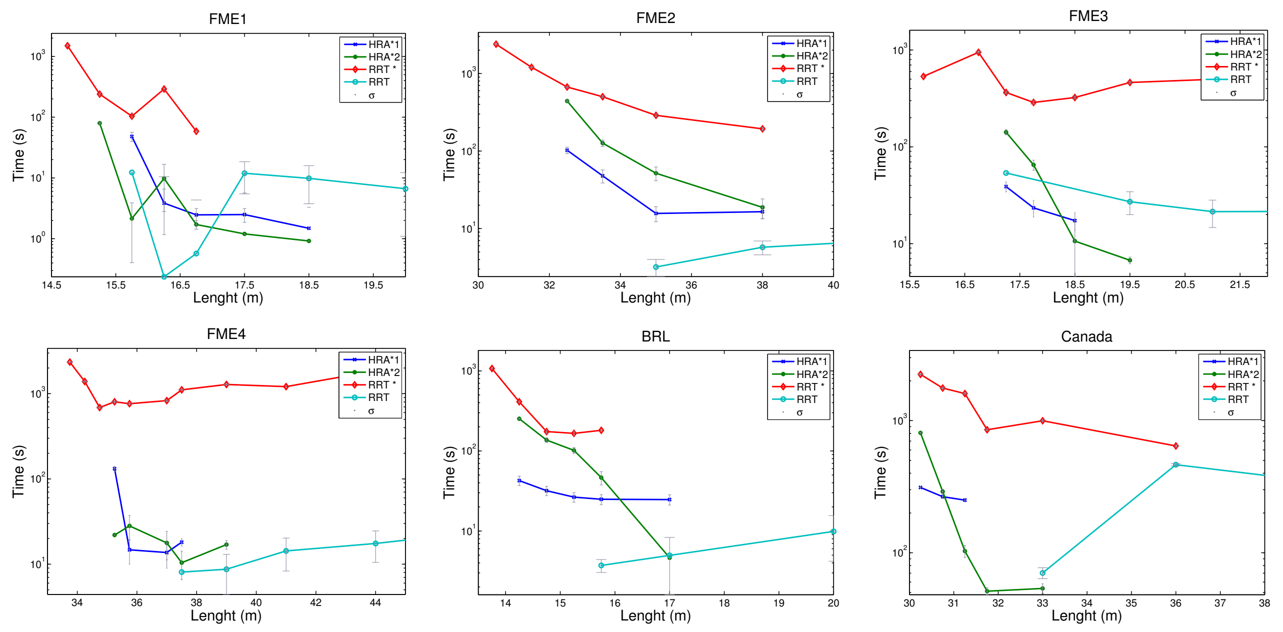

Fig. 4. Length vs. time plots for the two HRA* proposed methods, RRT and RRT*. The greay ticks show one standard deviation bounds from the various Monte Carlo runs. Note the logarithmic scale of the time axis.

HRA*2, – randomized action sampling with heuristic cost penalties, with and without bookkeeping –, outperform their counterpart methods in both ends. HRA* implementations take significantly less time to compute solutions with the same quality as those of RRT*, and are also able to compute shorter paths than RRT for the same alloted execution time.

V. Conclusions

We presented a method to compute paths for a mobile robot in outdoor challenging environments. The method, called HRA*, is a modified A* algorithm that uses a hybrid node expansion technique that combines a random exploration of the action space meeting vehicle kinematic constraints, with a cost to goal metric that considers only kinematically feasible paths to the goal. The method is extended with a number of heuristics to penalize configurations near obstacles that accelerate search time. The technique was successfully tested on several real outdoors environments and was shown to outperform A* with rewiring, RRT and RRT* in computation time, and A* with rewiring and RRT in path length.

References

[1] O. Wulf, A. Nüchter, J. Hertzberg, and B. Wagner, "Benchmarking urban six-degree-of-freedom simultaneous localization and mapping," *J. Field Robotics*, vol. 25, no. 3, pp. 148–163, Mar. 2008.

[2] R. Valencia, E. Teniente, E. Trulls, and J. Andrade-Cetto, "3D mapping for urban service robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Saint Louis, Oct. 2009, pp. 3076–3081.

[3] R. Dechter and J. Pearl, "Generalized best-first search strategies and the optimality of A*," *J. ACM*, vol. 32, no. 3, pp. 505–536, Jul. 1985.

[4] J. Barraquand and J.-C. Latombe, "Nonholonomic multibody mobile robots: controllability and motion planning in the presence of obstacles," in *Proc. IEEE Int. Conf. Robot. Autom.*, Sacramento, Apr. 1991, pp. 2328–2335.

[5] 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.

[6] M. Hwangbo, J. Kuffner, and T. Kanade, "Efficient two-phase 3D motion planning for small fixed-wing UAVs," in *Proc. IEEE Int. Conf. Robot. Autom.*, Rome, Apr. 2007, pp. 1035–1041.

[7] S. LaValle, *Planning Algorithm*. Cambridge University Press, 2006.

[8] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, 2011.

[9] ——, "Optimal kinodynamic motion planning using incremental sampling-based methods," in *Proc. IEEE Conf. Decision Control*, Atlanta, Dec. 2010, pp. 7681–7687.

[10] S. Karaman, M. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the RRT*," in *Proc. IEEE Int. Conf. Robot. Autom.*, Shanghai, May 2011, pp. 1478–1483.

[11] M. Kobilarov, "Cross entropy motion planning," *Int. J. Robot. Res.*, vol. 31, no. 7, pp. 855–871, 2012.

[12] E. Frazzoli, M. Dahleh, and E. Feron, "Real-time motion planning for agile autonomous vehicles," *J. Guid. Control Dyn.*, vol. 25, no. 1, pp. 116–129, 2002.

[13] Y. Kuwata, S. Karaman, J. Teo, E. Frazzoli, J. How, and G. Fiore, "Real-time motion planning with applications to autonomous urban driving," *IEEE Trans. Contr. Syst. Technol.*, vol. 17, no. 5, pp. 1105–1118, 2009.

[14] D. Gingras, E. Dupuis, G. Payre, and J. de Lafontaine, "Path planning based on fluid mechanics for mobile robots using unstructured terrain models," in *Proc. IEEE Int. Conf. Robot. Autom.*, Anchorage, May 2010, pp. 1978–1984.

[15] M. B. Kobilarov and G. S. Sukhatme, "Near time-optimal constrained trajectory planning on outdoor terrain," in *Proc. IEEE Int. Conf. Robot. Autom.*, Barcelona, Apr. 2005, pp. 1821–1828.

[16] T. Simeon and B. Dacre-Wright, "A practical motion planner for all-terrain mobile robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Tokyo, Jul. 1993, pp. 1357–1363.

[17] E. Teniente, M. Morta, A. Ortega, E. Trulls, and J. Andrade-Cetto, "Barcelona Robot Lab data set," [online] http://www.iri.upc.edu/research/webprojects/pau/datasets/BRL/php/dataset.php, 2011.

[18] C. Tong, D. Gingras, K. Larose, T. Barfoot, and E. Dupuis, "The canadian planetary emulation terrain 3d mapping dataset," [online]http://asrl.utias.utoronto.ca/datasets/3dmap/, 2012.

[19] D. M. Mount and S. Arya, "ANN: A library for approximate nearest neighbor searching," in *Proc. 2nd Fall Workshop Comput. Comb. Geom.*, Durham, Oct. 1997.