Quantitatively Evaluating Performance of RRTs

Ananth Ranganathan
ananth@cc.gatech.edu
CS-7630 Project Report

April 22, 2003

1 Introduction

Rapidly-Exploring Random Trees (RRTs) [LaValle98] have been the focus of a significant amount of research recently. This simple algorithm provides an efficient randomized technique to tackle the difficult problem of motion planning involving high dimensional spaces. Various extensions and modifications have also been proposed to the basic algorithm [LaValle01]. While results regarding the probabilistic completeness (assurance of finding a path if one exists as the number of nodes approaches infinity) and the convergence rate of the RRT planner exist [LaValle01], there has been very little quantitative evaluation of the RRTs with respect to path quality, number of nodes required to find a solution, and scaling power with increase in dimensionality. This project aims to rectify this deficit through an empirical study of RRTs applied to various environments and differing non-holonomic constraints.

This project applies RRTs to a variety of robots, starting with the holonomic case of a point robot operating in 2-DOF, to a car robot with three trailers operating with non-holonomic constraints in a 7-DOF state space. The variation of the number of nodes required to find a path and the time to find a solution with an increase in state-space dimensionality are studied by making all the robots operate in the same environment. For the 2-DOF case the path quality obtained by use of RRTs is compared against the optimal path length. In addition, the performance of the RRT algorithm in environments with narrow passages or tight confines is also considered. The bi-directional variant is the RRT algorithm used for all experiments in this project, unless otherwise specified. All experiments are performed in simulation.

The report is laid out as follows. The next section gives an introduction to motion planning with an emphasis on non-holonomic planning. Previous work in this field is also described here. The subsequent section decribes the RRT algorithm along with its various variants. Following this, models of the robots used in the simulations are presented. A description of the experiments and results, and a discussion of the results obtained comprise the latter part of the report.

2 Robot Motion Planning

Motion Planning refers to the problem of planning a sequence of actions that an arbitrarily shaped robot should execute to move from an initial location to a specified goal location in an environment containing some obstacles. The position of the robot at any particular instant is defined, in general, by an n-tuple $x = (x_1, x_2, x_3, \dots, x_n)$. The arity of the tuple is the dimensionality of state space in which the robot exists. However, planning is performed in a configuration space, which may be distinct from the state space. The transformation from state space to configuration space is effected so that the robot is represented as a point in configuration space. The configuration space C, can be divided into two mutually exclusive subsets, namely the free configuration space C_{free}, and the obstacle space C_{obs}. The planner then has to find a path such that each configuration on the path lies in C_{free}. Motion planning was only given serious attention in the 80s starting with the Generalized Movers' Problem discussed below. Subsequently, holonomic and non-holonomic constraints were added to the basic problem and the latter is, at present, a hot topic of research. A comprehensive study of the field is contained in [Latombe91].

The basic motion planning problem is also known as the Generalized Movers' Problem [Schwartz87]. In this problem, the robot and the obstacles are assumed to be polyhedral in shape and no constraints are applied on the robot's movement. Schwartz and Sharir [Schwartz87] first solved this problem for two dimensions using a computational geometry method with complexity $O(n^5)$. However, Reif showed that the Generalized Movers' Problem for a configuration space of arbitrary dimensions is at least PSPACE-hard [Reif79]. Hence, even the basic motion planning problem is intractable due to the curse of dimensionality.

In practice, no robot can change the direction, velocity and acceleration of its motion instantaneously. Various constraints have to placed on the motion of the robot to ensure that the physical constraints operating on the robot are respected. These constraints fall into two categories - Holonomic and Non-holonomic.

2.1 Holonomic and Non-holonomic Constraints

A holonomic constraint is an equality relation among the parameters (or dimensions) of the configuration space. This relation can be solved for one of the parameters and thus, reduces the dimensionality of C by one. k such holonomic constraints reduce dimensionality by k. As an example, consider the case of a 3-dimensional object operating in 3-dimensional physical space. The configuration space is 6-DOF as there are three position and orientation coordinates respectively. If, however, the object is constrained to rotate around a fixed axis relative to itself, then C's dimensionality is only four. Holonomic constraints do not change the basic nature of the generalized movers' problem. Their only effect is to change the definition of the robot's configuration space and sometimes, its connectedness. Thus, motion planning with holonomic constraints can be performed in a similar manner to the basic motion planning problem, i.e. through use of geometrical methods.

A non-holonomic constraint is a non-integrable equation involving configuration space parameters and their derivatives. Such a constraint does not reduce the dimensionality of C, but reduces the space of possible marginal motions (i.e. the space of velocity directions) at any given configuration. It also renders the problem unsolvable using purely geometric methods. For example, consider the car-like robot in Figure 1.

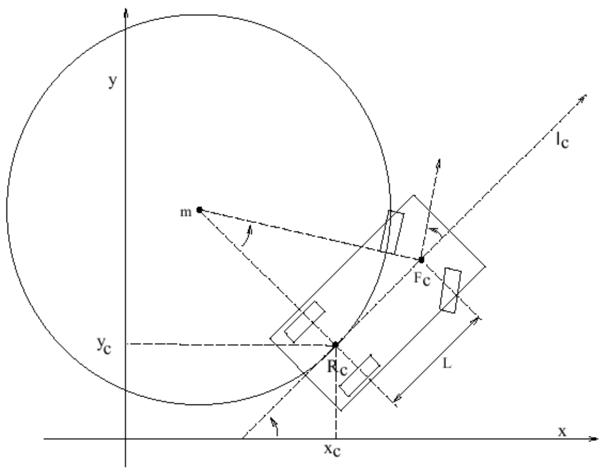


Figure 1: A car-like robot (from [Svestka93])

For this case, C is $\Re^2 \times [0, 2\pi)$. At any instant during motion, the velocity of the robot is constrained to point along the main axis of the robot. If a point in C is (x, y, θ), then this constraint can be represented mathematically by the relation -

$$-\sin\theta dx + \cos\theta dy = 0 \tag{1}$$

It can be shown that this constraint is non-integrable and hence, non-holonomic. Non-holonomic constraints restrict the geometry of the feasible free paths between two configurations and are therefore harder to deal with than holonomic ones.

2.2 Kinodynamic Planning

A specialized motion planning problem is that of *Kinodynamic planning* [Canny88][Donald93]. The kinodynamic planning problem is to synthesize a robot motion subject to simultaneous kinematic constraints, such as avoiding obstacles, and dynamic constraints, such as modulus bounds on velocity, acceleration and force. Kinematic constraints, such as joint limits and obstacles, limit the configuration of the robot. Dynamic constraints govern the time derivatives of the configurations. Hence, dynamic constraints are non-holonomic while kinematic constraints need not be.

Kinodynamic planning is generally harder than the corresponding motion planning problem. This is due to the fact that the robot may be unable to move to a configuration that is valid kinematically due to dynamic constraints; an example being finite braking time. In this project, kinodynamic planning is not implemented and hence will not be talked about further. Good references on the topic include [Donald93][Donald95] [LaValle01a].

2.3 Previous Approaches

The major approaches used to plan paths under non-holonomic constraints are -

- Roadmaps
- Randomized Potential Fields

Both these methods will be briefly described here.

2.3.1 Roadmaps

The roadmap approach first builds a network of one-dimensional curves, called a roadmap, that captures the connectivity of C_{free}. Once a roadmap $\mathscr{P}$ has been constructed, it is used as a set of standardized paths. Path planning is reduced to connecting the initial and goal states to points in $\mathscr{P}$ or searching for a path from the start to goal in $\mathscr{P}$. The constructed path is thus the concatenation of possibly three subpaths - from the start to a point on $\mathscr{P}$, a subpath in $\mathscr{P}$, and from a point on $\mathscr{P}$ to the goal.

Various types of roadmap exist. These include voronoi diagrams, visibility graphs, freeway nets, silhouttes and probabilistic roadmaps [Latombe91]. Probabilistic roadmaps are illustrated below as they most closely resemble the RRTs and provide a nice contrast to the way randomization is used in RRTs.

The probabilistic roadmap algorithm consists of two basic steps [Kavraki98]. In the first step, the roadmap is generated by randomly sampling the configuration space for points and connecting these points. Specifically, this is done as follows -

```
Algorithm Generate_PRM(N)
(∗ S is the PRM that is returned ∗)
1.   S = $\phi$
2.   for $i \leftarrow 1$ to N
3.       do
4.           Generate a random configuration $\mathscr{R}$
5.           if $\mathscr{R} \in C_{obs}$
6.              then Reject it and return to step 4
7.              else  S.add_vertex($\mathscr{R}$)
8.           Select a neighborhood of nodes $N_c$ near $\mathscr{R}$
9.           For $n \in N_c$, try and connect $n$ and $\mathscr{R}$ using a local planner
10.          If connected, S.add_edge($x_{near}, x_{new}, u$)
11.  return S
```

The nodes of the roadmap are generated in Step 1 and the edges in Step 4 above. As may be noted, the roadmap may consist of multiple unconnected components. A probabilistic roadmap constructed accoring to the above algorithm is shown in Figure 2. In the second step, called the query step, the roadmap is searched for a path from the start state to the goal state. As the generation and searching step are distinct, these can be interleaved if the query fails, and the roadmap can also be used for multiple queries.

The algorithm thus has four main components that are undefined. These are the the sampling strategy, the collision-detector, the neighborhood, and the local planner. The neighborhood of a configuration must be defined so that the local planner is able to reach most of the nodes in a neighborhood. If this is not true, a large number of failed calls to the planner may result. The planner

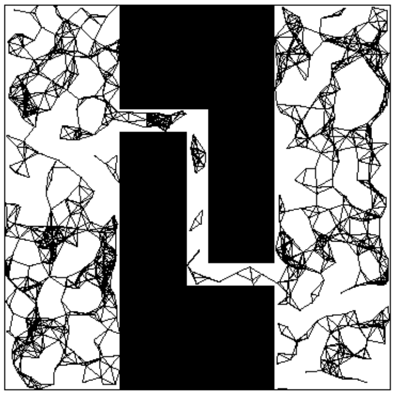


Figure 2: A Probabilistic Roadmap (from [Branicky02])

itself must be extremely simple (a “straight-line” planner) to avoid a large computational overhead at each call. The ideal sampling strategy is one which samples the environment uniformly, so as to capture the connectivity completely. Collision detection may be performed using computational geometric methods.

Advantages of Probabilistic Roadmaps (or PRMs) include the ability to interleave planning and execution and the reusability of the roadmap. A major short-coming is, however, the inability of PRMs to capture the connectivity of spaces that have narrow passages or corridor-like features. This is illustrated in Figure 2. A narrow passage results in a large number of disconnected components that the local planner may not be able to join in the query phase, thus requiring a return to the generation phase. A large amount of work in the literature has gone into finding strategies to overcome this problem, for example [Amato98][Lien03].

2.3.2 Potential Fields

The idea of potential fields to aid robot navigation is a relatively old one. The basic idea is to let the robot move under the influence of an artificial potential field produced due to the attractive influence of the goal configuration and the repulsive influence of the C obstacles. To overcome the curse of dimensionality, a randomized technique is used. The randomization incorporated may vary but the aim usually is to build a graph of nodes as in the case of PRMs. This graph may be of the local minima of the potential field, which can be found by random motion followed by gradient descent [Barraquand90]. The graph thus obtained, is searched for a path from the start to the goal. As this method is computationally much more complex than PRMs, it is less widely used. Another problem is that the definition of the potential function is extremely important in this method and becomes difficult in an environment with obstacles and kinematic constraints.

3 Rapidly-Exploring Random Trees (RRTs)

RRTs are randomized algorithms that have been designed for a broad class of planning problems. They can handle non-holonomic constraints and high dimensional spaces. They avoid problems that are inherent in other commonly used methods, such as the definition of a correct potential function for the field method and choice of an appropriate local planner for the PRMs. Both these definitions involve the creation of heuristics, which reduces the generality of the method. The name RRT refers to both a data structure, which is a graph in C space, and an algorithm that generates the data structure and finds a path by searching through it. A description of the RRT algorithm is given below.

Let a *state transition equation*, used to represent the non-holonomic constraints, be $\dot{x} = f(x,u)$ where x is a configuration and the vector u is selected from a set of inputs U. By integrating f over a fixed time interval Δt, the next state, x_{new}, can be obtained for a given $x \in C$ and $u \in U$. Though Euler integration, $x_{new} = x + f(x,u)\Delta t$, can be used, it is usually preferable to use a higher-order integration technique such as Runge-Kutta. Let $NEW\_STATE(x,u,\Delta t)$ denote an algorithm that returns x_{new}. For a given initial state, x_{init}, an RRT, τ, with K vertices is constructed as follows [LaValle98] -

```
Algorithm $GENERATE\_RRT(x\_init, K, \Delta t)$
1.  $\tau.init(x_{init})$;
2.  for $i \leftarrow 1$ to K
3.      do
4.          $x_{rand} \leftarrow$ RANDOM_STATE();
5.          $x_{near} \leftarrow$ NEAREST_NEIGHBOR($x_{rand}, \tau$);
6.          $u \leftarrow$ SELECT_INPUT($x_{rand}, x_{near}$);
7.          $x_{new} \leftarrow$ NEW_STATE($x_{near}, u, \Delta t$);
8.          $\tau$.add_vertex($x_{new}$);
9.          $\tau$.add_edge($x_{near}, x_{new}, u$);
10. return $\tau$
```

Let ρ be the distance metric on state space. The first vertex of τ is $x_{init} \in C_{free}$. In each iteration of the loop, a random state, x_{rand}, is generated and the closest node on the tree, x_{near}, is found in terms of ρ. The input, u, which minimizes the distance from x_{rand} to x_{near} is selected. Collision detection may be performed in this step or in the next step where a new state, x_{new}, is found by taking a "step" in the direction suggested by the u. x_{new} is added to the tree as a node, and an edge from x_{near} to x_{new} is also added. The input u is also stored with the edge as it is required to traverse the edge.

RRTs have several properties that make them widely applicable to varied planning problems. The expansion of an RRT is heavily biased towards unexplored regions. For example, in a two dimensional environment, using a euclidean metric, the tree first grows towards the four corners and subsequently "fills in the gaps" (Figure 3).

Another major advantage of RRTs over PRMs is that no unconnected components exist in the tree at any instant, so that the problem of connecting them does not arise. Also, as a set of predetermined inputs are used to generate new nodes, no local planner is necessary. Finally, an RRT planner is probabilistically complete under very general conditions, including non-convex environments.

A number of variations and modifications of the basic RRT algorithm exist. It has been observed that the basic algorithm has difficulty locating the goal. To overcome this a goal-bias may be added in the growth of the RRT. This can be simply accomplished by modifying the $RANDOM\_STATE()$ function in Algorithm *GENERATE_RRT* to return the goal state with a small

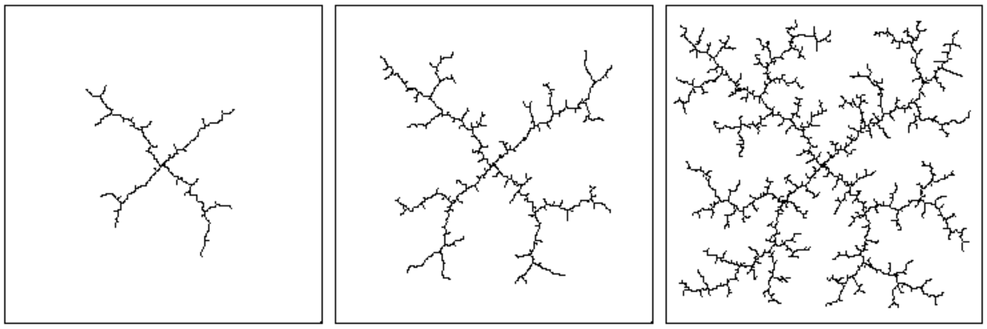


Figure 3: RRT growth in a 2-D environment with a Euclidean metric illustrating growth biased towards unexplored regions (from [LaValle98])

probability p, and a uniformly distributed random state with probability $1-p$. The resulting RRT coverges to the goal much more rapidly. However, if p is too large then the RRT behaves like a potential field algorithm and is susceptible to local minima.

Another obvious modification to the basic algorithm is to use the RRT as a bi-directional planner. Two trees, from the initial and goal states respectively, may be grown and a connection sought between them. It is seen that such a planner is much faster than a single-tree RRT planner. While this approach may be extended by growing n RRTs from different points in space, this introduces the problem of connecting these n RRTs. In very high-dimensional spaces it is hard for a node to randomly "come across" another node (of another tree) and hence in such cases the efficiency of the RRT planner may in fact decrease.

It has been mentioned that the $NEW\_STATE()$ function in Algorithm $GENERATE\_RRT$, returns a new node by taking a "step", Δt in size. However, the RRT may also be extended greedily, i.e. by making $NEW\_STATE()$ return the most distant node achievable by repeatedly applying the same input. This facilitates rapid growth of the tree towards unexplored regions but may result in the RRT overshooting the goal, and hence, in delayed convergence.

4 Implementation and Description of Models

The RRT implementation for this project was done in Python and the GUI was written in the Python's Tk interface. The emphasis in the implementation was not on efficiency - for example, the nearest neighbor search was done using the naive $O(n)$ method instead of a Kd-tree method - as all the comparisons were done using the same implementation. Hence, all the results should be viewed relatively. The RRTs implemented include the bi-directional planner, and the greedy and non-greedy methods. The default RRT used for the experiments was a bi-directional, non-greedy algorithm. This was picked as it usually results in the best convergence rate. In cases where the unidirectional planner was used, it too used a non-greedy technique.

For each model, the system expects a directory as input which contains configuration files specifying the start and goal states, the discrete input set and the location of obstacles in the environment. It is a strength of the RRT algorithm that no mention of state space dimensionality is needed in the program; the code is independent of the dimensionality.

The implementation consists of three separate modules - the *Model* module, the *RRT* module, and the *Simulation* module. These are described below -

- **The Model module**: This module encapsulates all model specific information, such as robot shape, state transition equation applicable, collision detection, dimensionality of space and the distance metric for the model. It consists of five classes in total; one base class and a class each for each of the models. The *model* base class is a superclass for the *PointModel* and the *CarModel* classes. These represent the 2D point object and the car-like robot respectively. The *TrailerModel* class is derived from *CarModel* and the *Trailer3Model* is derived from *TrailerModel* in turn.

- **The RRT module**: This module implements the actual RRT algorithm and its variant. It works on the abstract data structure and does not have to deal with any model-specific information, as all such information is abstracted away in the *Model* module described above. The module consists of two classes - *RRTBase* and *RRTDual*. The first of these is the base class that defines basic RRT functionality while the latter extends this to provide a bi-directional planner. Both the greedy and non-greedy versions are implemented as functions in *RRTBase*.

- **The Simulation module**: This module handles the GUI part of the application by getting information from the *RRT* and *Model* and displaying it graphically.

4.1 Models used in the simulation

This section describes the models used in the simulation experiments. A total of four different models are used in the experiments - a 2D holonomic point robot, a 4-DOF car-like robot in 2D physical space, a 5-DOF car-like robot with a trailer in 2D physical space, and a 7-DOF car-like robot with three trailers in 2D physical space. This wide spectrum of models is used so that trends concerning the scaling of RRTs with increase in C-space dimensionality may be effectively observed. The state transition equations and the metrics used in each of these models are presented below.

4.1.1 Point Robot

The holonomic point robot has a simple model. The metric used is a Euclidean distance measure. Let $p = (x,y)$ be a point in configuration space and $u = (v,\theta)$ be a input, $u \in U$, where v is the speed, θ is the direction of speed, and U is the set of inputs. Then the state equations are simply -

$$\begin{aligned} \dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \end{aligned} \tag{2}$$

4.1.2 Car-like Robot

A car-like robot is non-holonomic in the sense that it has a finite turning radius and it's velocity is constrained to point along its major axis. Referring back to Figure 1, the direction of the R_c's velocity points along l_c. In addition, the angle ϕ between the direction of F_c's velocity and l_c, called the *steering angle*, is constrained to lie in $[-\phi_{max}, \phi_{max}]$, where $\phi_{max} \in [0, \pi]$. In the present case, both the velocity and the steering angle are discrete functions as there exists a finite set of inputs U. In addition, suppose the length of the car (distance between R_c and F_c) is L. If a configuration in configuration space is represented as $p = (x,y,\theta,\phi)$ and $u = (v,\phi_i)$ is an input with v being the velocity and ϕ being the steering angle, then the state transition equations for change in x and y are same as in equation 3. The corresponding equations for θ and ϕ are -

$$\dot{\theta} = \frac{v \tan\phi}{L} \qquad (3)$$
$$\dot{\phi} = \phi_i$$

and a metric, given two configurations $p_1 = (x_1,y_1,\theta_1,\phi_1)$ and $p_2 = (x_2,y_2,\theta_2,\phi_2)$, can be defined as -

$$m_{car} = (x_1 - x_2)^2 + (y_1 - y_2)^2 + \frac{50 * (\theta_1 - \theta_2)^2}{\pi} + \frac{2 * (\phi_1 - \phi_2)^2}{\pi} \qquad (4)$$

Note that this is not the only metric possible.

4.1.3 Car-like robot with trailer

Configuration space for a car-like robot with a trailer is five dimensional, with the first four dimensions being the same as above and the fifth dimension being the hitch angle (angle between the orientation of the car and that of the trailer) . The hitch angle α is constrained to lie in $[-\alpha_{max}, \alpha_{max}]$. Suppose H is the hitch length, or length of the rod joining the car to the trailer. If a configuration is represented as $p = (x,y,\theta,\phi,\alpha)$ and $u = (v,\phi_i)$ is an input with v being the velocity and ϕ being the steering angle, then the state equations for x,y,θ and ϕ are the same as in equations 3 and 4. The equation for change in α is -

$$\dot{\alpha} = \frac{v \sin(\theta - \alpha)}{H} \qquad (5)$$

and given two configurations $p_1 = (x_1,y_1,\theta_1,\phi_1,\alpha_1)$ and $p_2 = (x_2,y_2,\theta_2,\phi_2,\alpha_2)$, a possible metric can be defined as -

$$m_{trailer} = (x_1 - x_2)^2 + (y_1 - y_2)^2 + \frac{5 * (\theta_1 - \theta_2)^2}{\pi} + \frac{2 * (\phi_1 - \phi_2)^2}{\pi} + \frac{5 * (\alpha_1 - \alpha_2)^2}{\pi} \qquad (6)$$

4.1.4 Car-like robot with three trailers

This case is a straight-forward extension of the previous model. As there are three hitch-angles and hitch-lengths each, two more terms are added in the state transition equation and the metric function. Let the three hitch-angles be denoted by α,β,γ and the corresponding hitch-lengths be H_1, H_2, H_3. Then a configuration can be represented as $p = (x, y, \theta, \phi, \alpha, \beta, \gamma)$ and is seven-dimensional. Assuming an input as before, the state transition equations for x, y, θ, ϕ and α are the same as in equations 3,4 and 5. The equations governing the change of β and γ are -

$$\dot{\beta} = \frac{v \cos(\theta - \alpha) \sin(\alpha - \beta)}{H_2} \tag{7}$$
$$\dot{\gamma} = \frac{v \cos(\theta - \alpha) \cos(\alpha - \beta) \sin(\beta - \gamma)}{H_3}$$

and similarly, given two configurations $p_1 = (x_1, y_1, \theta_1, \phi_1, \alpha_1, \beta_1, \gamma_1)$ and $p_2 = (x_2, y_2, \theta_2, \phi_2, \alpha_2, \beta_2, \gamma_@)$, a possible metric can be defined as -

$$m_{3trailers} = m_{trailer} \frac{5 * (\beta_1 - \beta_2)^2}{\pi} + \frac{5 * (\gamma_1 - \gamma_2)^2}{\pi} \tag{8}$$

where $m_{trailer}$ is taken from equation 6 above.

5 Description of Experiments

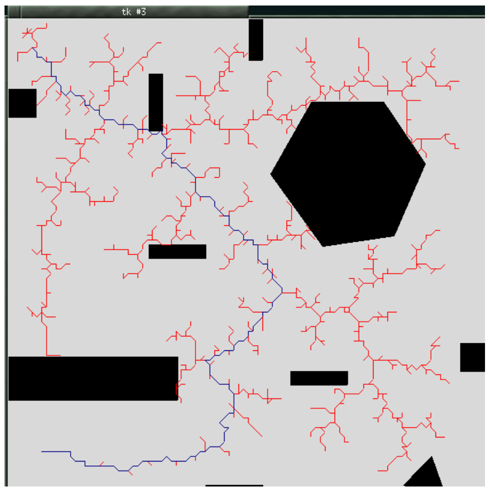


Figure 4: Goal-biased RRT for 2D holonomic robot

The first set of experiments was conducted in the environment (Environment 1) shown in Figure 4, which also shows the RRT generated for a 2D point robot. This environment was chosen as it represents a typical cluttered environment used in many other robot experiments. While for the case of the 2D robot, both the unidirectional and bi-directional planners were used, only the bidirectional planner was used for other higher dimensional cases as the unidirectional planner was too slow in these environments. This environment was used with every model and in each case the robot was required to move from the start place to the goal place which were kept fixed across all the experiments. The path length, time taken to find a path and the number of nodes required to find the path were noted during each trial. Every experiment was performed 20 times so as to

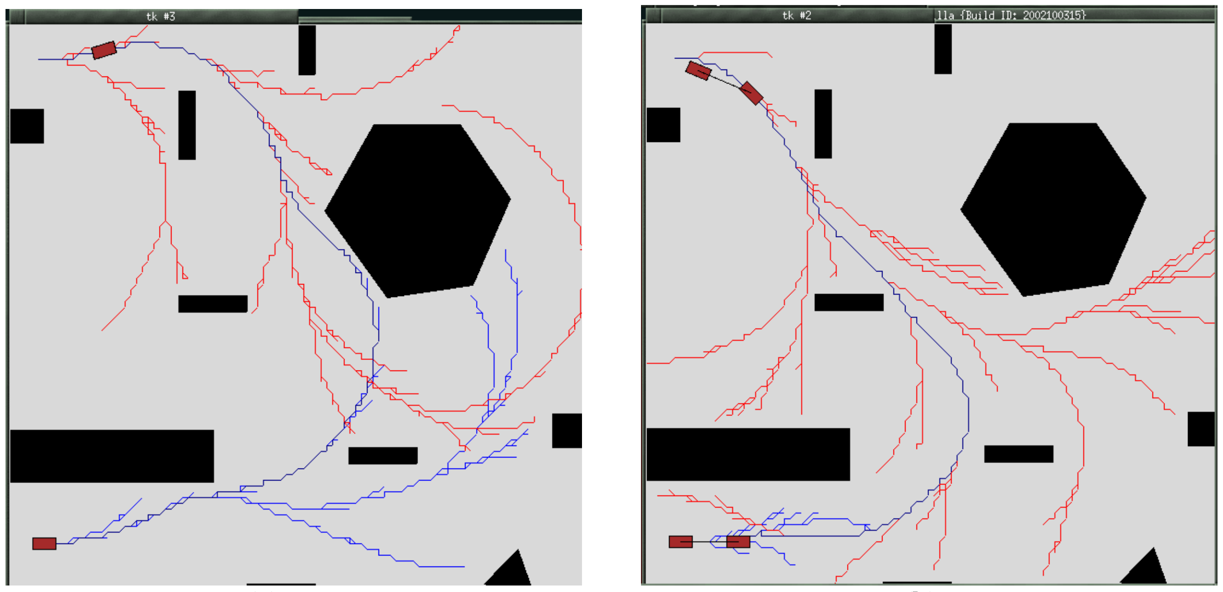


Figure 5: Goal-biased bidirectional RRTs for (a) a car-like robot (4-DOF) and (b) a car-like robot with trailer (5-DOF)

obtain statistically valid results. A tree generated using the car-like robot, and using the trailer model in the same environment are also shown in Figure 5.

The second set of experiments was conducted to assess the performance of RRTs in an environment that requires tighter maneuvering. It is well known that the performance of PRMs suffers in such environments. As RRTs have the property of growing towards unexplored areas, it is expected that their performance should not degrade more than slightly in such environments. This set of experiments was conducted to verify and quantify this hypothesis. The environment (henceforth known as Environment 2) chosen is shown in Figure 6 along with a path generated for the case of car-like robot with 3 trailers, a 7-DOF problem. This environment was chosen as the robot has to get out of a narrow passage and find the entrance to another passage to reach the goal. Even though the entrance to this adjacent passage is physically close, it is not close in configuration space of the car-like robots, as these have a finite turning radius and cannot turn into the passage leading to the goal directly. As before, the goal and start positions were kept fixed and all experiments were conducted 20 times.

6 Results

For the 2D robot in the two test environments, the mean path length of the path generated by the bi-directional RRT was on average 1.4 times longer than the optimal path length. The specific numbers for this case and for the case of the unidirectional planner are given in Figure 7. The single RRT planner produced a path that was on average 1.5 times longer than the corresponding optimal paths. This comparison was not extended to the other models as the calculation of shortest paths for non-holonomic robots is a hard problem and has itself been the focus of much research

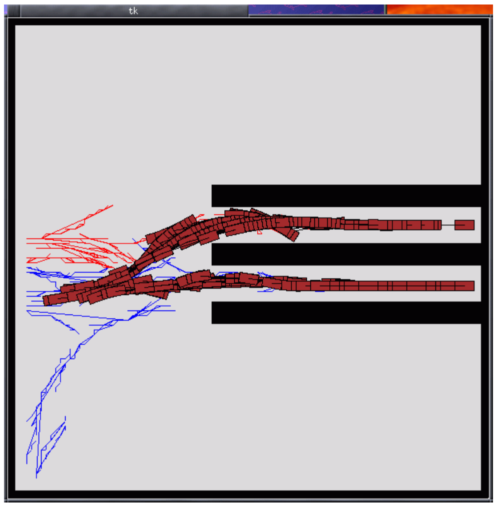


Figure 6: An environment with tight confines and the path generated for a car-like robot with 3 trailers in 7D configuration space

	Single RRT		Dual RRT		Shortest Path
	Mean	Std. Dev.	Mean	Std. Dev.	
Environment 1	167.2	17.94	143.29	6.22	111.25
Environment 2	75.6	12.48	77.35	10.28	50.42

Figure 7: Path lengths generated by RRTs compared with the optimal for a 2D robot

(for eg., [Soueres98]). However, it was observed that the average path length produced for even the 5-DOF robot was less than two times the length of the optimal path in the 2-DOF case, thus suggesting a generally good path quality.

The comparison of the efficiency of uni-directional (single) RRTs and bidirectional (dual) RRTs was also done only for the 2D domain. This is because for the other higher-dimensional cases, the single RRT agorithm fails to find a solution in any reasonable amount of time. Hence, the superiority of bidirectional RRTs in high dimensional spaces in obvious. However, even for the 2D case, the dual-RRT algorithm outperforms the single-RRT one by a considerable margin. The results of this comparison are tabulated in Figure 8. This figure shows the average number of nodes and the time taken to find a path by the two algorithms and the also the corresponding standard deviations.

It can be observed from the table that the single-RRT's performance deteriorates noticably in tight environments such as Environment 2. However, the dual-RRT does not suffer any such loss of performance. In fact, it takes less time and fewer nodes to find a solution in Environment 2 than in Environment 1. This should not be construed as an improvement in performance over Environment 1 as the robot is required to traverse a shorter path in Environment 2.

The scaling of the RRT's performance with increase in dimensionality of state space was evaluated by comparing the number of nodes needed to find a solution and the time taken to find this solution for the various robot models in both the environments. Figure 9 displays the results of these comparisons. The results present an interesting picture in that they differ from expectations.

		Single RRT		Dual RRT	
		Mean	Std. Dev.	Mean	Std. Dev.
Environment 1	No. of nodes	1315.3	267.48	236.4	29.53
	Time (in secs)	11.09	4.25	0.8635	0.15
Environment 2	No. of nodes	3540.7	1232.38	153.75	33.05
	Time (in secs)	161.04	107.13	0.8365	0.28

Figure 8: Path lengths generated by RRTs compared with the optimal for a 2D robot

Though it is well-known that randomized algorithms perform better in high-dimensional spaces than deterministic ones, this is only due to the fact that no discretization of state space is necessary for randomized algorithms as in the case of deterministic ones. If such discretization is performed in high-dimensional spaces, it leads to a large memory requirement and higher computational complexity, often rendering the problem intractable. Randomized algorithms avoid this by sampling state space instead of discretizing it. However, this does not abrogate the exponential nature of the algorithm with respect to dimensionality of space. Hence, it can be expected that the RRT algorithm too should follow this trend.

The results of the experiments, on the other hand, do not support this assumption. In fact in Environment 1, the 5-DOF case takes less time and less number of nodes than the 4-DOF case. In Environment 2, the data follows the expected trend although the number of nodes required by the 4-DOF and 5-DOF case are almost equal. Moreover, in all cases the variance of data is extremely large and renders the comparisons statistically insignificant in many cases (for eg., the time taken n Environment 2).

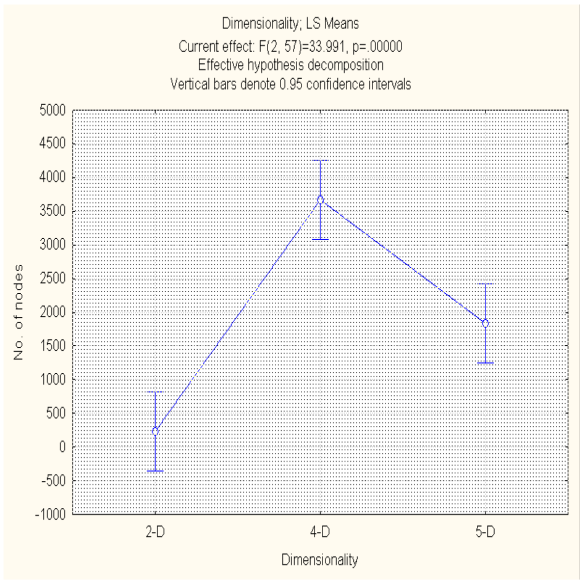
This large variance may be attributed to an insufficient number of trials. It is observed that in some trials the robot reaches the goal with very little exploration of the state space by the RRT while in others the RRT explores almost the whole state space before a solution is found. This results in large variance. It can be expected that as the number of trials increases, the variance in the data caused by this behavior should go down and a statistically significant trend should emerge. Another explanation for the results, especially for the better performance of the 5-DOF robot in Environment 1, may be that the complexity with respect to dimensionality of space is not always exponential but depends on the environment. Thus, in some environments the decrease in performance with increase in dimensionality may not be evident.

7 Conclusion and Discussion

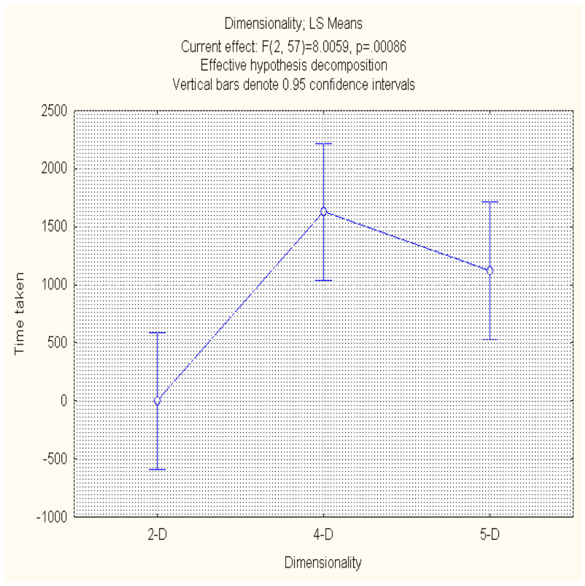
The following is a short summary of the results obtained in the project -

- The paths produced by the RRT are generally 1.3 to 1.5 times the optimal path length, at least for the case of holonomic robots; and are generally less than twice this length even for non-holonomic robots of up to five degrees of freedom.

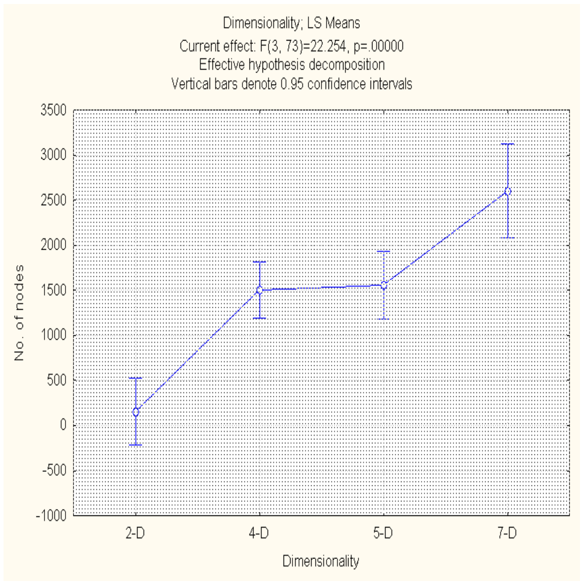
- The bi-directional RRT planner is much superior to the single RRT one even in the two dimensional case. For higher dimensional problems, it often fails to find a solution even after considerable intervals of time.



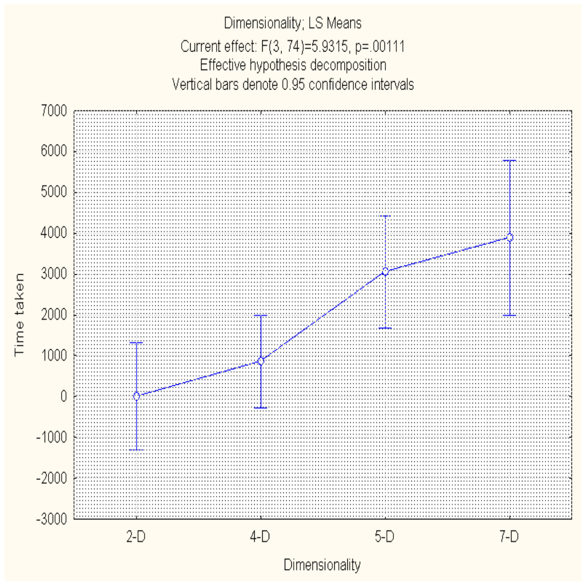
(a)



(b)



(c)



(d)

Figure 9: Scaling of performance of RRTs with respect to dimensionality of configuration space: (a) No. of nodes needed to find a solution of Environment 1. (b) Time required to find a solution in Environment 2. (c) No. of nodes required in Environment 2. (d) Time required in Environment 2.

- The single RRT planner also performs worse in tight environments while the bi-directional planner is unaffected in comparison to sparser environments. While in the experiments in this project this was found to hold true all the time, it is suspected that the bi-directional planner may also degrade in performance in environments with narrow passages in configuration space, though this degradation is expected be much less than in the single-RRT planner.
- No statistically valid conclusions could be obtained regarding the scaling of performance of the RRTs with increase in configuration space dimensionality. This may be due to insufficient number of trials or due to the dependence of the RRT's performance on the environment. In the latter case, a better approach would be to consider several widely disparate environments and repeat the tests conducted here on all of them.

Two main problems with the RRTs were observed during the course of this project. Firstly, in the case of the bi-directional RRTs, the joining of the two trees requires a long time and a lot of exploration, especially as the number of dimensions increase. For example, it was observed that in the case of the car-like robot (4-DOF), the two trees would often get to within 5 units of each other (according to the metric of Equation 4) with a total of only a few hundred nodes generated. The initial separation between the start and the goal states was about 10000 for Environment 1. After this phase, the trees would grow slowly and often away from each other before joining up after more than a thousand more nodes had been generated. This inefficiency can be removed, at the cost of added complexity to the algorithm, by using a local planner to connect the trees once their proxomity has been detected.

The second problem present in the RRT algorithm is the lack of indication by the algorithm if no path exists to the goal. Even though the RRT algorithm knows the environment completely (i.e., it has a map of the environment), it cannot provide information regarding the reachability of the goal. This problem was faced in this project when experimenting using the 7-DOF robot (car robot with three trailers), in Environment 1. No path was found even after an overnight trial (lasting about 10 hours), but it is uncertain whether this was due to the presence of a very narrow passage in configuration space or due to the absence of a path from the start state to the goal state.

The initial goal of this project was to gain a quantitative perspective in to the performance of the RRT algorithm and its variants in various environments and using various robot models. While the complete goal was not achieved due to statistical insufficiency of the data, the implementation yielded valuable knowledge about the working of motion planning algorithms. It is hoped that the insights gained during this project will help in future research on the RRTs in particular and motion planning in general.

References

[Amato98] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo. "OBPRM: An Obstacle-Based PRM for 3D Workspaces". *In Proceedings of the Workshop on Algorithmic Foundations of Robotics (WAFR'98)* , 1998, pp. 155–168.

[Barraquand90] J. Barraquand, J. -C. Latombe. "A Monte-Carlo Algorithm for Path Planning with Many Degrees of Freedom", *In Proc. IEEE Int. Conf. Robotics and Automation*, 1990, pp. 1712–1717.

[Branicky02] M. Branicky, S. M. LaValle, K. Olsen, and L. Yang. “Deterministic vs. probabilistic roadmaps”. *Unpublished manuscript, From http://msl.cs.uiuc.edu/ lavalle/papers.html*, 2002.

[Canny88] J. Canny, J. Reif, B. Donald, and P. Xavier. “On the complexity of kinodynamic planning”. *In Proc. IEEE Conf. on Foundations of Computer Science*. 1988, pp. 306– 316.

[Donald93] B. Donald, P. Xavier, J. Canny, and J. Reif. “Kinodynamic motion planning”. *Journal of the ACM*, vol. 40, No. 5, 1993, pp. 1048–1066.

[Donald95] B. Donald and P. Xavier. “Provably good approximation algorithms for optimal kinodynamic planning for Cartesian robots and open chain manipulators”. *Algorithmica*, vol. 14, no. 6, 1995, pp. 480-530.

[Kavraki98] L. Kavraki and J.-C. Latombe. “Probabilistic Roadmaps for Robot Path Planning”. *Practical Motion Planning in Robotics: Current Approaches and Future Directions*, (K. Gupta and P. del Pobil, eds.), John Wiley, 1998, pp. 33–53.

[Latombe91] J.-C. Latombe. “Robot Motion Planning”. Kluwer Academic Publishers, Boston, 1991.

[LaValle98] S. M. LaValle. “Rapidly-exploring random trees: A new tool for path planning”. *Technical Report 98-11*, Iowa State University, 1998.

[LaValle01] S.M. LaValle and J.J. Kufner. “Rapidly-exploring random trees: Progress and prospects”. *In Algorithmic and Computational Robotics: New Directions*. (D. Rus B. Donald, K. Lynch, eds.), 2001, A.K. Peters, pp. 45–59.

[LaValle01a] S. M. LaValle and J. J. Kuffner. “Randomized kinodynamic planning”. *In International Journal of Robotics Research*, Vol. 20, No. 5, 2001 pp. 378–400.

[Lien03] J. Lien, S. L. Thomas and N. M. Amato. “A General Framework for Sampling on the Medial Axis of the Free Space”. *In Proceedings of the 2003 IEEE International Conference on Robotics and Automation (ICRA’03)*, 2003.

[Reif79] J.H. Reif. “Complexity of the Mover’s Problem and Generalizations”. *In Proceedings of the 20th IEEE Symposium on Foundations of Computer Science* , 1979, pp. 421–427.

[Schwartz87] J. T. Schwartz, M. Sharir, and J. Hopcroft. “Planning, Geometry, and the Complexity of Robot Motion”. Norwood, 1987.

[Soueres98] P. Soueres and J. D. Boisonnat. “Optimal trajectories for nonholonomic mobile robots” *In Robot motion planning and control, Lectures Notes in Control and Information Sciences 229*, (J. P. Laumond, Ed.). Springer, 1998, pp. 93–170.

[Svestka93] P. Svestka. “A probabilistic approach to motion planning for car-like robots”. *Technical Report RUU-CS-93-18*, Comp. Sci., Utrecht Univ., the Netherlands, 1993.