Proceedings of the ASME 2015 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference
IDETC/CIE 2015
August 2-5, 2015, Boston, Massachusetts, USA

DETC2015-46535

FOCUSED REFINEMENT IN THE RRT*: TRADING OPTIMALITY FOR IMPROVED PERFORMANCE

Beth Boardman *
Sonia Martínez
Mechanical and Aerospace Engineering
University of California San Diego
9500 Gilman Dr, La Jolla, California 92093-0411
Email: {bboardman, soniamd}@ucsd.edu

Troy Harden

Los Alamos National Laboratory
PO Box 1663, MS J580, Los Alamos, New Mexico 87545
Email: harden@lanl.gov

ABSTRACT

This paper investigates how to limit the exploration property of the RRT algorithm in order to decrease the computation time needed to produce a low-cost, but good enough, path. We aim to do this by (i) focusing the attention of the RRT* algorithm on paths that are found quickly by RRT*, and by (ii) reducing the number of nodes in the obtained paths. The latter is achieved by an online smoothing process that aims to connect added nodes directly to their grandparents. Extensive two-dimensional simulation results are provided to examine how the number of obstacles in an environment affects the proposed extensions. Simulations for a Dubins' vehicle are presented to show how the modifications perform for vehicles with differential constraints.*

INTRODUCTION

Motion planning for high-degree-of-freedom systems in many obstacle environments is a difficult problem that has stimulated a main research effort in the robotics field. By relaxing the notion of completeness to resolution or probabilistic completeness, sampling-based motion planners can return a feasible path to a goal configuration in a relatively fast manner. The obtained path can even satisfy asymptotically optimal performance at the cost of a higher run-time. However, obtaining the optimal path may be detrimental in real-time implementations, for which the computation time can outweigh the benefit of the path optimality. Motivated by this issue, we investigate and analyze exploration-exploitation tradeoffs in the Rapidly Random Tree Star exploration algorithm.

Rapidly-exploring Dense Tree algorithms (RDTs, also known as RRTs) [1] and Sampling-Based Roadmaps (SBRs, including Probabilistic Roadmaps (PRMs) [2]) are sampling-based motion planners which are resolution or probabilistically complete and are able to find a feasible path to the goal in the presence of narrow corridors. As opposed to SBRs, RDTs do not require pre-processing and can find a path relatively quickly. However, the path produced by these planners can be very jagged and result in unnecessary motion that can increase the execution time. Consequently, this motivated research into how to obtain better paths from these planners.

One way to obtain improved paths is to apply a post-processing algorithm. In [3], a post-processing algorithm for path smoothing is presented for any given path. The algorithm limits the allowable deviation from the original path and results in a path with fewer nodes. A divide and conquer method is used in [4] in order to shorten any given path by connecting the first and last nodes in the path directly. If not successful, then the set of nodes in the path list is bisected until the connections are successful. Similarly, the post-processing algorithm in [5] randomly selects two points from the path list and attempts to replace the segment between them with a straight line. This process is repeated a predetermined number of times.

A subsequent effort focuses on obtaining paths that guaran-

*Address all correspondence to this author.

Copyright © 2015 by ASME

tee asymptotic optimality with probability one. In this line of work, we can find PRM* and RRT* [6]. The RRT* paths can be composed of many more nodes than are strictly necessary. The RRT* can handle any-time applications [7] and manipulators [8]. The Ball-tree algorithm, presented in [9], is a sampling-based motion planner that improves the performance of the RRT and RRT* by using volumes of free-space instead of points in the free-space as the vertices of the tree.

The following papers also study the effects of exploitation versus exploration on the RRT*. Akgun et al. [10] uses local biasing to choose the sampling point based upon the current best path to the goal. A node, q, is selected from the path and then the sampled configuration point is, in comparison to q, chosen to be closer to the path between q's parent node and child node. The RRT*-Smart in [11] finds an initial path to the goal, then it optimizes it using first a smoothing technique, and then it further shapes it by biasing sampling to balls around the nodes in the optimized path. While these two papers share the same idea of exploitation of a given path to the goal, this paper presents an alternative approach to choosing the biased sampling points and smooths the entire tree, not just a single path. This more global methodology can lead to a further cost reduction, and this paper evaluates the computational tradeoffs.

This paper introduces the Focused-Refinement algorithm, a modification of RRT*, to reduce the computation time needed to obtain a low-cost path to the goal. This is done by exploring the environment quickly until a set of paths to the goal is found. Then, the algorithm focuses on lowering the cost of this set of paths while periodically exploring the environment. In this way, the algorithm returns an asymptotically optimal path within the regions that are more intensively exploited. We present a novel way of uniformly sampling randomly within these regions that, with the right parameters, can recover the entire configuration space. We combine this idea with a so-called grandparent connection modification to reduce the number of nodes in the path, and thus, lower the cost. This grandparent connection strategy acts similar to the path smoothing algorithms discussed above except the path is smoothed directly in the motion planning algorithm after each node is added. Also, unlike the earlier smoothing techniques, our grandparent connection smooths out the entire tree, not just a single path which results into a lower cost.

The paper is organized as follows. First, the RRT* algorithm is reviewed. Then, the details of the proposed algorithm modifications are presented. These are then analyzed through extensive simulation. Finally, conclusions and ideas for future work are presented.

RAPIDLY-EXPLORING RANDOM TREE STAR

This section briefly describes the RRT* algorithm by Karaman and Frazzoli which is theoretically analyzed in [6]. The kinodynamic RRT* is presented in [12].

The RRT*, outlined in Algorithm 1, builds a tree, $\mathscr{T}$ which is dense with probability one in the entire configuration space, X, as the number of samples, n, goes to infinity. Denote by X_{free} the free configuration space in X and X_{obs} as the obstacles space. The tree is composed of a set of vertices, $v \in \mathscr{T}.V$, and edges, $e \in \mathscr{T}.E$. Each edge is an ordered pair of vertices $e_{1,2} = (v_1, v_2)$, where v_1 is the parent and v_2 is the child. We use Cost as the notation for the cost function being minimized. Each edge added to $\mathscr{T}$ has a cost associated with it, denoted $c_{\text{edge}}(e)$, where $e \in \mathscr{T}.E$. In the original work by [6], the edge cost considered is the *cost-to-go*; that is the cost of $e_{1,2} = (v_1, v_2)$ is the cost of moving from the parent v_1 to the child v_2. Then, the cost of a vertex, $\mathsf{Cost}(v)$, is the sum of the costs of the edges connecting the root to v. The paths in $\mathscr{T}$ are then asymptotically optimal, meaning that as $n \to \infty$ the optimal path from the initial configuration, $x_I \in X_{\text{free}}$, to any other configuration in X_{free} is recovered. More precisely, the functions involved in the RRT* process are described as follows. With some abuse of notation, we will use the configuration, x_v, instead of v.

After initializing $\mathscr{T}$ at x_I, the RRT* begins by using the Sample function to output x_{rand}, a uniformly sampled random configuration from X_{free}. The Nearest function finds the nearest vertex, $x_{\text{nearest}} \in \mathscr{T}$, and extends $\mathscr{T}$ toward x_{rand} a distance ε from x_{nearest} to get x_{new}.

Next, the set of near vertices from $\mathscr{T}$ with respect to x_{new} are output as the set X_{near} from the function Near. Vertices that are farther than $r = \min(\varepsilon, \gamma(\log(n_v)/n_v)^{(1/d)})$, where n_v is the number of vertices in $\mathscr{T}$, d is the dimension of the configuration space, and γ is an independent parameter, are omitted from X_{near}. The best parent for x_{new}, determined in FindBestParent, is the vertex in X_{near} that has a collision free path with the lowest $\mathsf{Cost}(x_{\text{new}})$, as outlined in Algorithm 2. The paths that connect the vertices to each other (determined using Steer), do so according to the system dynamics. Only collision free edges are added to $\mathscr{T}$. The collision checker, CollisionFree, returns true if the edge is collision-free. If x_{new} is added to $\mathscr{T}$, then Rewire attempts to add the other vertices in X_{near} as children of x_{new} based upon a lower cost and collision-free edge. The Rewire function is outlined in Algorithm 3.

THE FOCUSED-REFINEMENT AND GRANDPARENT CONNECTION MODIFICATIONS

In this section, two extensions to the RRT* are described in detail. The first is the Focused-Refinement (FR) modification that attempts to recover a near optimal path much quicker than RRT*. The second extension, the grandparent connection, is aimed at reducing the number of nodes in the path to the goal and reducing the computation time needed to discover the optimal path.

Copyright © 2015 by ASME

Algorithm 1 $\mathscr{T} = (V,E) \leftarrow \mathsf{RRT}^*(x_I, \varepsilon)$

$\mathscr{T} \leftarrow \mathsf{InitializeTree}()$;
$\mathscr{T} \leftarrow \mathsf{InsertNode}(\emptyset, x_I, \mathscr{T})$;
for $i = 1$ to $i = N$ **do**
  $x_{\text{rand}} \leftarrow \mathsf{Sample}(i)$;
  $x_{\text{new}} \leftarrow \mathsf{Nearest}(\mathscr{T}, x_{\text{rand}}, \varepsilon)$;
  $X_{\text{near}} \leftarrow \mathsf{Near}(\mathscr{T}, x_{\text{new}})$;
  $x_{\text{parent}} \leftarrow \mathsf{FindBestParent}(X_{\text{near}}, x_{\text{new}})$;
  if $x_{\text{parent}} \neq \mathsf{NULL}$ **then**
    $\mathscr{T} \leftarrow \mathsf{InsertNode}((x_{\text{parent}}, x_{\text{new}}), x_{\text{new}}, \mathscr{T})$;
    $\mathscr{T} \leftarrow \mathsf{Rewire}(\mathscr{T}, X_{\text{near}}, x_{\text{new}})$;
  end if
end for
return $\mathscr{T}$

Algorithm 2 $x_{\text{parent}} \leftarrow \mathsf{FindBestParent}(X_{\text{near}}, x_{\text{new}})$

$x_{\text{parent}} \leftarrow \emptyset$;
$c_{\min} \leftarrow \infty$;
for $x_{\text{near}} \in X_{\text{near}}$ **do**
  $e_{\text{near,new}} \leftarrow \mathsf{Steer}(x_{\text{near}}, x_{\text{new}})$;
  $c_{\text{near}} \leftarrow \mathsf{Cost}(x_{\text{near}}) + c_{\text{edge}}(e_{\text{near,new}})$;
  if $c_{\text{near}} < c_{\min}$ and $\mathsf{CollisionFree}(e_{\text{near,new}})$ **then**
    $x_{\text{parent}} \leftarrow x_{\text{near}}$;
    $c_{\min} \leftarrow c_{\text{near}}$;
  end if
end for
return x_{parent};

Algorithm 3 $\mathscr{T} \leftarrow \mathsf{Rewire}(\mathscr{T}, X_{\text{near}}, x_{\text{new}})$

for $(x_{\text{near}}) \in X_{\text{near}}$ **do**
  $e_{\text{near,new}} = \mathsf{Steer}(x_{\text{new}}, x_{\text{near}})$;
  if $\mathsf{Cost}(x_{\text{new}}) + c_{\text{edge}}(e_{\text{near,new}}) < \mathsf{Cost}(x_{\text{near}})$ **then**
    if $\mathsf{CollisionFree}(e_{\text{near,new}})$ **then**
      $x_{\text{oldparent}} \leftarrow \mathsf{Parent}(\mathscr{T}, x_{\text{near}})$;
      $\mathscr{T}.\mathsf{remove}((x_{\text{oldparent}}, x_{\text{near}}))$;
      $\mathscr{T}.\mathsf{add}((x_{\text{new}}, x_{\text{near}}))$;
    end if
  end if
end for
return $\mathscr{T}$;

The Focused-Refinement algorithm

As shown in [6], the RRT* initially constructs a tree that is the same as the the RRT and then, as more nodes are added, the RRT* begins to look at many neighboring vertices to recover an asymptotically optimal path. The RRT* finds and refines all paths in the configuration space. The refinement extension, Focused-Refinement (FR), focuses on refining only those paths that have already reached the goal region after some time in hopes of reducing the amount of time needed to find a sufficiently optimal path.

The FR begins the construction of a tree using the RRT* algorithm until there exists at least one path that reaches the goal region. This set of paths is denoted as Π, with p vertices defining a set V_Π. The FR has two options: exploring the configuration space or exploiting Π by lowering its cost. If exploring, then the algorithm proceeds as the RRT* but if exploiting then the set of vertices in Π, V_Π, is determined. The sample x_{new} is determined by perturbing a vertex randomly drawn from the set V_Π. The FR then proceeds as the RRT*.

The pseudo code for the FR is presented in Algorithm 4, and uses three parameters. The first is $C_{\text{exploit}} \in \mathbb{N}$, which is the number of consecutive iterations the FR will exploit Π. The number of consecutive iterations to explore X_{free} is the second parameter needed, $C_{\text{explore}} \in \mathbb{N}$. The third parameter, $C_{\text{reset}} \in \mathbb{N}$, tells the algorithm when to update V_Π. The sampling region defined by V_Π does not change dramatically every iteration, therefore, to save computation time, the set V_Π is only updated every $C_{\text{reset}} + C_{\text{explore}}$ iterations. If $C_{\text{exploit}} = 0$ and $C_{\text{explore}} = \infty$, the FR becomes the RRT*. In order to take advantage of the exploitation property of the FR, C_{exploit} should be greater than C_{explore}. In environments with multiple routes to the goal, C_{explore} can be increased in hopes of finding a better or more direct route to the goal than what is found first.

Exploitation only occurs if GoalReach returns true (there exists at least one path to X_{G}) and exploitation has occurred less than C_{exploit} times consecutively. Once Π has been exploited C_{exploit} iterations, the RRT* is allowed to explore the space as normal for C_{explore} iterations. The following are the details of how x_{new} is chosen when in the exploitation stage of the FR.

The new sample, x_{new}, is determined as illustrated in Fig. 1 and in Algorithm 5. Given a d-dimensional configuration space, $X \subset \mathbb{R}^d$, consider $k \in \{1, 2, \cdots, d\}$. First, the minimum and maximum k-component from $V_\Pi \in \mathbb{R}^{d \times p}$, $w_{\min} = \min V_\Pi^k$ and $w_{\max} = \max V_\Pi^k$, are found. Here, V_Π^k is the set of all k-components of the vertices in V_Π. Next, the k-component of x_{new} (x_{new}^k) is taken as a uniformly random sample between $w_{\min} - \varepsilon$ and $w_{\max} + \varepsilon$. For every $j \neq k$, the j-component of the vertex whose k-component is nearest to x_{new}^k is determined, x_{nearest}^j, where

$$x_{\text{nearest}} = \underset{x \in V_\Pi}{\operatorname{argmin}} \; \|x_{\text{new}}^k - x^k\|.$$

Given $\varepsilon > 0$, the j-component of x_{new} is uniformly sampled between $x_{\text{nearest}}^j - \varepsilon$ and $x_{\text{nearest}}^j + \varepsilon$. The FR alternates through which k-component is used to determine x_{new}, this provides a uniform distribution of samples around Π. As ε is increased, the entire configuration space is uniformly sampled randomly, thus recovering the original RRT*.

Copyright © 2015 by ASME

Note that V_Π can consist of multiple distinct paths to the goal. Determining distinct paths is non-trivial and potentially time-consuming. In general, and in the simulation section, V_Π is taken to be only the current best path. Efficiently determining distinct paths is a subject of future work.

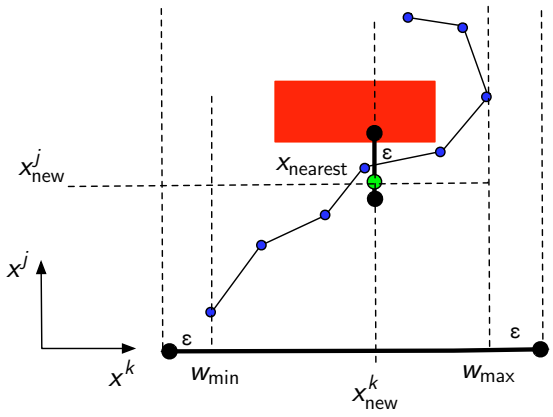


FIGURE 1: An illustrative example of how the FR algorithm chooses x_{new} when refining a single path. The red rectangle is an obstacle in the environment. The blue dots are the the set of vertices, V_Π, used to determine the region from which x_{new} is sampled. The k-component of x_{new} is a uniform random sample between the maximum and minimum (plus and minus ε, respectively) k-component values from V_Π. Next, with respect to x^k_{new}, determine the nearest k-component from V_Π and label its corresponding j-component as x^j_{nearest}. Finally, the x^j_{new} is a random value from between $x^j_{\text{nearest}}-\varepsilon$ and $x^j_{\text{nearest}}+\varepsilon$, $\varepsilon>0$. The sample x_{new} is represented as the green dot.

The Grandparent Connection modification

Reducing the amount of time needed to determine the controls to get from one node to the next is important for high-dimensional systems. One way of doing this is to reduce the number of nodes in a path; this idea is what inspires the grandparent connection, RRT*-gp. Before adding a node to the tree, the modified algorithm attempts to connect directly to its grandparent node, as outlined in Algorithm 6. A successful connection to the grandparent occurs when a lower cost, collision-free path is found. It is also predicted that the grandparent connection will produce smoother paths with fewer nodes. Because the grandparent connection is applied during construction of the tree as every node is added to the tree, the grandparent connection smooths out every path in the tree. There are two advantages to smoothing the entire tree as opposed to a single path. First, the grandparent connection can potentially find a better route than the first recovered.

Algorithm 4 $\mathscr{T}=(V,E) \leftarrow$ FR$(x_i,\varepsilon,d,C_{\text{exploit}},C_{\text{explore}},C_{\text{reset}})$

$\mathscr{T} \leftarrow$ InitializeTree();
$\mathscr{T} \leftarrow$ InsertNode$(\emptyset,x_i,\mathscr{T})$;
$c_{\text{reset}}=1$; $c_{\text{exploit}}=1$; $c_{\text{explore}}=1$; $k=1$;
for $i=1$ to $i=N$ **do**
  if GoalReach and $c_{\text{exploit}}<C_{\text{exploit}}$ **then**
    if $c_{\text{reset}}=1$; **then**
      $V_\Pi=$ PathSet$(\mathscr{T})$;
    end if
    $(c_{\text{reset}},c_{\text{exploit}}) \leftarrow$ UpdateParameters$(c_{\text{reset}},c_{\text{exploit}},C_{\text{reset}})$;
    $x_{\text{new}}=$ NewPointPathSet(V_Π,ε,k,d);
    $k \leftarrow$ UpdateDimension(d);
  else
    $(c_{\text{explore}},c_{\text{exploit}}) \leftarrow$ UpdateParameters$(c_{\text{explore}},c_{\text{exploit}},C_{\text{explore}})$;
    $x_{\text{rand}} \leftarrow$ Sample(i);
    $x_{\text{new}} \leftarrow$ Nearest$(\mathscr{T},x_{\text{rand}},\varepsilon)$;
  end if
  $X_{\text{near}} \leftarrow$ Near$(\mathscr{T},x_{\text{new}})$;
  $x_{\text{parent}} \leftarrow$ FindBestParent$(X_{\text{near}},x_{\text{new}})$;
  if $x_{\text{parent}} \neq$ NULL **then**
    $\mathscr{T} \leftarrow$ InsertNode$((x_{\text{parent}},x_{\text{new}}),x_{\text{new}},\mathscr{T})$;
    $\mathscr{T} \leftarrow$ Rewire$(\mathscr{T},X_{\text{near}},x_{\text{new}})$;
  end if
end for
return $\mathscr{T}$

Algorithm 5 $x_{\text{new}} \leftarrow$ NewPointPathSet(V,ε,k,d)

$w_{\min}=\min(V^k)$;
$w_{\max}=\max(V^k)$;
$x^k_{\text{new}}=$ Rand$(w_{\min}-\varepsilon,w_{\max}+\varepsilon)$;
$x_{\text{nearest}}=$ NearestComponent(V,x^k_{new});
for $j=1$ to $j=d$; **do**
  if $j\neq k$ **then**
    $v_{\min}=x^j_{\text{nearest}}-\varepsilon$;
    $v_{\max}=x^j_{\text{nearest}}+\varepsilon$;
    $x^j_{\text{new}}=$ Rand$(v_{\min},v_{\max})$;
  end if
end for

Second, if the tree is ever needed for replanning (i.e. when dealing with unexpected obstacles as in [13–18]), then only having a single smoothed path is a disadvantage compared to having a tree filled with smoothed paths. The grandparent connection can also be used in combination with the FR algorithm.

Copyright © 2015 by ASME

Algorithm 6 $x_{\text{parent}} \leftarrow \mathsf{FindBestParent}(X_{\text{near}}, x_{\text{new}})$

$x_{\text{parent}} \leftarrow \emptyset$;
$c_{\min} \leftarrow \infty$;
for $x_{\text{near}} \in X_{\text{near}}$ **do**
  $e_{\text{near,new}} \leftarrow \mathsf{Steer}(x_{\text{near}}, x_{\text{new}})$;
  $c_{\text{near}} \leftarrow \mathsf{Cost}(x_{\text{near}}) + c_{\text{edge}}(e_{\text{near,new}})$;
  if $c_{\text{near}} < c_{\min}$ and $\mathsf{CollisionFree}(e_{\text{near,new}})$ **then**
    $x_{\text{parent}} \leftarrow x_{\text{near}}$;
    $c_{\min} \leftarrow c_{\text{near}}$;
  end if
  if $x_{\text{parent}} \neq \emptyset$ **then**
    $x_{\text{grandparent}} \leftarrow \mathscr{T}.\mathsf{parent}(x_{\text{parent}})$;
    $e_{\text{grandparent,new}} \leftarrow \mathsf{Steer}(x_{\text{grandparent}}, x_{\text{new}})$;
    $c_{\text{grandparent}} \leftarrow \mathsf{Cost}(x_{\text{grandparent}}) + c_{\text{edge}}(e_{\text{grandparent,new}})$;
    if $c_{\text{grandparent}} < c_{\min}$ and $\mathsf{CollisionFree}(e_{\text{grandparent,new}})$ **then**
      $x_{\text{parent}} \leftarrow x_{\text{grandparent}}$;
      $c_{\min} \leftarrow c_{\text{grandparent}}$;
    end if
  end if
end for
return x_{parent};

SIMULATION RESULTS

Simulations of the proposed algorithms were implemented in MATLAB on a computer with a 2.66 GHz Intel Core i7 processor and 8 GB RAM running Mac OS X 10.8.5. The performance of the proposed algorithms is compared to the performance of the RRT*, RRT*-smart [11], and RRT with smoothing [4]. The RRT*-smart begins by building an RRT*, but once a path to the goal is determined, the path is smoothed and then the algorithm continues to refine the path using samples drawn from balls of radius epsilon centered at each node in the smoothed path. The path smoothing used in the simulations attempts to connect the first and last node in the path list, then if not successful the set of nodes in the path list is bisected until the connection is successful. The RRT is very similar to the RRT*, except that x_{new} is added to the tree as a child of x_{nearest} and there is no Rewire phase. There are two different vehicle types tested in simulation. The first is a vehicle without differential constraints with the graph edge cost being Euclidean distance. The second is a Dubins' vehicle with the graph edge cost being the travel time.

Two-Dimensional Euclidean Space

The results presented in this section are an average over 25 simulations for each algorithm. The same 25 sample sequences of 20 thousand configurations, drawn randomly from X_{free}, are used for each algorithm's results. The FR simulation results are run with $C_{\text{exploit}} = 21$, $C_{\text{explore}} = 1$, $C_{\text{reset}} = 10$, and $\varepsilon = 0.5$. For consistency, the RRT*-smart also used an $\varepsilon = 0.5$.

The algorithms are tested in five two-dimensional environments, each containing a different number of obstacles: 10, 25, 50, 75, and 100. A pentagon is used as the standard obstacle that is randomly inserted in the environment 10, 25, 50, 75, or 100 times. Because the placement is random, the obstacles are allowed to overlap.

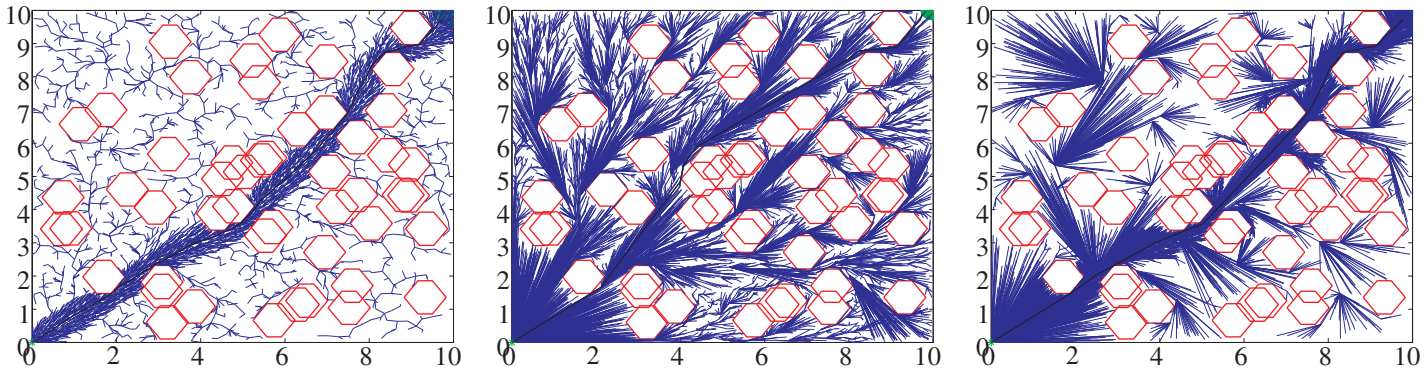
To begin, the graphs of the trees made by the RRT* and FR, with and without the grandparent connection, in the 50 obstacle environment are shown in Figs. 2a-2c. These represent a typical tree produced by the respective algorithms.

Average Cost For each of the five environments, Figs. 3a - 3e show the average minimum cost of the path to the goal plotted as a function of the run-time for each of the five algorithms, RRT*, RRT*-gp, FR, FR-gp, and RRT*-smart. The RRT* and FR find the same initial path, Π, at the same time. But, because the FR switches to sampling near Π, the cost for FR initially decreases much quicker compared to the RRT*. The RRT*-smart initially finds a low cost path in comparison to the RRT* and FR. But, because the RRT*-smart does not continue to explore, after an initial cost decrease, the cost flattens out. The RRT*-smart also has an increased run-time due to the extra collision checks performed during the path smoothing stage every time the cost decreases. The RRT*-gp and FR-gp both find low cost initial paths and continue to decrease the path cost, but at a much slower rate compared to the FR-gp. The FR-gp decreases the path cost the fastest of all five algorithms and by the end of the 20 thousand iterations has found the lowest cost path.

Table 1 compares the average cost of the first path found by the RRT* and RRT*-gp to the RRT with smoothing applied. For comparison the average cost of the best path found near the end of the algorithm run-time of the RRT* and RRT*-gp are also presented. The optimal cost is the single best path cost found by the simulations. Each path cost is compared to the optimal cost and the cost error is given. In all environments, the first path the RRT*-gp finds is lower than that of the RRT-smooth, usually at the expense of an increase in run-time. As the RRT*-gp is run longer, to 50 seconds in the 10 obstacle environment, the cost error decreases and in the more complex environments this decrease is significant. In the 100 obstacle environment the RRT-smooth has a path cost error of 11.33%, the RRT*-gp originally finds a path with a cost error of 10.73%, and then after 450 seconds the RRT*-gp finds a path with cost error of 2.53%. This drop in cost is attributed to the exploration property of the RRT*-gp which was able to find a lower cost path.

A comparison of the algorithms' average path cost at a given run-time, toward the end of the 20 thousand iterations, is in Table 2. The average costs are compared to the optimal cost and the error is given. In all environments the FR-gp produced the best paths. The reason the RRT*-smart does not perform as well is because it does not continue to explore the environment looking

Copyright © 2015 by ASME



(a) FR (b) RRT*-gp (c) FR-gp

FIGURE 2: These are typical trees produced in the 50 obstacle two-dimensional Euclidean environment. The obstacles are outlined in red and the trees are in blue. Figure 2b is the RRT* tree with grandparent connection. The FR produced the tree in Figure 2a and Figure 2c is the FR with grandparent connection. The minimum cost path to the goal region for each algorithm is outlined in black.

for a better route and it spends extra time smoothing the current best path. The proposed FR-gp is the algorithm most closely related to the RRT*-smart. In all but the 75 obstacle environment, the FR-gp has a cost error less than half of the cost error of the RRT*-smart. All of the proposed modifications perform better than the RRT*.

Number of Nodes The RRT* and FR produced paths with an increasing number of nodes, this is due to the limited connection range x_{new} has to the vertices in the tree. The grandparent connection modification substantially decreased the number of nodes in both the RRT*-gp and FR-gp. The paths from the FR simulations show an increase in the number of nodes in the path compared to the RRT*, which is expected. The RRT*-gp and FR-gp found paths with fewer nodes than their non-grandparent connection counterparts, RRT* and FR, respectively. This reduction in the number of nodes in the path is one of the desired effects that was expected.

Dubins' Vehicle

Initial simulations were also run for a Dubins' vehicle, using the same algorithm parameter values as the Euclidean simulations. The results are averaged over ten simulations in the 10, 25, and 50 obstacle environments. Each simulation was run for 10,000 iterations. The dynamics for the Dubins' vehicle are

$$\dot{x}(t) = v\cos(\theta) \tag{1a}$$

$$\dot{y}(t) = v\sin(\theta) \tag{1b}$$

$$\dot{\theta}(t) = u, \quad |u| \leq \frac{v}{\rho}, \tag{1c}$$

where v is the speed of the vehicle and ρ is the minimum turning radius. It is assumed that both v and ρ are constant ($v = 0.1$ and $\rho = 0.1$). The optimal trajectory between two configurations for these dynamic constraints is discussed in [19]. Simulations were not done in the 75 and 100 obstacle environments because they produced corridors that were too narrow for the Dubins' vehicle trajectory with $\rho = 0.1$.

Average Cost Figures 4a-4c show typical trees produced by each of the four algorithms. The FR tree, in Fig. 4a, has not explored the full configuration space as it has focused on sampling around the path to the goal. The grandparent connection modifications, in Figs. 4b and 4c, produced trees with straighter paths, which is beneficial for the Dubins' vehicle as the paths without the grandparent modification tend to be very curvy and even circular.

Table 3 compares the RRT* and RRT*-gp to the RRT and RRT with path smoothing. The average cost of the first path found and the average run-time at which it was found are shown and compared to the optimal cost via the calculated error. The optimal cost is the cost of the single best path found in the simulations. For further comparison the average cost of the RRT* and RRT*-gp at a run-time close to the end of the simulation is presented. The RRT-smooth and the RRT*-gp both produce initial paths whose average cost is significantly less than those produced by the RRT and RRT*. In the 10 obstacle environment, the RRT*-gp initially finds, on average, a better path than the RRT-smooth and as the RRT*-gp continues to run its average path cost decreases further. In the 25 obstacle environment the RRT-smooth on average produces the better path, then eventually the RRT*-gp reduces its average path cost to that of the RRT-smooth.

The average path cost near the end of the simulations' run-

Copyright © 2015 by ASME

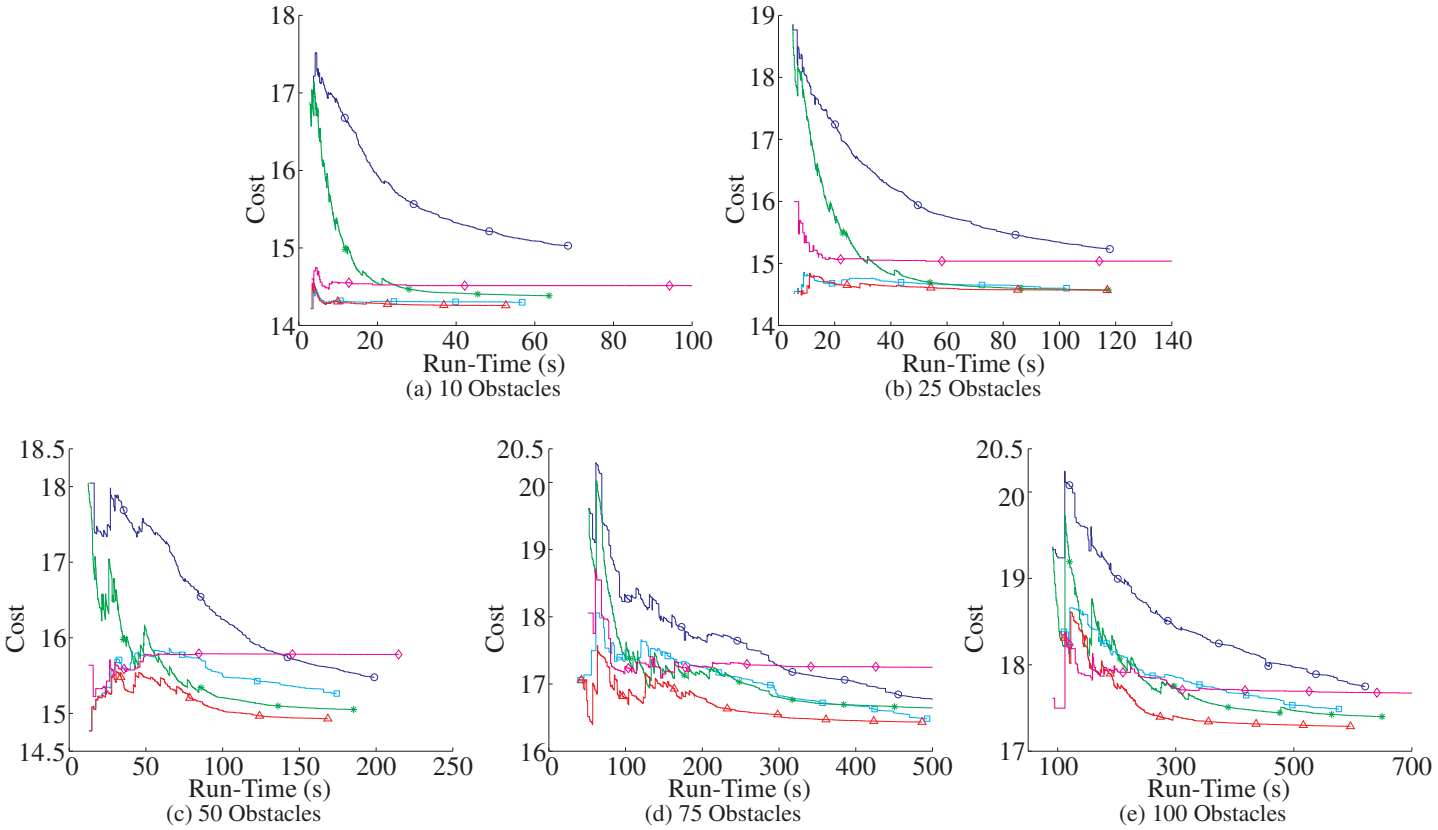


FIGURE 3: Minimum cost of the path to the goal as function of the run-time for (left-to-right, top-to-bottom) 10, 25, 50, 75, and 100 obstacles in the Euclidean environment. The blue-$\circ$ are the results for the RRT*, cyan-$\square$ for the results of the RRT*-gp, green-$*$ for the FR, red-$\triangle$ for the FR-gp, and magenta-$\diamond$ for the RRT*-smart.

times for the RRT*, RRT*-gp, FR, FR-gp, and RRT*-smart are compared in Table 4. The optimal cost is the single best cost found by the simulations. The average costs are compared to the optimal cost and the cost error is calculated. The FR-gp on average produced the best path and the RRT* produced the worst.

Number of Nodes While the Dubins' vehicle simulations show that the grandparent connection modification can, on average, reduce the number of nodes in the best path to the goal, the reduction is not as significant as seen in Euclidean simulations. The Dubins' vehicle simulations for the FR algorithms increased the number of nodes but once again, the increase is not as dramatic as seen in the Euclidean simulations.

CONCLUSION AND FUTURE WORK

In conclusion, the Focused-Refinement algorithms decrease the minimum cost path much quicker than the standard RRT*. The grandparent connection modification decreases the minimum cost path in low obstacle density environments. More work is needed to determine how effective the grandparent connection is in environments with high obstacle density. The Dubins' vehicle simulations show that the Focused-Refinement and the grandparent connection can be useful modifications to the RRT* for vehicles with differential constraints. More Dubins' vehicle simulations are needed to determine how efficient the FR and grandparent connection are in higher obstacle density environments.

More analysis will be done to analyze the performance of the Focused-Refinement and the grandparent modification for higher dimensional configuration spaces. Ways to more efficiently recover V_Π when there are two or more distinct paths are being investigated. Because the optimal path is not always within the region defined by ε and V_Π based on the first path the RRT* finds, especially in the denser environments, it may be beneficial to wait to exploit Π until it contains two or more sufficiently different paths. The tuning of ε to deal with higher obstacle density environments should also be investigated.

Copyright © 2015 by ASME

TABLE 1: This table compares, for each of the five Euclidean environments, the performance of the algorithms when they return their first path. The average costs of the first path, and average run-time at which that path was recovered, are shown for the RRT, RRT-smooth, RRT*, and RRT*-gp. The cost of the first path is compared to the optimal (best path found) cost and the error is calculated. Further comparison is given to the path cost of RRT* and RRT*-gp at a time that is near the end of the algorithms' run-time.

		First Path				Final Path		
		RRT	RRT-Smooth	RRT*	RRT*-gp	RRT*	RRT*-gp	Optimal
	Run-Time (s)	6.74	6.74	6.99	7.04	50.00	50.00	
10 obs	Cost	17.63	14.46	17.10	14.35	15.20	14.30	14.18
	Error (%)	24.33	1.95	20.57	1.22	7.20	0.84	
	Run-Time (s)	11.88	11.92	13.77	12.98	100.00	100.00	
25 obs	Cost	18.34	15.13	17.65	14.78	15.34	14.60	14.31
	Error (%)	28.17	5.77	23.41	3.30	7.20	2.07	
	Run-Time (s)	33.82	34.00	37.72	32.92	160.00	160.00	
50 obs	Cost	19.24	16.39	18.08	15.99	15.63	15.33	14.44
	Error (%)	33.25	13.51	25.17	10.73	8.25	6.13	
	Run-Time (s)	101.74	102.24	139.33	115.68	400.00	400.00	
75 obs	Cost	21.62	18.74	18.84	17.90	16.85	16.60	15.14
	Error (%)	42.83	23.81	24.46	18.26	11.34	9.64	
	Run-Time (s)	156.68	157.30	211.94	180.26	450.00	450.00	
100 obs	Cost	21.93	19.04	19.11	18.24	17.97	17.54	17.11
	Error (%)	28.20	11.33	25.17	10.73	5.06	2.53	

ACKNOWLEDGMENT

This work was supported by Los Alamos National Laboratory and is approved for public release under LA-UR-15-20423. Los Alamos National Laboratory, an affirmative action/equal opportunity employer,is operated by the Los Alamos National Security, LLC for the National Nuclear Security Administration of the U.S. Department of Energy under contract DE-AC52-06NA25396. By approving this article, the publisher recognizes that the U.S. Government retains nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy. Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.

REFERENCES

[1] LaValle, S. M., 2006. *Planning algorithms*. Cambridge University Press.

[2] Kavraki, L. E., Svestka, P., Latombe, J.-C., and Overmars, M. H., 1996. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces". *IEEE Transactions on Robotics and Automation,* ***12***(4), pp. 566–580.

[3] Waringo, M., and Henrich, D., 2006. "Efficient smoothing of piecewise linear paths with minimal deviation". In IEEE/RSJ Int. Conf. on Intelligent Robots & Systems, pp. 3867–3872.

[4] Carpin, S., and Pillonetto, G., 2005. "Motion planning using adaptive random walks". *IEEE Transactions on Robotics,* ***21***(1), pp. 129–136.

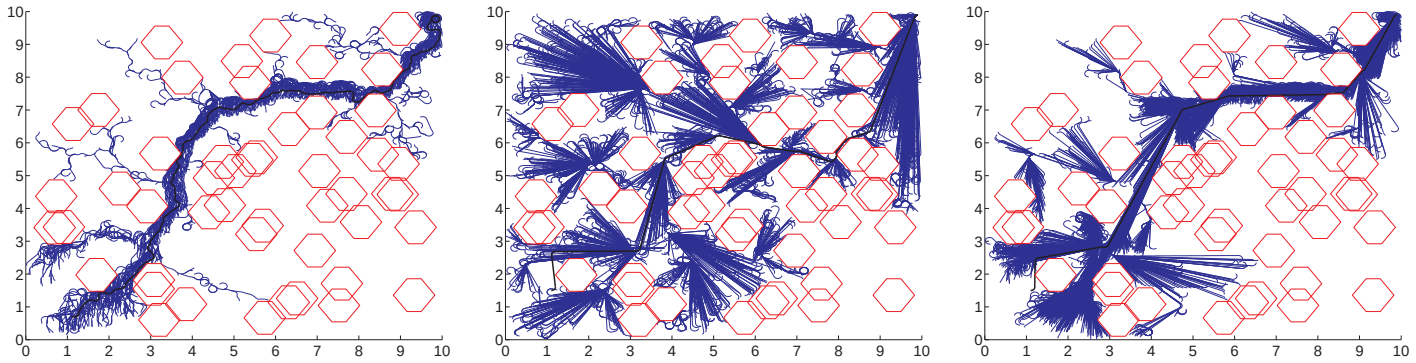
[5] Sánchez, G., and Latombe, J.-C., 2003. "A single-query bi-directional probabilistic roadmap planner with lazy collision checking". In *International Symposium on Robotic Research*. Springer, pp. 403–417.

[6] Karaman, S., and Frazzoli, E., 2011. "Sampling-based al-

Copyright © 2015 by ASME

TABLE 2: For the Euclidean environments, the average path cost at a given time, close to the algorithms' total run-time, is compared to the optimal (best path found) cost and the error is calculated for the RRT*, RRT*-gp, FR, FR-gp, and RRT*-smart.

# Obstacles	Run-Time (s)		RRT*	RRT*-gp	FR	FR-gp	RRT*-smart	Optimal
10	50	Cost	15.20	14.30	14.40	14.26	14.51	14.18
		Error (%)	7.20	0.84	1.54	0.55	2.35	
25	100	Cost	15.34	14.60	14.59	14.57	15.04	14.31
		Error (%)	7.20	2.07	1.96	1.86	5.12	
50	160	Cost	15.63	15.33	15.07	14.94	15.78	14.44
		Error (%)	8.25	6.13	4.35	3.42	9.27	
75	450	Cost	16.85	16.60	16.67	16.44	17.25	15.14
		Error (%)	11.34	9.64	10.09	8.61	13.96	
100	500	Cost	17.97	17.54	17.45	17.30	17.69	17.11
		Error (%)	5.06	2.53	1.99	1.15	3.39	



(a) FR (b) RRT*-gp (c) FR-gp

FIGURE 4: These are the best paths produced in the 50 obstacle environment for the ten Dubins' vehicle simulations. The obstacles are outlined in red. Figure 4b is the RRT* tree with grandparent connection. The FR produced the tree in Figure 4a and Figure 4c is the RRT* with both the FR and grandparent connection.

gorithms for optimal motion planning". *International Journal of Robotics Research,* ***30***(7), pp. 846–894.

[7] Karaman, S., Walter, M. R., Perez, A., Frazzoli, E., and Teller, S., 2011. "Anytime motion planning using the RRT*". In IEEE Int. Conf. on Robotics and Automation, pp. 1478–1483.

[8] Perez, A., Karaman, S., Shkolnik, A., Frazzoli, E., Teller, S., and Walter, M. R., 2011. "Asymptotically-optimal path planning for manipulation using incremental sampling-based algorithms". In IEEE/RSJ Int. Conf. on Intelligent Robots & Systems, pp. 4307–4313.

[9] Shkolnik, A., and Tedrake, R., 2011. "Sample-based planning with volumes in configuration space". *Computing Research Repository,* ***arXiv:1109.3145***.

[10] Akgun, B., and Stilman, M., 2011. "Sampling heuristics for optimal motion planning in high dimensions". In IEEE/RSJ Int. Conf. on Intelligent Robots & Systems, pp. 2640–2645.

[11] Islam, F., Nasir, J., Malik, U., Ayaz, Y., and Hasan, O., 2012. "RRT-smart: Rapid convergence implementation of RRT towards optimal solution". In IEEE Int. Conf. on Mechatronics and Automation, pp. 1651–1656.

[12] Karaman, S., and Frazzoli, E., 2010. "Optimal kinodynamic motion planning using incremental sampling-based methods". In IEEE Int. Conf. on Decision and Control, pp. 7681–7687.

[13] Boardman, B. L., Harden, T. A., and Martinez, S., 2014.

Copyright © 2015 by ASME

TABLE 3: This table compares, for each of the three environments, the performance of the Dubins' vehicle algorithms when they return their first path. The average cost of the first path, and average run-time at which that path was recovered, are shown for the RRT, RRT-smooth, RRT*, and RRT*-gp. The cost of the first path is compared to the optimal (best path found) cost and the error is calculated. Further comparison is given to the path cost of RRT* and RRT*-gp at a time that is near the end of the algorithms' run-time.

		First Path				Final Path		
		RRT	RRT-Smooth	RRT*	RRT*-gp	RRT*	RRT*-gp	Optimal
10 obs	Run-Time (s)	48.03	52.17	17.60	135.54	450.00	450.00	122.49
	Cost	354.16	128.30	334.30	126.36	320.57	125.08	
	Error (%)	189.12	4.74	172.92	3.16	161.71	2.11	
25 obs	Run-Time (s)	62.98	70.35	65.95	232.71	1200.00	1200.00	124.88
	Cost	372.66	133.01	360.49	134.71	339.53	133.44	
	Error (%)	198.41	6.51	188.67	7.87	171.88	6.85	
50 obs	Run-Time (s)	551.41	586.20	211.94	858.87	1800.00	1800.00	134.64
	Cost	418.65	151.85	394.27	149.70	374.13	148.34	
	Error (%)	210.95	12.79	192.84	11.19	177.88	10.18	

TABLE 4: The average path cost for a Dubins' vehicle at a given time, close to the algorithms' total run-time, is compared for the RRT*, RRT*-gp, FR, FR-gp, and RRT*-smart. For each of the environments the path costs are compared to the optimal (best path found) cost and the error is calculated.

# Obstacles	Run-Time (s)		RRT*	RRT*-gp	FR	FR-gp	RRT*-smart	Optimal
10	450	Cost	320.57	125.08	149.71	124.30	127.23	122.49
		Error (%)	161.71	2.12	22.22	1.47	3.86	
25	1200	Cost	339.53	133.44	155.99	129.14	132.72	124.88
		Error (%)	171.88	6.85	24.91	3.41	6.28	
50	1200	Cost	374.1283	148.3430	175.3647	144.17	149.67	134.64
		Error (%)	177.88	10.18	30.25	7.08	11.16	

"Optimal kinodynamic motion planning in environments with unexpected obstacles".

[14] Ferguson, D., Kalra, N., and Stentz, A., 2006. "Replanning with RRTs". In IEEE Int. Conf. on Robotics and Automation, pp. 1243–1248.

[15] Zucker, M., Kuffner, J., and Branicky, M., 2007. "Multipartite RRTs for rapid replanning in dynamic environments". In IEEE Int. Conf. on Robotics and Automation, pp. 1603–1609.

[16] Bruce, J., and Veloso, M., 2002. "Real-time randomized path planning for robot navigation". In IEEE/RSJ Int. Conf. on Intelligent Robots & Systems, Vol. 3, pp. 2383–2388.

[17] Li, T.-Y., and Shie, Y.-C., 2002. "An incremental learning approach to motion planning with roadmap management". In IEEE Int. Conf. on Robotics and Automation, Vol. 4, pp. 3411–3416.

[18] Gayle, R., Klingler, K. R., and Xavier, P. G., 2007. "Lazy Reconfiguration Forest (LRF) - An approach for motion planning with multiple tasks in dynamic environments.". In IEEE Int. Conf. on Robotics and Automation, pp. 1316–1323.

[19] Dubins, L. E., 1957. "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents". *American Journal of Mathematics*, pp. 497–516.

Copyright © 2015 by ASME