Combining Cost and Reliability for Rough Terrain Navigation

Jun-young Kwak, Mihail Pivtoraiko, and Reid Simmons
The Robotics Institute, Carnegie Mellon University
{junyoung.kwak, mihail, reids}@cs.cmu.edu

Abstract

This paper presents an improved method for planetary rover path planning in very rough terrain, based on the particle-based Rapidly-exploring Random Tree (pRRT) algorithm. It inherits the benefits of pRRT, an improvement over the conventional RRT algorithm that explicitly considers uncertainty in sensing, modeling, and actuation by treating each addition to the tree as a stochastic process. Although pRRT is well-suited to planning under uncertainty, it has limitations in minimizing the cost of path plans. Our approach addresses these limitations by considering the relevant cost functions explicitly. Such cost functions depend on the application and can include time or distance of traversal, and energy consumption of the rover. The paper demonstrates the planner performance using a specific cost function defined in terms of the energy expenditure. The improved pRRT algorithm has been experimentally validated in simulation, and it has been shown to produce lower-cost plans than the standard pRRT algorithm. The proposed approach is likely to benefit the present and future space missions as an onboard motion planner and as a ground-based tool for plan validation.

1. Introduction

Over the past few years, experience with rover navigation on Mars has shown that a number of significant challenges still exist with rover autonomy. Generating correct and safe rover motions is complicated by a number of challenges, including:

1. environment constraints (cluttered obstacles, high wheel slip),
2. uncertainty of information about the environment,
3. rover constraints (kinematics and dynamics constraints),
4. resource constraints (energy expenditure).

Motion planning under partially-known, yet certain environment constraints has received considerable attention in robotics research, and some standard solutions exist [4] [6]. Planning under nonholonomic constraints also has been studied extensively [5]. However, planning with uncertainty is still among the most active areas of inquiry [7] [8] [9].

The particle-based Rapidly-exploring Random Tree (pRRT) algorithm has been shown to be effective at solving planning problems under uncertainty [1] [3]. It extends the RRT algorithm and inherits its capability to satisfy challenges 1 and 3 above. In addition, pRRT is also capable of computing the likelihood of successful execution of motion plans (i.e. probability of achieving the goal by following the plan). It selects a plan that is likely to succeed as the solution, thereby satisfying the challenge 2. However, it is not designed to consider any other performance measures. For example, in selecting a motion plan, it may be important to evaluate a trade-off between the likelihood of reaching the goal and a desired cost function. Examples of application-dependent motion costs could be energy expenditure, time or distance of traversal and others, including any combination of the above.

With respect to applying pRRT to rover motion planning, this can be an important limitation. Specifically, it leaves the challenge 4 unmet. The approach presented here addresses this limitation by considering arbitrary cost functions explicitly. The proposed motion planner therefore represents an incremental improvement over state of the art. It provides the complete capability of pRRT, while also considering a representation of the path's cost. Thus, the proposed method is a promising solution for rover motion planning, since it satisfies *all* important challenges of the problem, outlined above.

The proposed algorithm has been experimentally validated in simulation. The results confirm that it becomes possible to leverage a trade-off between the cost function of choice and the likelihood of achieving the goal. This property is an important advantage over the state of the art and is likely to enable unprecedented rover autonomy in planetary terrain.

2. Related work

Previous work in path planning has taken several approaches to planning with uncertainty. One of the most common approaches is to ensure proper operation in the worst case scenario. For example, if uncertainty is considered only in actuation, but not in sensing or modeling, Hait and Sim´eon [7] consider the range of possible rover poses and test for impact with the terrain. Related work by Esposito in the domain of plan validation [8] samples several possible values of the uncertain parameter from a given distribution and repeats planning for each value. In the more general case, approaches such as Iagnemma's [9] computes the cost metric of traversing a particular region based on the worst case estimate of uncertainty.

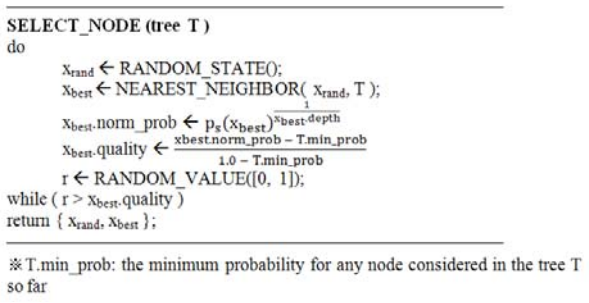
Most of previous path planning works assume that the environment is completely known before the robot begins its traverse (see Lavalle [10]). The optimal algorithms in this search a state space (e.g., visibility graph, grid cells) using the distance transform [13] or heuristics [14] to find the lowest cost path from the robot's start state to the goal state. Cost can be defined to be distance travelled, energy expended, time exposed to danger, etc.

There was also an approach to use the adaptive path planning algorithm based on the cost function. Cunningham and Roberts [15] present the way for cooperating unmanned air vehicles. A key requirement of the algorithm is that it adapts the subnets (and paths through the subnets) in response to the change of environment. More complex situations can be derived as combinations of several factors around robots. In this algorithm, subnet and path adaptation is driven by a global cost function that essentially shifts sensors into and out of subnets to reach a minimum cost.

3. The particle-based Rapidly-exploring Random Tree (pRRT) algorithm

In this work, we use the particle RRT (pRRT) algorithm [1] [3] which is an extension to the Rapidly-exploring Random Tree (RRT) algorithm introduced by Lavalle and Kuffner [10]. RRT is a widely-used algorithm for motion planning in high-dimensional spaces with kinodynamic constraints. Each iteration of the algorithm, as depicted in figure 3(a), begins with a tree of states that the rover can reach. At the first step, this tree only consists of the initial state of the rover. A new state x_{rand} is chosen stochastically from the state space, and the nearest node x_{best} in the tree is determined. An action is estimated to reach x_{rand} from x_{best}, and the action is executed. The resulting state x_{new} is added to the tree. The planner may compute the forward simulation of the action with any level of fidelity appropriate to the task at hand. It is not necessary for the final state x_{new} to coincide with x_{rand}, so it is often preferable to select the action using simple inverse kinematics, but simulate the result of the action using more accurate dynamic models.

The pRRT algorithm selects a pair of x_{rand} and x_{best} as the same way of RRT. However, the selected x_{best} is evaluated based on the quality value, the probability of reaching x_{best} from the root of the tree. A random value, r is drawn from a uniform distribution between 0 and 1, and if x_{best}.quality > r, the pair of points x_{rand} and x_{best} is accepted and an extension is attempted. Otherwise, a new pair of points is chosen. This procedure is called a *rejection test*. The detailed procedure is listed in figure 1. The extension introduced by pRRT, as shown in figure 3(b), produces distributions of states, rather than single states, at each node of the tree. The distributions are nonparametric, and are derived from the uncertainty specified as input to the algorithm, and the forward simulation process itself. Specifically, they use a set of discrete particles to estimate the distribution at each node. For each extension added to the tree, several particles are computed under uncertain environment. To compute a single particle, one particle from the node is chosen as the start state, and a value is drawn from the prior distribution over the uncertain parameter (in this case, terrain friction). Simulations of the same action under different values for friction will result in different final states for the rover. In this procedure, CM Lab's Vortex Simulator [16] is used to get particles under the uncertain factor with a specific vehicle model (See figure 2). After simulating several times in this manner, the resulting particles are clustered into one or more nodes, which are added to the tree with the same parent. The clustering is used to separate qualitatively different particles into different



```
SELECT_NODE (tree T)
do
        $x_{rand} \leftarrow$ RANDOM_STATE();
        $x_{best} \leftarrow$ NEAREST_NEIGHBOR( $x_{rand}$, T );
        $x_{best}$.norm_prob $\leftarrow p_s(x_{best})^{\frac{1}{x_{best}.depth}}$
        $x_{best}$.quality $\leftarrow \frac{xbest.norm\_prob - T.min\_prob}{1.0 - T.min\_prob}$
        r $\leftarrow$ RANDOM_VALUE([0, 1]);
while ( r > $x_{best}$.quality )
return { $x_{rand}$, $x_{best}$ };
```

※T.min_prob: the minimum probability for any node considered in the tree T so far

Figure 1. The pRRT algorithm - SELECT_NODE

nodes. This procedure is accomplished using a hierarchical clustering tree [11] with a weighted Euclidean distance metric to determine the difference between particles. The hierarchical clustering tree algorithm uses this metric to iteratively agglomerate the particles and clusters separated by the shortest distance.

In general, building a planning tree while accounting for uncertainty should result in nodes whose variance grows with the depth in the tree. However, since the clustering step permits a single extension to be broken into more than one node, the variance is split as well. In addition, the probability of a single particle is associated with the prior likelihood of the sampled value of friction used to produce it. Since the prior over the uncertain parameter can have an arbitrary probability distribution function and nodes may contain different numbers of particles, some nodes will contain more probability mass than others. Nodes which combine both low variance and high probability are good candidates for path planning, because the rover is expected to be able to reach them accurately despite the uncertainty. Consequently, such a path is executed with greater accuracy by the rover. Thus, as the planning tree is built, there is a bias towards extending new leaves from such high-probability paths. This bias is implemented using a selection mechanism similar to that introduced by Urmson in [2].

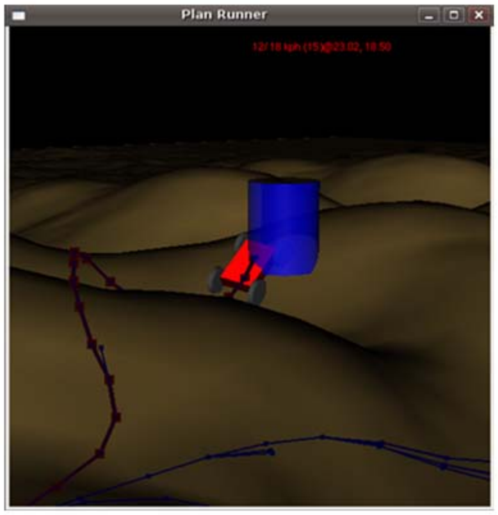


Figure 2. The Vortex simulator

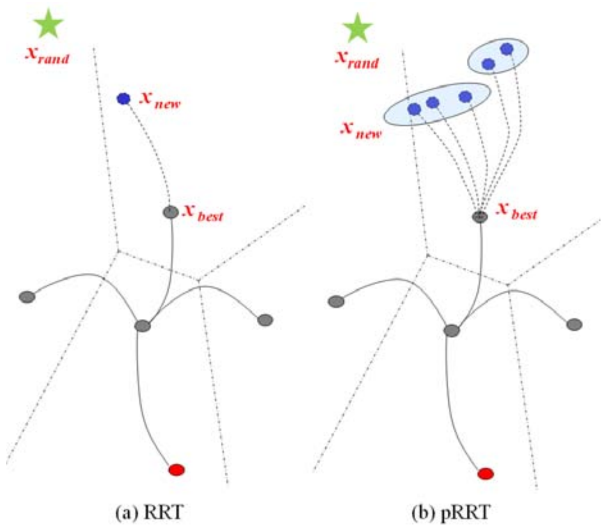


Figure 3. The RRT & pRRT algorithms

4. Enhancing pRRT with cost functions

The advantage of the proposed approach over pRRT lies in a more informed method of growing the search tree. It is guided by a cost function that represents the true cost of moving the rover from one state to another. Here we extend the pRRT algorithm to utilize the additional information, provided by the cost function. First, we will describe the mechanism for fusing the cost function estimates with the probability of successful execution of the plan. Afterwards, we will discuss the details of extending the pRRT algorithm to accommodate the new methodology for guiding the tree growth.

4.1. Weighted reward function

In order to facilitate the fusion of the cost function with the probability of successful plan execution, we use the notion of *reward*. It is loosely defined as the inverse of the cost. The mapping of tree nodes to the value of their exploration in pRRT search is referred to as a *reward function*. This function guides the growth of the tree toward the goal. The nodes that have a higher value of the reward function represent a more promising direction of search. Thus, the quality of the pRRT solution is more likely to be higher if the algorithm grows the tree toward nodes with the highest reward.

Since here we propose combining two reward functions – the conventional probability of successful execution and cost function of choice –, we require a methodology for fusing them, such that the resulting reward function still satisfies the requirements of pRRT. We propose a fusing technique that is loosely related to weighted averaging, hence the resulting function is referred to as a *weighted reward function*. The technique we developed was inspired by [12] and Urmson's hRRT technique for heuristically biasing RRT growth [2]. The weighted reward function of a node *n*, denoted as *W[R(n)]*, is defined in the following equation.

$$W[R(n)] = P_q(n)e^{-\alpha C(n)} \quad (\alpha > 0: a\ reward\ decay\ factor)$$

$$P_q(n) = \frac{p_{norm}(n) - T_m}{1.0 - T_m}$$

$$p_{norm}(n) = p_s(n)^{\frac{1}{d}} \quad \text{(d: the depth of node n in the tree)}$$

In the equation, $p_s(n)$ is the probability of reaching a state *n* by following the plan from the starting point. The variable $p_{norm}(n)$ is the normalized $p_s(n)$ by the path length, *d*. T_m is the minimum probability of all leaf nodes of the search tree. $P_q(n)$ is a quality value of a state *n*; it is normalized to ensure that the weight falls within the range [0, 1]. *C(n)* is the total accumulated cost function for a state *n* from the starting point.

The rationale for normalizing $p_{norm}(n)$ is as follows. We found that the probability of reaching nodes of the tree drops quickly with path length. This causes the algorithm to favor making extensions from nodes near the root, even when reasonably likely nodes exist closer to the goal. Hence, the path probability $p_{norm}(n)$ is normalized using the path length in order to encourage more extensions from nodes farther from the root.

The definition for *W[R(n)]* includes an exponential function in order to constrain its values between 0 and 1. Thus, the weighted reward value will also reside within [0, 1]. This might bring several advantages. For example, we can set a threshold value for a rejection test in the pRRT algorithm because we know the scope of the value explicitly. Also, because we use accumulated cost, *C(n)* increases very quickly. By utilizing the exponential function, large differences in cost are translated to small differences in state space. However, the reduction of a large cost difference via the exponential may make it difficult to compare costs of nodes. This is addressed with the relevant parameter α, a reward decay factor. It represents the degree of importance of reaching the goal quickly and is problem-specific. Section 5 briefly discusses choosing the value of this parameter, as well as the sensitivity of the present method to its value.

4.2. Extending pRRT with weighted reward function

Given a weighted reward function for guiding the growth of the tree, we extend the pRRT algorithm to accommodate it. Through this extension we achieve our goal of enabling the pRRT planner to incorporate arbitrary cost functions in its search. Relatively small changes to pRRT are required: only one routine of the algorithm, *SELECT_NODE*, needs to be modified.

There are two leading approaches for extending the pRRT. Below we describe the details of both, specify their re-designed *SELECT_NODE* routines, and conclude with outlining their benefits and drawbacks.

4.1.1. Algorithm 1. In this approach, we select the new node by picking a point x_{rand} stochastically and selecting the nearest neighbor x_{best} of the point x_{rand}. For the rejection test, we use the weighted reward function to evaluate the selected pair (x_{rand}, x_{best}). The algorithm is listed in figure 4.

4.1.2. Algorithm 2. In this version, we pick a point x_{rand} stochastically and select a node x_{best} with the maximum reward. For the rejection test, we use the weighted reward to evaluate the selected pair (x_{rand}, x_{best}). The algorithm is listed in figure 5.

The difference between Algorithms 1 and 2 is in selecting the point x_{best}. After picking a point x_{rand} stochastically, Algorithm 1 selects the nearest neighbor x_{best} of x_{rand}. On the other hand, Algorithm 2 selects x_{best} with the maximum reward. To calculate the maximum reward value, we combine a normalized distance and a weighted reward with a relevant weight factor, w_f. Because Algorithm 1 selects a pair (x_{rand}, x_{best}) based on the distance and does a rejection test with a weighted reward, we can get a pair with a reasonably good weighted reward on average. However, Algorithm 1 considers a weighted reward after selecting a pair, so we cannot guarantee the

```
SELECT_NODE (tree T )
do
        $x_{rand}$ ← RANDOM_STATE();
        $x_{best}$ ← NEAREST_NEIGHBOR( $x_{rand}$, T );
        $x_{best}$.norm_prob ← $p_s(x_{best})^{\frac{1}{x_{best}.depth}}$
        $x_{best}$.quality ← $\frac{xbest.norm\_prob - T.min\_prob}{1.0 - T.min\_prob}$
        $x_{best}$.weighted_reward ← $x_{best}$. quality × $e^{-\alpha C(x_{best})}$
        r ← RANDOM_VALUE([0, 1]);
while ( r > $x_{best}$.weighted_reward )
return { $x_{rand}$, $x_{best}$ };
```

※T.min_prob: the minimum probability for any node considered in the tree T so far

Figure 4. Algorithm 1

```
SELECT_NODE (tree T )
do
        $x_{rand}$ ← RANDOM_STATE();
        $x_{best}$ ← GET_NODE_WITH_MAX_REWARD( $x_{rand}$, T );
        $x_{best}$.normalized_prob ← $p_s(x_{best})^{\frac{1}{x_{best}.depth}}$
        $x_{best}$.quality ← $\frac{xbest.norm\_prob - T.min\_prob}{1.0 - T.min\_prob}$
        $x_{best}$.weighted_reward ← $x_{best}$. quality × $e^{-\alpha C(x_{best})}$
        r ← RANDOM_VALUE([0, 1]);
while ( r > $x_{best}$.weighted_reward )
return { $x_{rand}$, $x_{best}$ };
```

```
GET_NODE_WITH_MAX_REWARD( $x_{rand}$, tree T )
$d_{max}$ ← GET_MAX_DISTANCE( $x_{node} \in T$, $x_{rand}$ );
while ( $x_{node} \in T$ ) do
    d ← DISTANCE_BETWEEN( $x_{node}$, $x_{rand}$ );
    wr ← $x_{node}$.weighted_reward
    CR ← $-w_f \frac{d}{d_{max}} + (1 - w_f) \cdot wr$    ($w_f$: weight factor, $0 \le w_f \le 1$)
return $x_{best}$ ← $\mathrm{argmax}_{x_{node} \in T}\, CR(x_{node})$
```

Figure 5. Algorithm 2

selected pair has the highest reward, and if a threshold value is too high, it might take a long time to select a pair. Algorithm 2 is considering the distance and the weighted reward together, so we can select a pair based on our needs which are problem-specific.

The choice of 1 for w_f would result in considering only the distance in selecting the point x_{best}. This would be identical to Algorithm 1. On the other hand, if we choose 0 for w_f, we would only use the weighted reward to select x_{best}. Some guidelines for choosing the value of w_f, as well as sensitivity to this parameter, are offered in Section 5.

It is worth noting the benefits and drawbacks of the proposed algorithms. In Algorithm 2, we perform a rejection test with a weighted reward, even though we have already considered that value when we calculated the maximum reward. We take into account the weighted reward again in the computation of the maximum reward which gives more weight to a distance metric for choosing w_f. If the pair is selected based on the maximum reward, it would likely have a higher weighted reward. However, the best pair in terms of the maximum reward does not necessarily mean the pair having a high weighted reward. Therefore, with a rejection test using weighted reward, it would likely to get the better plan having higher weighted reward value than the case without a rejection test. In addition, since the weighted reward for each node has been already computed at the previous step, this rejection test does not require any additional computational resources. Also, because Algorithm 2 considers the maximum reward at every step, it is more likely to select a better pair even though the computation in terms of memory is more expensive than Algorithm 1.

5. Experimental results

In this section, we demonstrate the performance of the proposed algorithms using the specific cost function. Cost may be computed as a combination of several factors such as distance traveled, energy consumed, etc. In this experiment, cost is a synonym for energy consumption, so the cost function becomes:

$$C(n) = C(n_p) + \sum_{t=1}^{m} \sum_{i=1}^{k} \tau_i^t \cdot \omega_i^t$$

where τ_i^t and ω_i^t are torque value and angular velocity for wheel$_i$ at time step t. k is the number of wheels and m is the number of time steps. n_p is a parent node of a state n. In the pRRT algorithm, each node is composed of multiple particles, and thus the cost of a node means the average cost of the particles.

For the simulation, the challenge is to find the value for w_f that gives the optimal values for the following variables: planning time and total cost. The minor concern is to find the value for α since it does not affect the results of two algorithms relative to the other parameter (i.e. w_f). Specifically we picked α = 0.005, which has been shown to be appropriate for both algorithms in the environment. α is used as a scale factor to get the reasonable range of reward value. Of course we cannot say this value is the optimal one in this case. However, once we can compare each algorithm's performance based on cost and reward values with the selected α value, optimality is not necessarily required. In addition, simulation results were not sensitive to α value in the experiments. We changed α value from 0.001 to 0.1, but there were no significant effects in the results. For Algorithm 2, in order to acquire the appropriate w_f value, 300 iterations were repeatedly applied to both algorithms with a given terrain map. Based on the experimental results in table 1, we picked w_f = 0.7 which gives the best result

Table 1. Picking w_f

w_f	Planning time (s)	Total cost $C(n)$
1.0	124.39	15.64
0.9	117.26	15.45
0.7	103.88	15.01
0.5	153.71	15.23
0.3	231.75	14.98
0.1	352.35	14.87
0.0[1]	n/a	n/a

[1] With w_f = 0.0, the procedure of the proposed algorithms is terminated by the maximum iteration number of pRRT before getting a path reaching to the goal position.

in terms of planning time and total cost. Of course, w_f depends on the needs which are problem-specific or evaluation criteria. Therefore the selected w_f value is not generally optimal. However, we can at least get some ideas to select the relevant weight factor for each case from values in table 1 and Algorithm 2 described in figure 5.

With these two selected parameter values and a given terrain map, the procedure of each algorithm was iterated for 150 times to get average path length, planning time, total cost, and relative end-point errors. As a result, the three patterns as depicted in figure 6 were produced. The path of pattern 1 has the shortest length, but it requires the highest overall cost to get the goal position because it needs more energy when it climbs over the obstacle. The path of pattern 2 and 3 goes around the obstacles, and pattern 3 has the longest length among three cases. We show the detailed results in table 2 for each case.

When we generated the plan with the original pRRT algorithm, over 90% of paths showed pattern 1. Because pRRT considers only a distance metric to select a pair and just perform a rejection test based on a quality value, average planning time showed the best result among three algorithms. However, pRRT was the worst in terms of total cost. On the other hand, when we applied Algorithm 1 and 2, the planner generated paths that avoid the obstacle like pattern 2 and 3 with about 95%. Algorithm 1 and 2 generated the path with lower total cost than the path generated by the original pRRT because less energy is needed along paths of pattern 2 and 3. The interesting thing here is even though the total cost of pattern 3 is much higher than pattern 2, pattern 3 is chosen with relatively high frequency, about 30%, in Algorithm 1 and 2. This is because the proposed algorithms give still more weight to the distance metric for their rejection tests. As a result, if a random point is selected on the right side of the space, a node relatively close to the random point and having reasonable weighted reward would likely be selected.

For the planning time, as we expected, Algorithm 1 took the longest time on average because it considers the distance metric like pRRT, and this procedure is iterated until the pair with a reasonable weighted reward is selected to pass a rejection test. In fact, for all three algorithms, all nodes in the tree need to be investigated to select x_{best} to extend, and thus there is no significant difference in time when they select the pair at each step. However, after the pair (x_{rand}, x_{best}) is selected, when the rejection test is applied to the chosen pair, it is iterated until the pair has a reasonably good value in terms of a given evaluation criteria. This

Table 2. Detailed results for each algorithm

		pRRT	Alg1	Alg2
Pattern 1	Frequency (%)	90.67	4.67	5.33
	Path length (m)	9.58	9.62	9.81
	Planning time (s)	85.25	90.52	87.33
	Total cost *C(n)*	22.52	23.03	24.17
	Relative end-point error (m)	0.053	0.076	0.058
Pattern 2	Frequency (%)	6.67	61.33	69.34
	Path length (m)	11.32	11.52	11.45
	Planning time (s)	112.23	122.69	115.53
	Total cost *C(n)*	12.04	12.47	12.36
	Relative end-point error (m)	0.061	0.097	0.064
Pattern 3	Frequency (%)	2.66	34.00	25.33
	Path length (m)	18.52	20.21	18.25
	Planning time (s)	132.53	152.93	130.21
	Total cost *C(n)*	18.04	20.05	17.53
	Relative end-point error (m)	0.084	0.112	0.81
Average	Path length (m)	9.93	14.39	13.09
	Planning time (s)	88.31	131.47	117.75
	Total cost *C(n)*	21.70	15.54	14.29
	Relative end-point error (m)	0.054	0.101	0.068

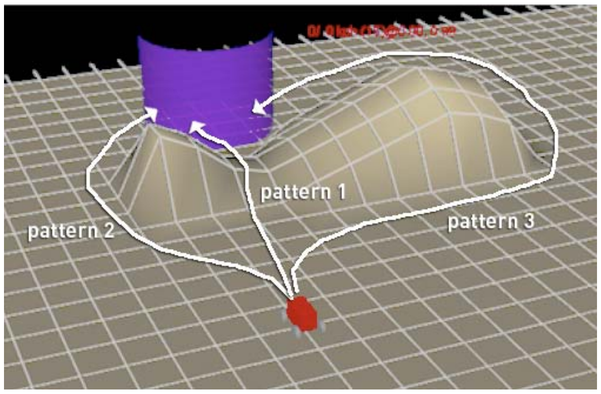


Figure 6. Three patterns of generated paths with each algorithm

procedure mainly affects overall planning time, and that's why pRRT and Algorithm 2 have nearly identical planning time for the same pattern, even though average planning time between pRRT and Algorithm 2 is fairly different because of the frequency of patterns. In cost criteria, Algorithm 2 showed the best result because it considers the weighted reward at every step when it selects the pair.

The reason why we mainly get the paths of pattern 2 or 3 with the proposed algorithms can be explained more clearly in the specific case. In figure 7, nodes inside a dotted circle were not extended more because those nodes already have fairly high overall cost. Specifically, even though those nodes are selected,

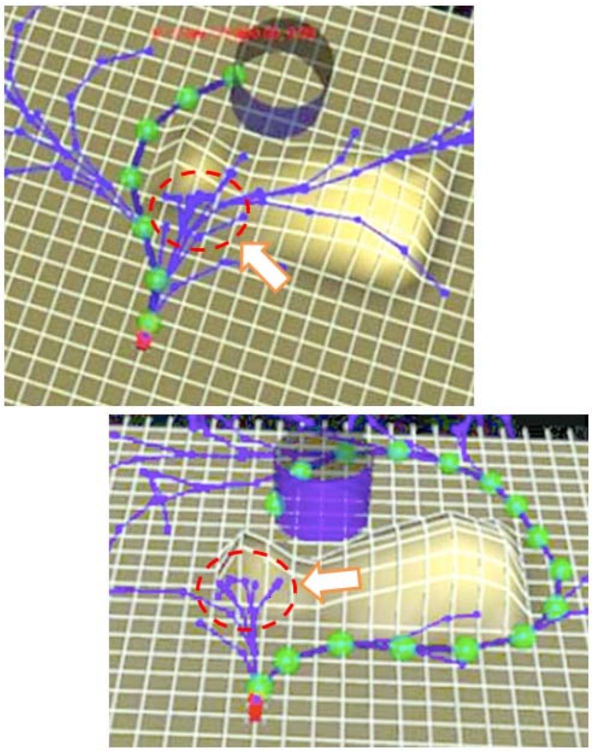


Figure 7. Nodes with higher total cost: These nodes are not selected by a *rejection test.*

they are rejected by *SELECT_NODE* procedure shown in the previous section. Therefore, the proposed algorithms select nodes which are not resided on the hill to extend the tree, and thus they produce the paths going around the obstacle.

According to the results in table 2, generating a plan based on the aggregated cost function slightly deteriorated the end-point accuracy of the plan, but the plan has lower overall cost. These results show a trade-off between different metrics of success, specifically, accuracy, planning time, path length and energy consumption.

6. Conclusions & future work

This paper presented an improved method for planetary rover path planning in rough terrain, based on the pRRT algorithm. Our contribution enables adding arbitrary cost functions to the pRRT algorithm, in order to allow the resulting planner to consider a trade-off between different metrics of success, such as accuracy, path length, and energy consumption. The method of incorporating the cost functions in pRRT has been experimentally validated in simulation.

Future work includes investigating additional cost criteria and methods for integrating them in the proposed algorithm. We would also like to undertake field validation of the algorithm, as well as evaluate its use for future space missions as an onboard rover motion planner and as a ground-based tool for plan validation.

Acknowledgments

This work has been funded by NASA under grant number NASA-03-OSS-01. We thank Nik and other colleagues at CMU and the JPL Robotics section for their support in developing and testing our algorithm.

References

[1] N. A. Melchior and R. Simmons, “Particle RRT for path planning with uncertainty,” in *Proceedings of IEEE International Conference on Robotics and Automation*, April 2007.

[2] C. Urmson and R. Simmons, “Approaches for heuristically biasing RRT growth,” in *IEEE/RSJ IROS 2003*, October 2003.

[3] N. A. Melchior, J. Kwak and R. Simmons, “Particle RRT for Path Planning in very rough terrain,” *NASA Science Technology Conference 2007 (NSTC 2007)*, 2007

[4] J. Carsten, A. Rankin, D. Ferguson and A. Stentz, “Global Path Planning On-board the Mars Exploration Rovers”, in *Proc. of the IEEE Aerospace Conference*, 2007.

[5] S. M. LaValle, *Planning Algorithms*, Cambridge University Press, 2006.

[6] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz and S. Thrun, “Anytime Dynamic A*: an Anytime, Replanning Algorithm”, in *Proc. of the Int. Conf. on Automated Planning and Scheduling*, 2005.

[7] A. Hait and T. Sim´eon, “Motion planning on rough terrain for an articulated vehicle in presence of uncertainties,” *IEEE/RSJ International Symposium on Intelligent Robots and Systems*, pp. 1126–1133, 1996.

[8] J. M. Esposito, J. Kim, and V. Kumar, “Adaptive RRTs for validating hybrid robotic control systems,” *International Workshop on the Algorithmic Foundations of Robotics*, July 2004.

[9] K. Iagnemma, F. Genot, and S. Dubowsky, “Rapid physics-based rough-terrain rover planning with sensor and control uncertainty,” in *Proceedings of IEEE International Conference on Robotics and Automation*, Detroit, MI, 1999, pp. 2286–2291.

[10] S. M. Lavalle and J. J. Kuffner, “Randomized kinodynamic planning,” *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, May 2001.

[11] A. K. Jain, M. N. Murty, and P. J. Flynn, “Data clustering: a review,” *ACM Computing Surveys*, vol. 31, no. 3, pp. 264-323, 1999.

[12] L. Lita, J. Schulte and S. Thrun, “A System for Multi-Agent Coordination in Uncertain Environments,” *Proceedings of the fifth international conference on Autonomous agents*, Montreal, Quebec, Canada, 2001.

[13] Jarvis, R. A., “Collision-Free Trajectory Planning Using the Distance Transforms,” *Mechanical Engineering Trans. of the Institution of Engineers*, Australia, Vol. ME10, No. 3, September, 1985.

[14] Nilsson, N. J., *Principles of Artificial Intelligence*, Tioga Publishing Company, 1980.

[15] C. Cunningham and R. Roberts, “An Adaptive Path Planning Algorithm for Cooperating Unmanned Air Vehicles,” *IEEE International Conference on Robotics and Automation*, Seoul, Korea, May 2001.

[16] CMLab Vortex: http://www.cm-labs.com/products/vortex/