# Particle RRT for Path Planning with Uncertainty

Nik A. Melchior and Reid Simmons

***Abstract*— This paper describes a new extension to the Rapidly–exploring Random Tree (RRT) path planning algorithm. The Particle RRT algorithm explicitly considers uncertainty in its domain, similar to the operation of a particle filter. Each extension to the search tree is treated as a stochastic process and is simulated multiple times. The behavior of the robot can be characterized based on the specified uncertainty in the environment, and guarantees can be made as to the performance under this uncertainty. Extensions to the search tree, and therefore entire paths, may be chosen based on the expected probability of successful execution. The benefit of this algorithm is demonstrated in the simulation of a rover operating in rough terrain with unknown coefficients of friction.**

## I. Introduction

The Rapidly–exploring Random Tree (RRT) algorithm [1] is a popular technique for path planning with kinodynamic constraints. In simple terms, RRT builds a search tree of reachable states by attempting to apply random actions at known–reachable states. Unless the action causes the robot to make contact with an obstacle or violate some dynamics constraint, the action is considered successful, and the resulting state is added to the tree of reachable states. A simulator is generally treated as a black box that determines the result of an action given the robot's initial state. This allows the algorithm to be applied to domains where complex system dynamics make analytic control difficult or impossible. RRT has been successfully applied to wheeled and legged vehicles, as well as underwater robots and aircraft.

However, this binary decision as to the success of an action can limit the application of this algorithm in two important ways. First, it does not allow for ranking or scoring multiple actions which may succeed from a given initial state. Actions often have associated costs (such as the energy or time required for execution), and planning may produce several paths from start to goal with varying cumulative costs. High–cost extensions might therefore be avoided in hopes of finding a better path, but these extensions should not be ignored completely, because a better path may not exist. Most RRT implementations at least consider path length in terms of Euclidean distance travelled, but other notions of path cost should be incorporated as well.

The second important use for a fuzzy notion of action success is when knowledge of the obstacles or dynamic properties of the environment cannot be precisely known. The application presented in this paper is a rover navigating through unknown terrain. Stereo vision is used to build a model of the terrain in the immediate area, but the accuracy of this model decreases with distance from the rover, and occlusions are possible. In addition, characteristics of the terrain such as the coefficient of friction can be estimated only roughly, but these characteristics can have substantial impact on the rover's behavior, particularly when traversing rough terrain. Path planning can benefit from explicitly considering the uncertainty in the terrain to generate paths with high likelihood of success.

melchior@cmu.edu, reids@cs.cmu.edu. The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213-3890.

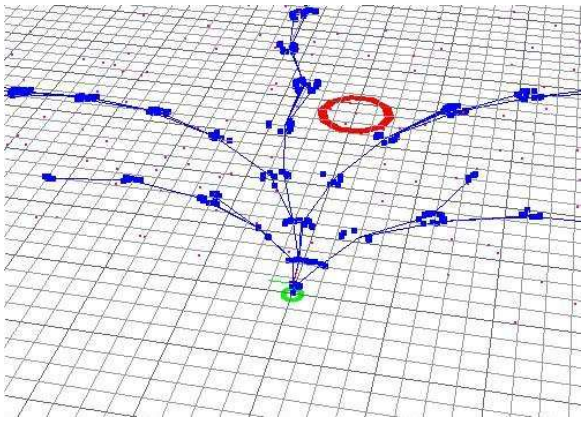


Fig. 1. A pRRT tree with several particles at each node

Related work in [2] has considered a changing environment in the form of moving obstacles by predicting the motion of these obstacles. If the obstacles move in an unexpected manner, the path is simply replanned. One approach for planning in uncertain terrain [3] ensures that the planned actions produce the same result for the entire range of expected values of the unknown conditions. Another application [4] builds a forest of search trees, using a different value for each tree.

This paper presents an extension to the RRT algorithm that directly addresses the issue of plans under uncertainty with a novel approach. The Particle RRT (pRRT) algorithm facilitates the creation of an RRT in an uncertain environment by propagating that uncertainty to the planned path. Each extension to the search tree is attempted several times under different likely conditions. Nodes in the search tree are created by clustering the results from these simulations. The likelihood of successfully executing each action is quantified so that the probability of following entire paths may be determined. An example search tree is shown in figure 1. Experimental results from simulations are presented, showing that this approach results in paths that are more robust to uncertainty.

## II. Algorithm Description

The pRRT algorithm extends the basic RRT algorithm to operate efficiently in an environment with characterizable uncertainty. The discussion in this paper considers a rover driving over terrain whose friction is not known. However, other types of uncertainty can be incorporated into this algorithm, such as imprecise action execution or obstacles whose size or location are uncertain.

### A. Basic RRT Algorithm

The Rapidly-exploring Random Tree (RRT) algorithm is a randomized algorithm useful for exploring large states spaces that cannot be searched exhaustively. The algorithm, listed in figure 2 and depicted in figure 3, iteratively chooses a random point $p$ in the state space and attempts to extend the current search tree toward that point. The RANDOM_STATE function may choose from a uniform distribution over the state space, but it typically includes some bias toward the goal. The extension is performed by considering the random point $p$, and its nearest neighbor $q$, within the tree $\mathcal{T}$. The algorithm owes its rapid exploration to the bias implicit in this procedure. A node $q$ is extended only when it is the nearest neighbor of the random point $p$. This means that $p$ must lie within the Voronoi region of $q$, and nodes on the frontier of the search tree generally have the largest Voronoi regions.

The NEW_STATE function determines an action $u_{new}$ that observes any dynamic constraints on the robot, and which, when applied to the robot at state $q$, results in some new state $x_{new}$. It is important to note that $x_{new}$ is *in the direction of* $p$ from $q$, but $x_{new}$ and $p$ are not expected to coincide. Indeed, it would be quite a coincidence if they did, since the RRT algorithm does not require that the inverse kinematics of the robot be known. Only the forward kinematics are required to determine $x_{new}$ given $q$ and $u_{new}$. If NEW_STATE finds a new state and action without violating dynamic constraints or colliding with an obstacle, the new state and action are added to the tree. Thus, NEW_STATE typically makes a binary decision as to the success of an extension, even when there is uncertainty in the environment, and thus in the outcome of a simulation. It is this disparity that we seek to remedy.

BUILD_RRT($x_{init}$)
1: $\mathcal{T}$.init($x_{init}$)
2: **while** ($x_{goal} \notin \mathcal{T}$) **do**
3: $\{p, q\} \leftarrow$ SELECT_EXTENSION($\mathcal{T}$);
4: EXTEND($\mathcal{T}, p, q$)
5: Return $\mathcal{T}$

SELECT_EXTENSION($\mathcal{T}$)
1: $p \leftarrow$ RANDOM_STATE();
2: $q \leftarrow$ NEAREST_NEIGHBOR($p, \mathcal{T}$);
3: Return $\{p, q\}$

EXTEND($\mathcal{T}, p, q$)
1: **if** (NEW_STATE($p, q, x_{new}, u_{new}$)) **then**
2: $\mathcal{T}$.add_vertex($x_{new}$);
3: $\mathcal{T}$.add_edge($q, x_{new}, u_{new}$);

Fig. 2. The RRT algorithm

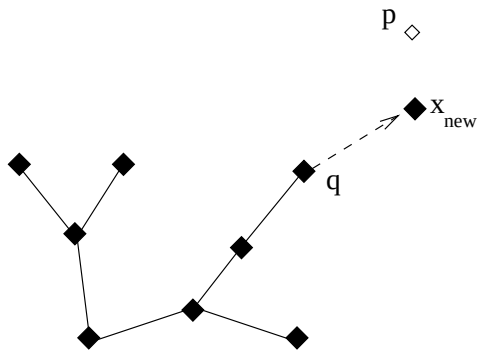



Fig. 3. An RRT extension

This process of selecting a node to extend and executing a simulation repeats until a state within some tolerance of the goal, $x_{goal}$, is added to the tree. Since RRT is a randomized algorithm, there is no guarantee that the goal will be found, even when a feasible path exists. In addition, due to the nature of exploration and extension used to generate the search tree, the final path may be jagged and meandering. Modest efforts to smooth the generated path are usually applied in order to generate more natural motion. Both these problems, though, are often addressed by using random restarts.

### B. Particle Extensions

To the RRT framework, we introduce a particle–based extension technique, similar to the prediction step of a particle filter. In short, the pRRT algorithm operates by simulating each extension multiple times under various likely conditions. The resulting states are grouped into similar clusters, and each cluster is treated as a single node in the tree. Thus, nodes in the tree are comprised of distributions of states (approximated using particles), rather than individual states. The likelihood of reaching a particular node as the result of an extension can be calculated based on the likelihood of the conditions which generated the particles underlying its distribution. While generating the search tree, we can use the calculated likelihood of nodes to bias the search, and when the goal is reached, we can evaluate the likelihood of the entire generated path based on the nodes in the path.

During the EXTEND step, an action $u_{new}$ is chosen as usual to apply to the known–reachable state $q$. In the application domain described in this paper, the coefficient of friction for the terrain is not known with certainty, so we simulate the execution of the chosen action using several likely values. In general, the uncertain parameter, or parameters, may be represented as a single variable **F**, and we assume that a

probability density function (PDF) is available from which to sample **F**. In fact, **F** may have a different distribution in different parts of the state space. For simplicity of notation, we assume here that **F** is represented by a single PDF. The technique of repeated simulation can always be applied since it requires no additional knowledge about the kinematics of the robot beyond the ability to simulate an action, and this capability is already required by the basic RRT algorithm. Within the framework introduced by [5], this uncertainty is in the realm of environment sensing. Uncertainty in initial configuration sensing can also be trivially incorporated by the same method. To incorporate uncertainties in the remaining categories of configuration and environment predictability would require altering the forward simulator such that each step is stochastic rather than deterministic.

Two elements are required for an extension step: the starting state for the simulation (the node $q$ in figure 3), and the destination (the point $p$ in figure 3) or, equivalently, the nominal action $u$ used to extend $q$ toward $p$. When extending from a particle node — a node that includes multiple particles — we have two primary options for the starting state. We might simply use the weighted mean of the state variables of each particle to represent the node, and execute all extensions from this state. This mean state, $\overline{q}$, is an estimate of the mean of the true PDF at particle node, $q$. That is, it is an estimate of the mean if this particle node included an infinite number of particles. This is the best estimate of the true state of the robot at this node of the tree under the uncertainty considered by the algorithm. The weight of each particle used for computing $\overline{q}$ is the probability that the condition used to compute that extension is the true condition. Let us denote the particles contained in the node $q$ as $q_1, q_2, \cdots, q_n$. Each particle $q_i$ has an associated value of the parameter **F**, which we denote $F_{q_i}$. $F^*$ will represent the true value of **F**. Using this notation, we can calculate the weighted mean state of a particle node by:

$$\begin{aligned} F_s &= \sum_{q_i \in q} P(F_{q_i} = F^*) \\ \overline{q} &= \frac{1}{F_s} \sum_{q_i \in q} (P(F_{q_i} = F^*) q_i) \end{aligned} \tag{1}$$

Since several extensions will be attempted, we also have the option of sampling the starting state for an extension from the PDF at the node $q$. If we were to approximate the true, continuous PDF by fitting a function such as a Gaussian to the particles at $q$, we could sample the starting state from this distribution. However, we expect to have few particles at each node. In fact, some nodes may only have a single particle, so the statistical basis for fitting a continuous function is somewhat precarious. Instead, we may choose to sample proportionally from the discrete set of particles. This has the added benefit of ensuring that the state from which we extend is truly feasible, since it has been calculated as the result of a simulation rather than interpolation between true particles.

The choice of strategy for choosing a starting state, either by calculating the mean state or by discrete sampling, is explored in section III. Once this choice is made, the action used to extend toward $p$ can be calculated as usual. In the EXTEND step of the algorithm, we apply this action to the starting state or states using forward simulation multiple times in order to produce new particles. For each simulation, we apply a value of **F** sampled from its PDF. The states that result from successful simulations are clustered as described below before they are added to the tree.

### C. Clustering

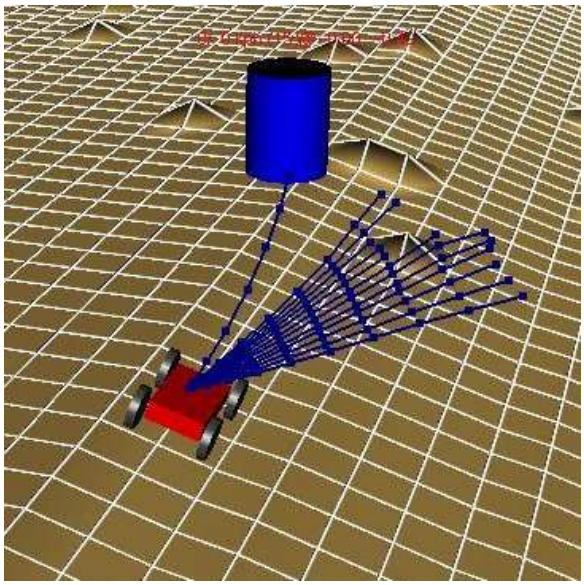
The underlying purpose of clustering is to group particles that are substantially similar. Although particles within clusters will differ somewhat, we wish to detect qualitative bifurcations in the tree caused by the changing values of **F**. Otherwise, the mean value $\overline{q}$, used to extend the node, becomes a poor approximation of the distribution of particles within that node. The example trajectories in figure 4 illustrate significant bifurcations in the path of the rover. In this example, the rover is moving along a ridge line toward the goal represented by a cylinder. If the coefficient of friction is sufficiently large, the rover can ascend the slope and reach the goal. For progressively smaller coefficients of friction, the rover slips down the hill as it moves. Depending on how much it slips, it may impact, or pass to the left or right of the rock ahead of it. These semantically different cases should be represented as different nodes in the planning tree.

Clustering is accomplished using a hierarchical clustering tree [6] with a weighted Euclidean distance metric. Since we are interested in the position $(x, y)$ and yaw $(\theta)$ of the robot, we have chosen to use the following distance metric:

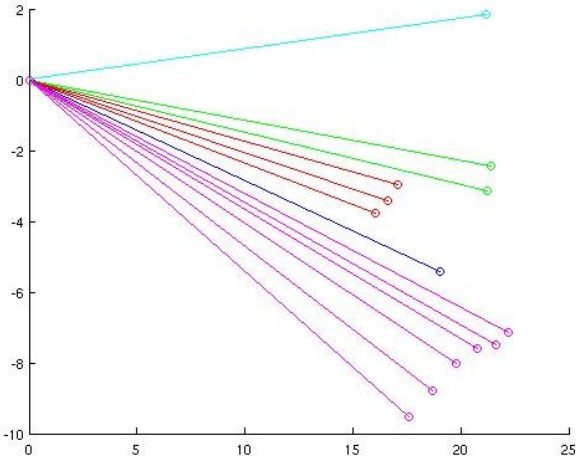
$$d = \sqrt{\alpha * (\delta x^2 + \delta y^2) + \beta * \delta\theta^2}$$

where $\alpha$ and $\beta$ are scaling factors used to normalize the units of measurement when determining the difference between particles. The hierarchical clustering tree algorithm uses this metric to iteratively agglomerate the particles and clusters separated by the shortest distance. The distance between particles is straightforward to calculate, but the distance between clusters may be calculated by a variety of strategies. The results section below will examine the difference in performance between two strategies known as single and complete linkage [7]. Single linkage computes the distance between two clusters as the minimum of all pairwise distances between particles in the clusters, while complete linkage uses the maximum.

The dendrogram in figure 5 illustrates the operation of the algorithm on the particles from figure 4. Particles are numbered along the horizontal axis, and distances are represented on the vertical. The iterative algorithm builds the dendrogram from the bottom up. The closest particles in this case are 8 and 9, so these are combined first, as represented by the horizontal line segment near the lower left corner of the diagram. The next closest particles are joined together until all particles have been combined into a single cluster.



(a) Simulation




(b) Plot

Fig. 4. Trajectories with qualitatively different endpoints

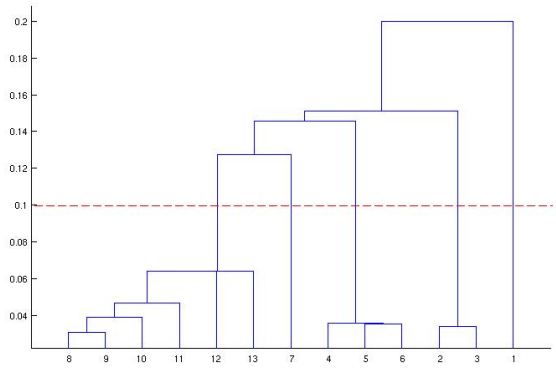



Fig. 5. Dendrogram produced by the hierarchical clustering tree algorithm

In this case, the final agglomeration combines particle 1 with a cluster containing all other particles.

Although agglomeration continues until all particles are linked, we must determine how much aggregation is actually appropriate for each set of particles. We locate the link in the tree which combines the most dissimilar subtrees by searching for the largest difference in distances between successive agglomerations. This link and all following links are disregarded, while any links made previously are used to determine the particle clustering. This cutoff is represented by the dashed horizontal line in figure 5. In this case, the particles have been split into five clusters.

In tests, we found that this approach performs better than the Gaussian clustering of the k–means algorithm [8]. The hierarchical clustering tree allows us to calculate fewer parameters since we do not need to directly estimate the means and covariance matrices of each cluster. This is especially important since we are clustering only a handful of points.

### *D. Node and Path Probability*

The particles at each node provide an estimate of the distribution of values of **F** that allow the robot to reach that state. By combining this with a prior distribution over **F**, the probability of reaching any particular node, or indeed the probability of following an entire path, may be calculated. In this way, the growth of the tree structure can be biased toward generating paths that are more likely to be followed by the robot.

To bias the search, we adapt Urmson's hRRT technique for heuristically biasing RRT growth [9]. The heuristic modifies the SELECT_EXTENSION function of the RRT algorithm. Rather than simply accepting a random point, $p$, in the state space and its nearest neighbor, $q$, in the tree, the hRRT technique chooses to extend proportional to the quality of the node $q$. In our case, the quality of $q$ is defined as:

$$q_{quality} = \frac{q_{prob} - m}{1.0 - m} \tag{2}$$

where $q_{prob}$ is the probability of reaching $q$ from the root of the tree, and $m$ is the minimum probability of all leaf nodes of the search tree. A random value, $r$ is drawn from a uniform distribution between 0 and 1. If $q_{quality} > r$, the pair of points $p$ and $q$ are accepted and an extension is attempted. Otherwise, a new pair of points are chosen. Alternately, we might keep $p$ and try the next nearest neighbor in the tree [9]. The random value $r$ is used to promote the use of extensions from high quality nodes without excluding the possibility of extending lower quality nodes should the algorithm become stuck in a situation where reaching the goal at all is unlikely.

The effectiveness of the quality heuristic might be improved if we could calculate not only the probability of reaching a node from the root of the tree, but also an estimate

of the probability of reaching the goal from this node. This would produce a heuristic in the style of A* [10] that would estimate the probability of successfully travelling from start to goal through any particular node. However, any estimate of the probability–to–go must be optimistic, meaning it cannot underestimate the possibility of completing the path from any state. Without exploring all options from this state, the only admissible heuristic is 1.0, which does not provide any additional information. Future work may investigate whether non–admissible heuristics improve the runtime of the algorithm without significantly affecting path quality.

In practice, 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. In order to encourage more extensions from nodes far from the root, the path probability $q_{prob}$ is normalized using the path length. We substitute $\sqrt[n]{q_{prob}}$ in the quality calculation, where $n$ is the depth of node $q$ in the tree. This improves the runtime since node quality does not drop so quickly as the distance from the root increases. However, the effect of averaging the path probability over the length of the path may allow the effect of a single unlikely node to go unnoticed in a long path of otherwise likely nodes. Test results in the next section illustrate the effects of this tradeoff.

## III. Testing and Results

The pRRT algorithm has been implemented and tested in simulation using several rover navigation scenarios. Testing began with fairly simple domains like the slope in figure 6. Situations like these allow us to verify that the decisions made by the pRRT algorithm reflect a concern for consistent execution regardless of friction. The robust path represented in this image by the string of spheres advances along flat ground, makes a wide turn, then climbs the slope head–on. Since the rover is capable of climbing over rocks, a shorter path is possible by crossing the slope at an angle. However, taking advantage of these capabilities would result in a less robust path.

Additional tests were conducted in more complicated environments, such as the terrains shown as contour plots in figure 7. The rover's starting position is marked with a star, and the goal is marked with a circle. These terrains were chosen as representative of situations that real rovers are expected to encounter, but they are also meant to illustrate that the most direct path is not always the best. In both the plateau and the crater scenarios, the direct path to the goal requires the rover to negotiate elevation changes that the physical simulations will show are safe for the rover to traverse. Unfortunately, given that the real terrain characteristics cannot be determined with certainty by a real rover, it would be unlikely that the rover would track those paths successfully. Thus, the longer paths on flatter terrain are to be preferred.

The purpose of these tests is not merely to show that the pRRT planner is capable of avoiding what would otherwise

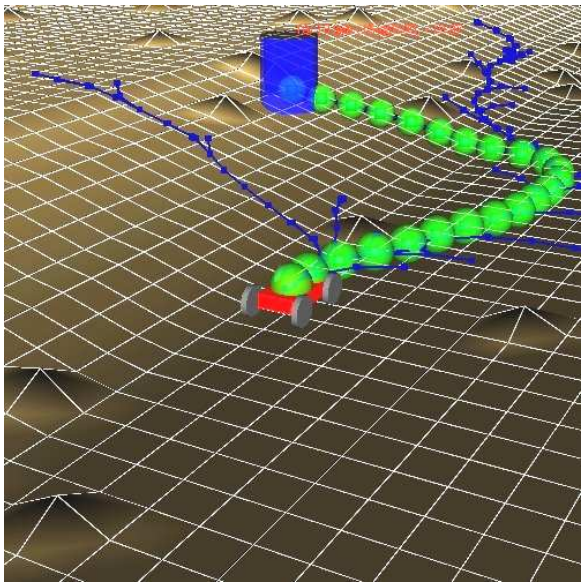


Fig. 6. A tree built by pRRT. Spheres mark the planned path.

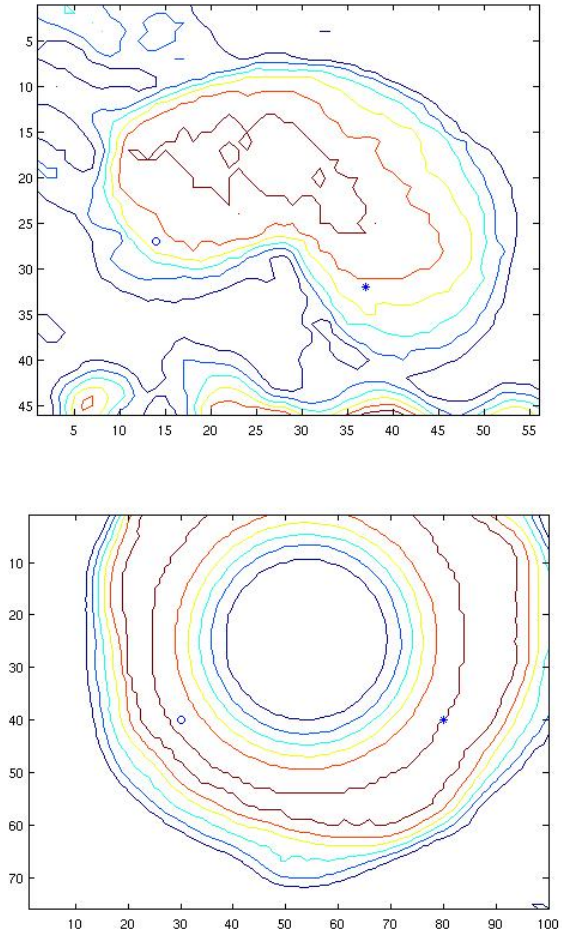
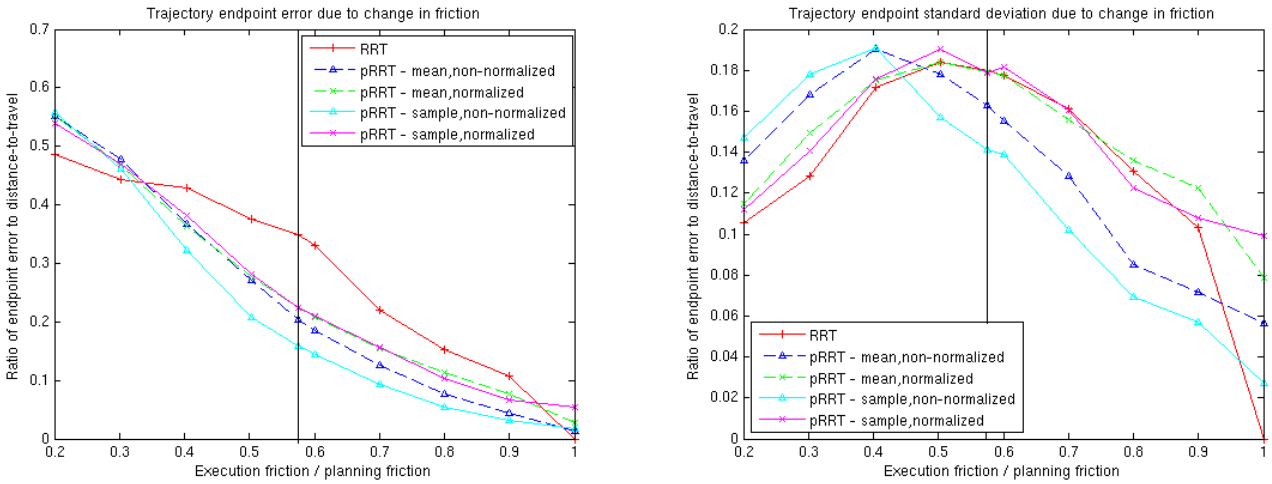



Fig. 7. Contour plots of some terrains used for testing: Plateau (top), Crater (bottom)




(a) Endpoint error

(b) Endpoint standard deviation

Fig. 8. Error and standard deviation plots for extension strategies

| Start node | Probability heuristic | Success | Nodes | Nodes per extension | Path probability | Normalized path probability |
|---|---|---|---|---|---|---|
| Mean | non–normalized | 80.6% | 123.93 | 1.40 | 59.6% | 96.8% |
| Mean | normalized | 81.8% | 97.92 | 1.37 | 51.3% | 94.8% |
| Sample | non–normalized | 72.2% | 233.82 | 2.00 | 25.7% | 91.2% |
| Sample | normalized | 81.0% | 173.57 | 1.95 | 17.9% | 82.3% |
| — | RRT | 82.2% | 80.92 | 1 | — | — |

TABLE I

EFFECTS OF PARTICLE NODE REPRESENTATIONS

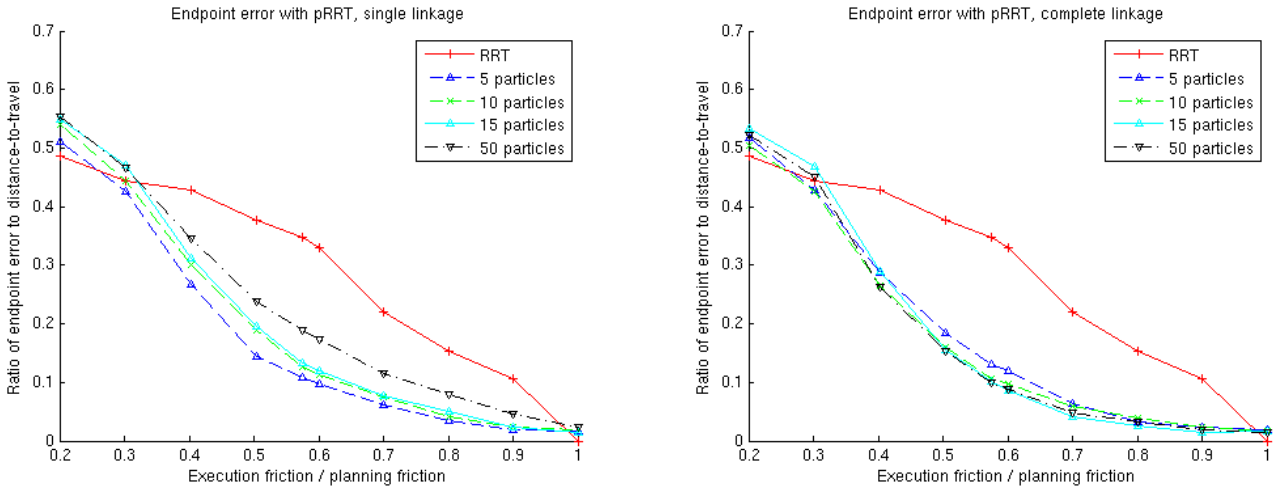
be considered high–cost terrain. However, simulations are likely to produce different results in these areas for different coefficients of friction. Any area in which a small expected range of values of $\mathbf{F}$ can produce great variability in simulation results will produce unlikely paths, and will thus be avoided by pRRT.

To quantify the robustness of paths using pRRT and regular RRT, paths were planned in each scenario, then executed open–loop under a variety of uniform friction conditions. Conventional RRT extensions such as the connect heuristic [11] were also implemented in both cases to improve planned paths and decrease planning times. Trees built using conventional RRT assumed a single value of friction at all times, while pRRT considered a uniform distribution over a range of possible values. Several variations of the pRRT algorithm were used in order to determine the best choices for the algorithmic and parameter options presented in the previous section.

### A. Extension Strategy

For the first set of tests, four different variations of the pRRT algorithm were used. As discussed in section II-B, we must choose the strategy for determining the initial state for each extension. We tested the algorithm using the mean state $\overline{q}$ of a node, and by sampling from the particles at each node. We also explored the effect of normalizing the probability of successfully tracking paths. This option was discussed in section II-D as a method of reducing the bias toward shorter paths. All combinations of these choices were tested.

Error measurements were taken to compare the endpoints of the planned paths to the endpoints arrived at by open–loop execution of the plans when the actual coefficient of friction was less than that planned by RRT. The plots in figure 8 show the average error (and standard deviation of error) averaged over 500 runs in the plateau scenario (top of figure 7). Results are similar for the crater scenario. The horizontal axis in these plots shows the uniform coefficient of friction for the open–loop execution as a fraction of the value used by the RRT. The pRRT algorithm considered all values of friction to the right of the vertical line in the plots while planning. The error shown on the vertical axis is expressed as a fraction of the distance between the specified start and goal states. This metric facilitates comparisons of the performance in various scenarios because the distance from start to goal will differ. These plots indicate that consideration of the uncertainty in friction by the pRRT algorithm results in paths that are significantly more accurate for all variations of the pRRT algorithm. Although regular RRT produces no error in a completely deterministic world (where true friction is always equal to the friction used for planning), pRRT produces less error than RRT when uncertainty is introduced. This means that paths planned by pRRT can be executed safely and consistently as the true value of $\mathbf{F}$ changes. In fact, pRRT continues to produce more accurate paths to the left of the vertical line, outside the range of friction values which it considered when planning.




(a) Single linkage

(b) Complete linkage

Fig. 9. Error plots for particle and clustering strategies

Error was reduced the most when sampling the starting state for each extension from among the particles at a node. In fact, figure 8(b) shows that the variation in endpoint error was also significantly reduced in these cases. This approach is more robust because the effect of uncertainty is more faithfully considered at each extension of the tree. However, the cost of this increased accuracy and precision can be seen in the statistics presented in table I. This table presents some of the differences caused by the variations which have been discussed for pRRT. The algorithm is halted if the goal is not reached by the time the search tree contains 250 nodes, so the success column lists the percentage of the 500 attempted runs in which the goal was found. Without the normalization of the path probability heuristic, the natural bias toward short paths prevents any branch of the search tree from growing long enough to reach the goal. This effect is most pronounced when the start node is sampled because this approach also leads to more nodes per extension. Since each start node is less precise, the particles of a single extension will be spread over a larger area, thus leading to more clusters. Each of these clusters will have a lower probability, so without the effect of normalization, probabilities of long paths will suffer. The final two columns list the probability of tracking a complete path from the root to the goal.

The version of pRRT described by the second row in the table performs most similarly to RRT because the effects of uncertainty are minimized by using the mean state $\overline{q}$ for starting extensions and normalizing the path probability. The third row corresponds to the variation which produced the lowest error and standard deviation in the plots.

### B. Clustering Strategy

Another important parameter for this algorithm is the number of particles created with each extension. Since the simulation of the rover's actions is the most computationally expensive step in this algorithm, the running time of pRRT is nearly equal to the running time of RRT multiplied by the number of particles per extension. Choices for this parameter were explored in conjunction with the choice of linkage strategy for clustering particles. Error plots in figure 9 summarize the results of these tests. Surprisingly, single linkage produced worse results as the number of particles increased. We believe that this is due to the lack of a natural limit on the size of a cluster. To illustrate this problem, consider a set of nearly–equally spaced points along a line. A complete linkage strategy will subdivide the points into clusters of nearly equal size. The divisions produced by single linkage will appear to be arbitrary, so the probability of particles arranged in this manner will be split arbitrarily between clusters. Figure 9(b) is encouraging since it shows that the best performance with complete linkage may be reached with very few particles.

### C. Execution Monitoring

The discussion so far has evaluated the robustness of open–loop execution of planned paths. However, if the rover has some method reliably estimating its location during execution, it may be able to improve the tracking of a planned path by adjusting the executed actions based on its actual location and the location of the next node in its planned path. Again, we discuss two strategies for performing this update. The first is a well–known algorithm called pure pursuit [12]. In this strategy, we calculate the new action using the same simple kinematic equations that are used by RRT to estimate the action for an extension. These equations assume flat terrain and ignore dynamics and the effects of terrain interaction. The second strategy, which we call the proportional approach, attempts to use information about the terrain that was captured in the building of the planning tree. This strategy makes use of the fact that, due to slippage, the steering angle used by the vehicle to advance from one node to another is not necessarily equal to the angle between the

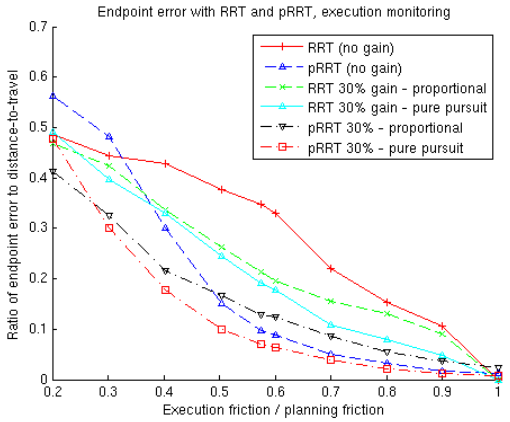



Fig. 10. Error of execution monitoring strategies

nodes. The ratio of these two angles is used to compute a new steering angle.

The results of these tests are shown in figure 10. In both cases, the new action that was computed did not completely replace the planned action. Instead, a linear combination of the two actions was used with a gain parameter to determine the proportion assigned to the new action. For both strategies, the best performance was achieved with a gain near 30%, so this is the gain shown in the plots. Although the simpler pure pursuit method produced lower error than the proportional approach, we note that pRRT was able to produce better results, even without execution monitoring, than the original RRT with execution monitoring.

## IV. Conclusions and Future Work

The Particle RRT algorithm is an effective method for generating robust paths that are more precise and more accurate in the face of uncertainty in the environment. Paths planned using pRRT are inherently safer, since they can be followed more closely despite variations in the uncertain characteristics of the environment.

Now that Particle RRT has shown itself to be an effective extension to randomized path planning, we would like to combine it with other extensions such as cost–based tree building heuristics. The probability of successfully following a path and the expected cost of that path are both important criteria in guiding the selection of nodes to extend in a tree and in assessing potential paths from root to leaves in that tree.

Although RRTs are often considered to be single–query path planners, we expect to be able to reuse portions of an existing tree for replanning as additional information becomes available to the robot. As a planned path is executed, the uncertainty about the environment decreases, while portions of the planning tree remain relevant. If past simulations are cached, they can be used to build new RRTs for additional queries or refined if the same query is issued again. We will investigate methods for reactive, real–time replanning [13] when exploration of the terrain causes the PDF of $\mathbf{F}$ to be reshaped, thus changing the probabilities of nodes in the planned tree.

Immediate plans for continuing investigation of pRRT include the move from simulation to execution on a real rover. Implementation is planned for the iRobot ATRV–Jr robotic platform using the CLARAty programming framework [14]. This testbed will demonstrate the efficacy of pRRT in the face of the true uncertainty of a terrain model built using stereo vision. It will also help prepare for planned deployment on a future NASA rover mission on Mars.

The advancements in this algorithm will improve the efficiency of future rovers in especially rough terrain compared to the Mars Exploration Rovers which are currently in service. Some of the most interesting science targets for those rovers are rock outcroppings located in areas of difficult terrain such as craters and steep hillsides. Currently, the rovers must be teleoperated in these areas, which requires a delay of at least one day. The rovers sit idle while a team of humans analyzes stereo vision data and constructs a precise set of driving commands to upload to the rover. Even with increased computational demands on the modest hardware of a rover, the pRRT algorithm should help reduce the amount of time that the rovers spend waiting for their next driving commands. The algorithm will also be useful as a ground–based tool for verifying plans which are to be uploaded to the rovers.

## References


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

[2] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *International Journal of Robotics Research*, vol. 21, no. 3, pp. 233–255, 2000.

[3] A. Hait and T. Simeon, "Motion planning on rought terrain for an articulated vehicle in presence of uncertainties," *IEEE/RSJ International Symposium on Intelligent Robots and Systems*, pp. 1126–1133, 1996.

[4] 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.

[5] S. M. Lavalle and R. Sharma, "On motion planning in changing, partially–predictable environments," *International Journal of Robotics Research*, 1997.

[6] 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.

[7] P. H. A. Sneath and R. R. Sokal, *Numerical Taxonomy*. London, UK: Freeman, 1973.

[8] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*. John Wiley & Sons, 1973.

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

[10] N. J. Nilsson, *Principles of artificial intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1980.

[11] S. M. Lavalle and J. J. Kuffner, "RRT–Connect: An efficient approach to single–query path planning," in *Proceedings of IEEE International Conference on Robotics and Automation*, April 2000, pp. 995–1001.

[12] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-92-01, January 1992.

[13] J. Bruce and M. Veloso, "Real–time randomized path planning for robot navigation," in *Proceedings of IROS–2002*, Switzerland, October 2002.

[14] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das, "The claraty architecture for robotic autonomy," in *Proceedings of IEEE Aerospace Conference*, March 2001.