# RSRT: RAPIDLY EXPLORING SORTED RANDOM TREE
## *Online Adapting RRT to Reduce Computational Solving Time while Motion Planning in Wide Configuration Spaces*

Nicolas Jouandeau
L.I.A.S.D.
Dept. MIME
Université Paris8

Keywords: Motion-planning, soft computing.

Abstract: We present a new algorithm, named *RSRT*, for *Rapidly-exploring Random Trees* (*RRT*) based on inherent relations analysis between *RRT* components. *RRT* algorithms are designed to consider interactions between these inherent components. We explain properties of known variations and we present some future once which are required to deal with dynamic strategies. We present experimental results for a wide set of path planning problems involving a free flying object in a static environment. The results show that our *RSRT* algorithm (where *RSRT* stands for Rapidly-exploring Sorted Random Trees) is faster than existing ones. This results can also stand as a starting point of a motion planning benchmark instances which would make easier further comparative studies of path planning algorithms.

# 1 INTRODUCTION

Literally, planning is the definition of a sequence of orders which reach a previously selected goal. In a geometrical context, planning considers a workspace, an initial position, a final position and a set of constraints characterizing a mobile $\mathcal{M}$. The problem of planning could be resumed in two questions: the existence of a solution to a given problem and the definition of a solution to a problem that has at least one solution. In this paper, the problem of planning is focused on the second one, *i.e.* identifying solutions for problems that have at least one solution. The complexity of such a solution depends on the mobile workspace, its caracteristics (*i.e.* number of degrees of freedom) and the required answer complexity (*i.e.* the model and the local planner). Each dimension of these three parts contributes to define the problem dimension. Complexity is exponential in the problem dimension, so probabilistic methods propose to solve geometrical path-planning problems by finding a valid solution without guarantee of optimality. This particular relation to optimality associates probabilistic methods with problems known as difficult (also called non-deterministic polynomial in space (Canny, 1987)). In these methods, solving a path-planning problem consists in exploring the space in order to compute a solution with a determinist algorithm (Latombe, 1991). The specificity of these methods can be summarized with a random sampling of the search space, which reduces the determinist-polynomial complexity of the resolution (Schwartz and Sharir, 1983). The increase of computers capacities and the progress of the probabilistic methods, made solvable problems more complex during last decades. The principal alternatives of research space are the configuration space $C$ (Lozano-Pérez, 1983), the state space $X$ (Donald et al., 1993) and the state-time space $ST$ (Fraichard, 1993). $C$ is intended to motion planning in static environments. $X$ adds differential constraints. $ST$ adds the possibility of a dynamic environment. The concept of high-dimensional configuration spaces is initiated by J. Barraquand *et al.* (Barraquand and Latombe, 1990) to use a manipulator with 31 degrees of freedom. P. Cheng (Cheng, 2001) uses these methods with a 12 dimensional state space involving rotating rigid objects in $3D$ space. S. M. LaValle (LaValle, 2004) presents such a space with a hundred dimensions for either a robot manipulator or a couple of mobiles. The probabilistic methods mostly used in such spaces are *Randomized Path Planning* (*RPP*), *Probabilistic RoadMap* (*PRM*) and *Rapidly exploring Random*


Jouandeau N. (2007).
RSRT: RAPIDLY EXPLORING SORTED RANDOM TREE - Online Adapting RRT to Reduce Computational Solving Time while Motion Planning in Wide Configuration Spaces.
In *Proceedings of the Fourth International Conference on Informatics in Control, Automation and Robotics*, pages 100-107
DOI: 10.5220/0001622001000107

*Trees* (*RRT*). The *RPP* method introduced by J. Barraquand *et al.* (Barraquand and Latombe, 1991) is a variation of the gradient method also introduced by O. Khatib (Khatib, 1985). Random moves make possible to escape from the local minima and recover the completeness. These random moves follow a Gaussian law. Each move is independent of the previous one. With obstacles, moves must remain in free space $C_{free}$. In case of collision, moves are reflected on the obstacles. The *PRM* method is introduced simultaneously by L.E. Kavraki *et al.* (Kavraki, 1995) and by P. Svestka *et al.* (Svestka, 1997) under the heading *Probabilistic Path Planner* (*PPP*). Their resolution principle divides the path planning problem into two successive stages: a *learning phase* that builds a graph and a *query phase* that builds a solution based on the previous graph. During the *learning phase*, the graph is built in $C_{free}$ where each node is randomly selected according to a uniform distribution in $C_{free}$. This uniform distribution is justified by the need of exploring the entire free space. It is obtained by a random sampling associated to a collision detector. During the *query phase*, the graph is used to connect two configurations $q_{init}$ and $q_{obj}$ included in $C_{free}$. At each iteration, a local path planner seeks a way to connect a new node to the graph and also tries to connect $q_{init}$ and $q_{obj}$. The *RRT* method introduced by S.M. LaValle (LaValle, 1998) is based on the construction of a tree $T$ in the considered space $\mathcal{S}$. Starting from the initial position $q_{init}$, the construction of the tree is carried out by integrating control commands iteratively. Each iteration aims at bringing closer the mobile $\mathcal{M}$ to an element $e$ randomly selected in $\mathcal{S}$. To avoid cycles, two elements $e$ of $T$ cannot be identical. In practice, *RRT* is used to solve various problems such as negotiating narrow passages made of obstacles (Ferré and Laumond, 2004), finding motions that satisfy obstacle-avoidance and dynamic balances constraints (Kuffner et al., 2003), making Mars exploration vehicles strategies (Williams et al., ), searching hidden objects (Tovar et al., 2003), rallying a set of points or playing hide-and-seek with another mobile (Simov et al., 2002) and many others mentioned in (LaValle, 2004). Thus by their efficiency to solve a large set of problems, the *RRT* method can be considered as the most general one.

In the next section, we present existing *RRT* algorithms.

# 2 RAPIDLY EXPLORING RANDOM TREES

In its initial formulation, *RRT* algorithms are defined without goal. The exploration tree covers the surrounding space and progress blindly towards free space.

A geometrical path planning problem aims generally at joining a final configuration $q_{obj}$. To solve the path planning problem, the *RRT* method searches a solution by building a tree (ALG. 1) rooted at the initial configuration $q_{init}$. Each node of the tree results from the mobile constraints integration. Its edges are commands that are applied to move the mobile from a configuration to another.

The *RRT* method is a random incremental search which could be casting in the same framework of Las Vegas Algorithms (*LVA*). It repeats successively a loop made of three phases: generating a random configuration $q_{rand}$, selecting the nearest configuration $q_{prox}$, generating a new configuration $q_{new}$ obtained by numerical integration over a fixed time step $\Delta t$. The mobile $\mathcal{M}$ and its constraints are not explicitly specified. Therefore, modifications for additional constraints (such as non-holonomic) are considered minor in the algorithm formulation.

In this first version, $C$ is presented without obstacle in an arbitrary space dimension. At each iteration, a local planner is used to connect each couples $(q_{new}, q_{prox})$ in $C$. The distance between two configurations in $T$ is defined by the time-step $\Delta t$. The local planner is composed by temporal and geometrical integration constraints. The resulting solution accuracy is mainly due to the chosen local planner. $k$ defines the maximum depth of the search. If no solution is found after $k$ iterations, the search can be restarted with the previous $T$ without re-executing the init function (ALG. 1 line 1).

The *RRT* method, inspired by traditional Artificial Intelligent techniques for finding sequences between an initial and a final element (*i.e.* $q_{init}$ and $q_{obj}$) in a well-known environment, can become a bidirectional search (shortened *Bi-RRT* (LaValle and Kuffner, 1999)). Its principle is based on the simultaneous construction of two trees (called $T_{init}$ and $T_{obj}$) in which the first grows from $q_{init}$ and the second from $q_{obj}$. The two trees are developped towards each other while no connection is established between them. This bidirectional search is justified because the meeting configuration of the two trees is nearly the half-course of the configuration space separating $q_{init}$ and $q_{obj}$. Therefore, the resolution time complexity is reduced (Russell and Norvig, 2003).

```
rrt(q_init, k, Δt, C)
    init(q_init, T);
    for i ← 1 à k
        | q_rand ← randomState(C);
        | q_prox ← nearbyState(q_rand, T);
        | q_new ← newState(q_prox, q_rand, Δt);
        | addState(q_new, T);
        | addLink(q_prox, q_new, T);
    return T;
```

ALG. 1: Basic *RRT* building algorithm.

*RRT-Connect* (Kuffner and LaValle, 2000) is a variation of *Bi-RRT* that consequently increase the *Bi-RRT* convergence towards a solution thanks to the enhancement of the two trees convergence. This has been settled to:

- ensure a fast resolution for "simple" problems (in a space without obstacle, the *RRT* growth should be faster than in a space with many obstacles);
- maintain the probabilistic convergence property. Using heuristics modify the probability convergence towards the goal and also should modify its evolving distribution. Modifying the random sampling can create local minima that could slow down the algorithm convergence.

```
connectT(q, Δt, T)
    r ← ADVANCED;
    while r = ADVANCED
        | r ← expandT(q, Δt, T);
    return r;
```

ALG. 2: Connecting a configuration $q$ to a graph $T$ with *RRT-Connect*.

In *RRT-Connect*, the two graphs previously called $T_{init}$ and $T_{obj}$ are called now $T_a$ and $T_b$ (ALG. 3). $T_a$ (*respectively* $T_b$) replaces $T_{init}$ and $T_{obj}$ alternatively (*respectively* $T_{obj}$ and $T_{init}$). The main contribution of *RRT-Connect* is the ConnectT function which move towards the same configuration as long as possible (*i.e.* without collision). As the incremental nature algorithm is reduced, this variation is designed for non-differential constraints. This is iteratively realized by the expansion function (ALG. 2). A connection is defined as a succession of successful extensions. An expansion towards a configuration $q$ becomes either an extension or a connection. After connecting successfully $q_{new}$ to $T_a$, the algorithm tries as many extensions as possible towards $q_{new}$ to $T_b$. The configuration $q_{new}$ becomes the convergence configuration $q_{co}$ (ALG. 3 lines 8 and 10).

```
rrtConnect(q_init, q_obj, k, Δt, C)
1       init(q_init, T_a);
2       init(q_obj, T_b);
3       for i ← 1 à k
4           | q_rand ← randomState(C);
5           | r ← expandT(q_rand, Δt, T_a);
6           | if r ≠ TRAPPED
7                   if r = REACHED
8                       | q_co ← q_rand;
9                   else
10                      | q_co ← q_new;
11                  if connectT(q_co, Δt, T_b) =
                      REACHED
12                      | sol ← plan(q_co, T_a, T_b);
13                      | return sol;
14          | swapT(T_a, T_b);
15      return TRAPPED;
```

ALG. 3: Expanding two graphs $T_a$ and $T_b$ towards themselves with *RRT-Connect*. $q_{new}$ mentionned line 10 correponds to the $q_{new}$ variable mentionned line 9 ALG. 4.

Inherent relations inside the adequate construction of $T$ in $C_{free}$ shown in previous works are:

- the deviation of random sampling in the variations *Bi-RRT* and *RRT-Connect*. Variations include in *RRT-Connect* are called *RRT-ExtCon*, *RRT-ConCon* and *RRT-ExtExt*; they modify the construction strategy of one of the two trees of the method *RRT-Connect* by changing priorities of the extension and connection phases (LaValle and Kuffner, 2000).
- the well-adapted $q_{prox}$ element selected according to its collision probability in the variation *CVP* and the integration of collision detection since $q_{prox}$ generation (Cheng and LaValle, 2001).
- the adaptation of $C$ to the vicinity accessibility of $q_{prox}$ in the variation *RC-RRT* (Cheng and LaValle, 2002).
- the parallel execution of growing operations for $n$ distinct graphs in the variation *OR parallel Bi-RRT* and the growing of a shared graph with a parallel $q_{new}$ sampling in the variation *embarrassingly parallel Bi-RRT* (Carpin and Pagello, 2002).
- the sampling adaptation to the *RRT*

growth (Jouandeau and Chérif, 2004; Cortès and Siméon, 2004; Lindemann and LaValle, 2003; Lindemann and LaValle, 2004; Yershova et al., 2005).

By adding the collision detection in the given space $S$ during the expansion phase, the selection of nearest neighbor $q_{prox}$ is realized in $S \cap C_{free}$ (ALG. 4). Although the collision detection is expensive in computing time, the distance metric evaluation $\rho$ is subordinate to the collision detector. $U$ defines the set of admissible orders available to the mobile $\mathcal{M}$. For each expansion, the function expandT (ALG. 4) returns three possible values: REACHED if the configuration $q_{new}$ is connected to $T$, ADVANCED if $q$ is only an extension of $q_{new}$ which is not connected to $T$, and TRAPPED if $q$ cannot accept any successor configuration $q_{new}$.

```
expandT(q, Δt, T)
    q_prox ← nearbyState(q, T);
    d_min ← ρ(q_prox, q);
    success ← FALSE;
    foreach u ∈ U
        q_tmp ← integrate(q, u, Δt);
        if isCollisionFree(q_tmp, q_prox, M, C)
            d ← ρ(q_tmp, q_rand);
            if d < d_min
                q_new ← q_tmp;
                d_min ← d;
                success ← TRUE;
    if success = TRUE
        insertState(q_prox, q_new, T);
        if q_new = q
            return REACHED;
        return ADVANCED;
    return TRAPPED;
```

ALG. 4: Expanding $T$ with obstacles.

In the next section, we examine in detail justifications of our algorithm and the inherent relations in the various components used. This study enables us to synthesize a new algorithm named Rapidly exploring Sorted Random Tree (*RSRT*), based on reducing collision detector calls without modification of the classical random sampling strategy.

## 3 RSRT ALGORITHM

Variations of *RRT* method presented in the previous section is based on the following sequence :

- generating $q_{rand}$;
- selecting $q_{prox}$ in $T$;
- generating each successor of $q_{prox}$ defined in $U$.
- realizing a colliding test for each successor previously defined;
- selecting a configuration called $q_{new}$ that is the closest to $q_{rand}$ among successors previously defined; This selected configuration has to be collision free.

The construction of $T$ corresponds to the repetition of such a sequence. The collision detection discriminates the two possible results of each sequence:

- the insertion of $q_{new}$ in $T$ (*i.e.* without obstacle along the path between $q_{prox}$ and $q_{new}$);
- the rejection of each $q_{prox}$ successors (*i.e.* due to the presence of at least one obstacle along each successors path rooted at $q_{prox}$).

The rejection of $q_{new}$ induces an expansion probability related to its vicinity (and then also to $q_{prox}$ vicinity); the more the configuration $q_{prox}$ is close to obstacles, the more its expansion probability is weak. It reminds one of fundamentals *RRT* paradigm: free spaces are made of configurations that admit various number of available successors; good configurations admit many successors and bad configurations admit only few ones. Therefore, the more good configurations are inserted in $T$, the better the *RRT* expansion will be. The problem is that we do not previously know which good and bad configurations are needed during *RRT* construction, because the solution of the considered problem is not yet known. This problem is also underlined by the parallel variation *OR Bi-RRT* (Carpin and Pagello, 2002) (*i.e.* to define the depth of a search in a specific vicinity). For a path planning problem $p$ with a solution $s$ available after $n$ integrations starting from $q_{init}$, the question is to maximize the probability of finding a solution; According to the concept of “rational action”, the response of $P3$ class to adapt a on-line search can be solved by the definition of a formula that defines the cost of the search in terms of “local effects” and “propagations” (Russell, 2002). These problems find a way in the tuning of the behavior algorithm like *CVP* did (Cheng and LaValle, 2001).

In the case of a space made of a single narrow passage, the use of bad configurations (which successors generally collide) is necessary to resolve such problem. The weak probability of such configurations extension is one of the weakness of the *RRT* method.

```
newExpandT(q, Δt, T)
1     q_prox ← nearbyState(q, T);
2     S ← ∅;
3     foreach u ∈ U
4         | q ← integrate(q_prox, u, Δt);
5         | d ← ρ(q, q_rand);
6         | S ← S + {(q, d)};
7     qsort(S, d);
8     n ← 0;
10    while n < Card(S)
11        | s ← getTupleIn(n, S);
12        | q_new ← firstElementOf(s);
13        | if isCollisionFree(q_new, q_prox, M, C)
14              insertState(q_prox, q_new, T);
15              if q_new = q
16                  | return REACHED;
17              return ADVANCED;
18        | n ← n + 1;
19    return TRAPPED;
```

ALG. 5: Expanding $T$ and reducing the collision detection.

To bypass this weakness, we propose to reduce research from the closest element (ALG. 4) to the first free element of $C_{free}$. This is realized by reversing the relation between collision detection and distance metric; the solution of each iteration is validated by subordinating collision tests to the distance metric; the first success call to the collision detector validates a solution. This inversion induces:

- a reduction of the number of calls to the collision detector proportionally to the nature and the dimension of $U$; Its goal is to connect the collision detector and the derivative function that produce each $q_{prox}$ successor.
- an equiprobability expansion of each node independently of their relationship with obstacles;

The $T$ construction is now based on the following sequence:

1. generating a random configuration $q_{rand}$ in $C$;
2. selecting $q_{prox}$ the nearest configuration to $q_{rand}$ in $T$ (ALG. 5 line 1);
3. generating each successors of $q_{prox}$ (ALG. 5 lines 3 to 6); each successor is associated with its distance metric from $q_{rand}$. It produces a couple called $s$ stored in $S$;
4. sorting $s$ elements by distance (ALG. 5 lines 7);
5. selecting the first collision-free element of $S$ and breaking the loop as soon as this first element is discovered (ALG. 5 lines 16 and 17);

## 4 EXPERIMENTS

This section presents experiments performed on a Redhat Linux Cluster that consists of 8 Dual Core processor 2.8 GHz Pentium 4 (5583 bogomips) with 512 MB DDR Ram.

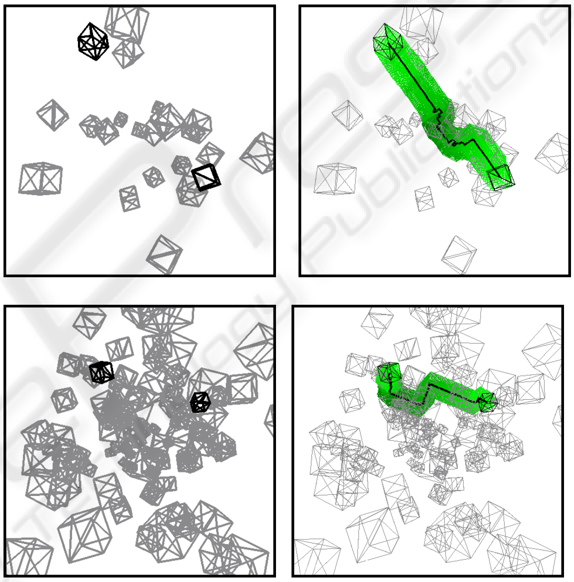

Figure 1: 20 obstacles problem and its solution (upper couple). 100 obstacles problem and its solution (lower couple).

To perform the run-time behavior analysis for our algorithm, we have generated series of problems that gradually contains more 3D-obstacles. For each problem, we have randomly generated ten different instances. The number of obstacles is defined by the sequence 20, 40, 60, ..., 200, 220. In each instance, all obstacles are cubes and their sizes are randomly varying between $(5,5,5)$ and $(20,20,20)$. The mobile is a cube with a fixed size $(10,10,10)$. Obstacles and mobile coordinates are varying between $(-100,-100,-100)$ and $(100,100,100)$. For each instance, a set of 120 $q_{init}$ and 120 $q_{obj}$ are generated in $C_{free}$. By combinating each $q_{init}$ and each $q_{obj}$, 14400 configuration-tuples are available for each instance of each problem. For all that, our benchmark is made of more than 1.5 million problems. An instance with 20 obstacles is shown in FIG. 1 on the lower part and another instance with 100 obstacles in FIG. 1

on the left part. On these two examples, $q_{init}$ and $q_{obj}$ are also visible. We used the Proximity Query Package (*PQP*) library presented in (Gottschalk et al., 1996) to perform the collision detection. The mobile is a free-flying object controlled by a discretized command that contains 25 different inputs uniformly dispatched over translations and rotations. The performance was compared between *RRT-Connect* (using the *RRT-ExtCon* strategy) and our *RSRT* algorithm (ALG. 5).

The choice of the distance metric implies important consequences on configurations' connexity in $C_{free}$. It defines the next convergence node $q_{co}$ for the local planner. The metric distance must be selected according to the behavior of the local planner to limit its failures. The local planner chosen is the straight line in $C$. To validate the toughness of our algorithm regarding to *RRT-Connect*, we had use three different distance metrics. Used distance metrics are:

- the Euclidean distance (mentioned *Eucl* in FIG. 2 to 4)

$$d(q,q') = \left(\sum_{k=0}^{i}(c_k - c'_k)^2 + nf^2\sum_{k=0}^{j}(\alpha_k - \alpha'_k)^2\right)^{\frac{1}{2}}$$

  where $nf$ is the normalization factor that is equal to the maximum of $c_k$ range values.

- the scaled Euclidean distance metric (mentioned *Eucl2* in FIG. 2 to 4)

$$d(q,q') = \left(s\sum_{k=0}^{i}(c_k - c'_k)^2 + nf^2(1-s)\sum_{k=0}^{j}(\alpha_k - \alpha'_k)^2\right)^{\frac{1}{2}}$$

  where $s$ is a fixed value 0.9;

- the Manhattan distance metric (mentioned *Manh* in FIG. 2 to 4)

$$d(q,q') = \sum_{k=0}^{i}\|c_k - c'_k\| + nf\sum_{k=0}^{j}\|\alpha_k - \alpha'_k\|$$

  where $c_k$ are axis coordinates and $\alpha_k$ are angular coordinates.

For each instance, we compute the first thousand successful trials to establish average resolving times (FIG. 2), standard deviation resolving times (FIG. 3) and midpoint resolving times (FIG. 4). These trials are initiated with a fixed random set of seed. Those fixed seed assume that tested random suite are different between each other and are the same between instances of all problems. As each instance is associated to one thousand trials, each point of each graph is the average over ten instances (and then over ten thousands trials).

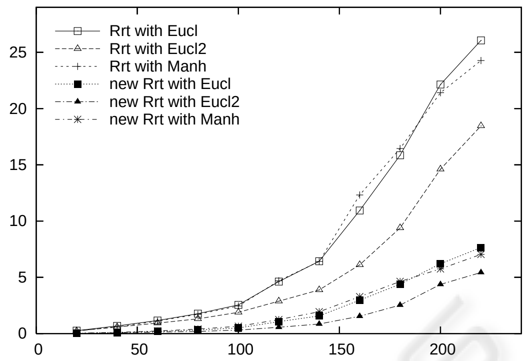


Figure 2: Averages resolving times.

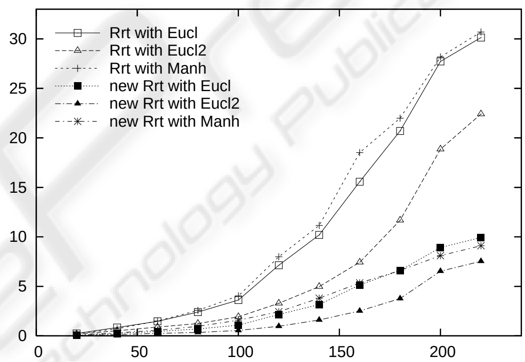


Figure 3: Standard deviation resolving times.

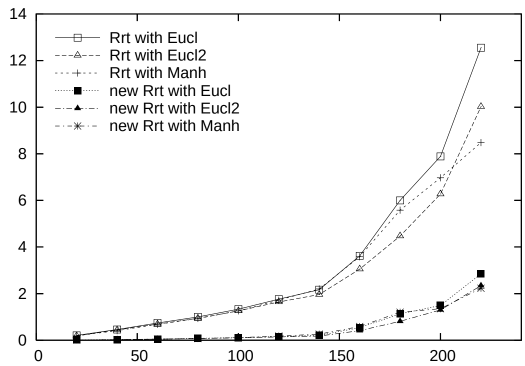


Figure 4: Midpoint resolving times.

On each graph, the number of obstacles is on x-axis and resolving time in *sec.* is on y-axis.

Figure 2 shows that average resolving time of our algorithm oscillates between 10 and 4 times faster than the original *RRT-Connect* algorithm. As the space obstruction grows linearly, the resolving time of *RRT-Connect* grows exponentially while *RSRT* algorithm grows linearly. Figure 3 shows that the standard deviation follows the same profile. It shows that *RSRT* algorithm is more robust than *RRT-Connect*. Figure 4 shows that midpoints' distributions follow the average resolving time behavior. This is a reinforcement of the success of the *RSRT* algorithm. This assumes that half part of time distribution are 10 to 4 times faster than *RRT-Connect*.

## 5 CONCLUSION

We have described a new *RRT* algorithm, the *RSRT* algorithm, to solve motion planning problems in static environments. *RSRT* algorithm accelerates consequently the resulting resolving time. The experiments show the practical performances of the *RSRT* algorithm, and the results reflect its classical behavior. The results given above( have been evaluated on a cluster which provide a massive experiment analysis. The challenging goal is now to extend the benchmark that is proposed to every motion planning methods. The proposed benchmark will be enhanced to specific situations that allow *RRT* to deal with motion planning strategies based on statistical analysis.

## REFERENCES


Barraquand, J. and Latombe, J. (1990). A Monte-Carlo Algorithm for Path Planning with many degrees of Freedom. In *Int. Conf. on Robotics and Automation (ICRA'90)*.

Barraquand, J. and Latombe, J. (1991). Robot motion planning: A distributed representation approach. *Int. Journal of Robotics Research (IJRR'91)*.

Canny, J. (1987). *The complexity of robot motion planning*. PhD thesis, Massachusetts Institute of Technology. Artificial Intelligence Laboratory.

Carpin, S. and Pagello, E. (2002). On Parallel RRTs for Multi-robot Systems. In *8th Conf. of the Italian Association for Artificial Intelligence (AI*IA'02)*.

Cheng, P. (2001). Reducing rrt metric sensitivity for motion planning with differential constraints. Master's thesis, Iowa State University.

Cheng, P. and LaValle, S. (2001). Reducing Metric Sensitivity in Randomized Trajectory Design. In *Int. Conf. on Intelligent Robots and Systems (IROS'01)*.

Cheng, P. and LaValle, S. (2002). Resolution Complete Rapidly-Exploring Random Trees. In *Int. Conf. on Robotics and Automation (ICRA'02)*.

Cortès, J. and Siméon, T. (2004). Sampling-based motion planning under kinematic loop-closure constraints. In *Workshop on the Algorithmic Foundations of Robotics (WAFR'04)*.

Donald, B., Xavier, P., Canny, J., and Reif, J. (1993). Kinodynamic Motion Planning. *Journal of the ACM*.

Ferré, E. and Laumond, J. (2004). An iterative diffusion algorithm for part disassembly. In *Int. Conf. Robotics and Automation (ICRA'04)*.

Fraichard, T. (1993). Dynamic trajectory planning with dynamic constraints: a "state-time space" approach. In *Int. Conf. Robotics and Automation (ICRA'93)*.

Gottschalk, S., Lin, M., and Manocha, D. (1996). Obb-tree: A hierarchical structure for rapid interference detection. In *Proc. of ACM Siggraph'96*.

Jouandeau, N. and Chérif, A. A. (2004). Fast Approximation to gaussian random sampling for randomized motion planning. In *Int. Symp. on Intelligent Autonomous Vehicules (IAV'04)*.

Kavraki, L. (1995). *Random networks in configuration space for fast path planning*. PhD thesis, Stanford University.

Khatib, O. (1985). Real-time obstacle avoidance for manipulators and mobile robots. In *Int. Conf. on Robotics and Automation (ICRA'85)*.

Kuffner, J. and LaValle, S. (2000). RRT-Connect: An efficient approach to single-query path planning. In *Int. Conf. on Robotics and Automation (ICRA'00)*.

Kuffner, J., Nishiwaki, K., Kagami, S., Inaba, M., and Inoue, H. (2003). Motion planning for humanoid robots. In *Int'l Symp. Robotics Research (ISRR'03)*.

Latombe, J. (1991). *Robot Motion Planning (4th edition)*. Kluwer Academic.

LaValle, S. (1998). Rapidly-exploring random trees: A new tool for path planning. Technical Report 98-11, Dept. of Computer Science, Iowa State University.

LaValle, S. (2004). *Planning Algorithms*. [on-line book]. http://msl.cs.uiuc.edu/planning/.

LaValle, S. and Kuffner, J. (1999). Randomized kinodynamic planning. In *Int. Conf. on Robotics and Automation (ICRA'99)*.

LaValle, S. and Kuffner, J. (2000). Rapidly-exploring random trees: Progress and prospects. In *Workshop on the Algorithmic Foundations of Robotics (WAFR'00)*.

Lindemann, S. and LaValle, S. (2004). Incrementally reducing dispersion by increasing Voronoi bias in RRTs. In *Int. Conf. on Robotics and Automation (ICRA'04)*.

Lindemann, S. R. and LaValle, S. M. (2003). Current issues in sampling-based motion planning. In *Int. Symp. on Robotics Research (ISRR'03)*.

Lozano-Pérez, T. (1983). Spatial Planning: A Configuration Space Approach. In *Trans. on Computers*.

Russell, S. (2002). Rationality and Intelligence. In Press, O. U., editor, *Common sense, reasoning, and rationality*.

Russell, S. and Norvig, P. (2003). *Artificial Intelligence, A Modern Approach (2ème édition)*. Prentice Hall.

Schwartz, J. and Sharir, M. (1983). On the piano movers problem:I, II, III, IV, V. Technical report, New York University, Courant Institute, Department of Computer Sciences.

Simov, B., LaValle, S., and Slutzki, G. (2002). A complete pursuit-evasion algorithm for two pursuers using beam detection. In *Int. Conf. on Robotics and Automation (ICRA'02)*.

Svestka, P. (1997). *Robot Motion Planning using Probabilistic Roadmaps*. PhD thesis, Utrecht University.

Tovar, B., LaValle, S., and Murrieta, R. (2003). Optimal navigation and object finding without geometric maps or localization. In *Int. Conf. on Robotics and Automation (ICRA'03)*.

Williams, B. C., B.C., Kim, P., Hofbaur, M., How, J., Kennell, J., Loy, J., Ragno, R., Stedl, J., and Walcott, A. Model-based reactive programming of cooperative vehicles for mars exploration. In *Int. Symp. on Artificial Intelligence, Robotics and Automation in Space*, page 2001.

Yershova, A., Jaillet, L., Simeon, T., and LaValle, S. M. (2005). Dynamic-domain rrts: Efficient exploration by controlling the sampling domain. In *Int. Conf. on Robotics and Automation (ICRA'05)*.