RRT*-Quick: A Motion Planning Algorithm with Faster Convergence Rate

In-Bae Jeong, Seung-Jae Lee, and Jong-Hwan Kim

Department of Electrical Engineering, KAIST
335 Gwahangno, Yuseong-gu, Daejeon 305-701, Republic of Korea
{ibjeong,sjlee,johkim}@rit.kaist.ac.kr

Abstract. This paper proposes RRT*-Quick as an improved version of Rapidly-exploring Random Tree Star (RRT*). The proposed RRT*-Quick utilizes one of the characteristics of RRT* that nodes in local area tend to share common parents. It uses the ancestor nodes to efficiently enlarge the pool of parent candidates for a faster convergence rate. Branch-and-bound, one of the key extensions of RRT*, prunes the unuseful nodes from the tree to help the search algorithm focus on improving solutions. Since the proposed algorithm generates the initial solution with a lower cost, it can prune unuseful nodes earlier than the conventional RRT*.

1 Introduction

Motion planning algorithm that finds a series of motions to complete the given tasks, became essential part of robotics. As robotics technology develops and the degree of freedom of a robot or a manipulator increases, motion planning problem gets harder to solve because the dimension of problems also increases.

Conventional motion planning algorithms were mainly based on graph theories that needs to discretizing the solution space. However, the resolution of discretized solution space is limited by the memory requirement, since the requirement increases exponentially to the problem dimension [1–3].

One of the commonly used techniques to solve problems in a high-dimensional continuous solution space uses a random sampling-based algorithm. It selects a randomly chosen state and tries to iteratively solve the given problem by repeating the procedure. Two of the commonly used sampling-based algorithms are Probabilistic RoadMap (PRM) and Rapidly-exploring Random Tree (RRT) [4, 5]. These algorithms have been researched extensively; therefore various variants exist to fulfill the requirement of each domain [6, 7]. Among them, RRT* is a well known RRT variant. RRT* not only finds a feasible solution but also tries to improve the solution while a robot is following the generated path [8, 9].

In this paper, an algorithm that has a faster convergence rate is proposed. To have a faster convergence rate without causing much additional computation time, it utilizes the characteristics of RRT* that nodes in local area tend to share common parents. As the cost of the best solution gets lowered, more nodes are

J.-H. Kim et al. (eds.), *Robot Intelligence Technology and Applications 3*,
Advances in Intelligent Systems and Computing 345, DOI: 10.1007/978-3-319-16841-8_7

pruned by branch-and-bound, one of the key extensions of RRT*, therefore the solutions gets improved more.

This paper is organized as follows. Section 2 introduces preliminaries on RRT and RRT*. The proposed algorithm is explained in Section 3, and the simulation environment and the results are shown in Section 4. Concluding remarks follow in Section 5.

2 Preliminaries

RRT and RRT* are briefly described with their characteristics in this section.

2.1 RRT

RRT is an algorithm that searches for a feasible trajectory in a non-convex high-dimensional space by repeatedly connecting a randomly sampled state to the existing tree if there is no collision between them, as shown in Algorithm 1. It has been proven that RRT is probabilistically complete, which means that it fully covers the whole space as the number of samples increases infinitely. Since RRT focuses on finding a feasible solution but not improving the solution, it rarely generates an optimal solution.

Input: $T = (V, E)$
Output: T
while $i < N$ **do**
  $x_{rand} \leftarrow Sample(i)$;
  $x_{nearest} \leftarrow Nearest(T, x_{rand})$;
  $x_{new} \leftarrow Extend(x_{nearest}, x_{rand})$;
  $\sigma_{new} \leftarrow Steer(x_{nearest}, x_{new})$;
  if $CollisionFree(\sigma_{new})$ **then**
    $V \leftarrow V \cup x_{new}$;
    $E \leftarrow E \cup (x_{nearest}, x_{new})$;
  end
end
$T \leftarrow (V, E)$;
return T;

Algorithm 1. RRT

2.2 RRT*

RRT* is an algorithm based on RRT, which focuses on improving the quality of solutions. While RRT simply connects a random sample to the nearest node in the tree, RRT* searches the nodes in a volume sphere with a specific radius to find the node which makes the lowest cost to get to the random sample, as described in Algorithm 2. It also tries to rewire the existing node to the random

sample if it lowers the cost. As the number of samples increases, the solution generated by RRT* asymptotically converges to the optimal solution.

There are two key extensions of RRT*, comitted trajectory and branch-and-bound.

Committed Trajectory. Comitted trajectory takes the initial portion of currently best solution then sets a new root node. While a robot follows the committed trajectory, RRT* tries to improve the new (uncommitted) tree.

Branch-and-Bound. Branch-and-bound technique is employed to reduce the number of nodes in the tree. Once a feasible solution is found, the cost of the solution is used as the upper bound. Every node in the tree which has higher cost than the upper bound is removed from the tree along with its descendents. It makes RRT* focus on improving the solution. More nodes are pruned as the upper bound gets lowered, so the computation time for processing an iteration decreases.

3 RRT*-Quick

In RRT*, the radius of the volume sphere decides the converge rate. The solution improves faster with a larger radius, but the number of nodes in the volume sphere also increases exponentially; therefore much more computation time is needed. RRT*-Quick that is proposed in this paper, utilizes the characteristics of the shape of the tree to efficiently increase the converge rate without causing much computation time.

The main idea of RRT*-Quick is based on the observation that nodes in local area tend to share a commmon parent because of the procedure 'ChooseParent' and the rewire operation of RRT*. Since commonly used metrics to measure the cost satisfies the triangular inequality, the parents are good candidates for the parent with the lowest cost.

As described in Algorithm 3, in addition to the nodes in the volume sphere, RRT*-Quick also takes account of the ancestors of them. The ancestors always provides a path with a lower (or equal) cost if the metric satisfies the triangular inequalty, even if it doesn't, the increased computation time will be negligible compared to that caused from a larger radius. In Algorithm 3, $Ancestors(x, degree)$ is a procedure that returns the set of the ancestors of x up to the given $degree$. Although three or four is enough for $degree$, setting $degree$ to infinite is fine because the number of siblings decreases exponentially as following up along the ancestry, which means it hardly affects the performance.

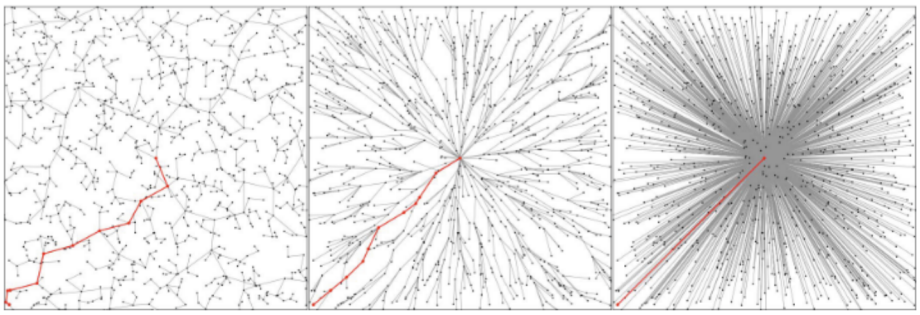
The shapes of trees of RRT, RRT*, and the proposed RRT*-Quick are depicted in Fig. 1.

Input: $T = (V, E)$
Output: T
while $i < N$ **do**
  $x_{rand} \leftarrow Sample(i)$;
  $x_{nearest} \leftarrow Nearest(T, x_{rand})$;
  $x_{new} \leftarrow Extend(x_{nearest}, x_{rand})$;
  $\sigma_{new} \leftarrow Steer(x_{nearest}, x_{rand})$;
  if $CollisionFree(\sigma_{new})$ **then**
    $X_{near} \leftarrow Near(T, x_{new})$;
    $x_{min} \leftarrow x_{nearest}$;
    $c_{min} \leftarrow Cost(x_{nearest}) + Cost(\sigma_{new})$;
    foreach $x_{near} \in X_{near}$ **do**
      $\sigma_{near} \leftarrow Steer(x_{near}, x_{new})$;
      if $Cost(x_{near}) + Cost(\sigma_{near}) < c_{min} \wedge CollisionFree(\sigma_{near})$ **then**
        $x_{min} \leftarrow x_{near}$;
        $c_{min} \leftarrow Cost(x_{near}) + Cost(\sigma_{near})$;
      end
    end
    $V \leftarrow V \cup x_{new}$;
    $E \leftarrow E \cup \{(x_{min}, x_{new})\}$;
    foreach $x_{near} \in X_{near}$ **do**
      $\sigma_{near} \leftarrow Steer(x_{new}, x_{near})$;
      if $Cost(x_{new}) + Cost(\sigma_{near}) < Cost(x_{near}) \wedge CollisionFree(\sigma_{near})$ **then**
        $E \leftarrow E \setminus \{(Parent(x_{near}), x_{near})\}$;
        $E \leftarrow E \cup \{(x_{new}, x_{near})\}$;
      end
    end
  end
end
$T \leftarrow (V, E)$;
return T;

Algorithm 2. RRT*

Input: $T = (V, E)$
Output: T
while $i < N$ **do**
  $x_{rand} \leftarrow Sample(i)$;
  $x_{nearest} \leftarrow Nearest(T, x_{rand})$;
  $x_{new} \leftarrow Extend(x_{nearest}, x_{rand})$;
  $\sigma_{new} \leftarrow Steer(x_{nearest}, x_{rand})$;
  if $CollisionFree(\sigma_{new})$ **then**
    $X_{near} \leftarrow Near(T, x_{new})$;
    $x_{min} \leftarrow x_{nearest}$;
    $c_{min} \leftarrow Cost(x_{nearest}) + Cost(\sigma_{new})$;
    foreach $x_{near} \in X_{near} \cup Ancestors(X_{near}, degree)$ **do**
      $\sigma_{near} \leftarrow Steer(x_{near}, x_{new})$;
      if $Cost(x_{near}) + Cost(\sigma_{near}) < c_{min} \wedge CollisionFree(\sigma_{near})$ **then**
        $x_{min} \leftarrow x_{near}$;
        $c_{min} \leftarrow Cost(x_{near}) + Cost(\sigma_{near})$;
      end
    end
    $V \leftarrow V \cup x_{new}$;
    $E \leftarrow E \cup \{(x_{min}, x_{new})\}$;
    foreach $x_{near} \in X_{near}$ **do**
      $\sigma_{near} \leftarrow Steer(x_{new}, x_{near})$;
      $X_{candidates} \leftarrow Ancestors(x_{new}, degree) \setminus Ancestors(x_{near}, degree)$;
      $x_{min} \leftarrow x_{new}$;
      $c_{min} \leftarrow Cost(x_{new}) + Cost(\sigma_{new})$;
      foreach $x_c \in X_{candidates}$ **do**
        $\sigma_c \leftarrow Steer(x_c, x_{near})$;
        if $Cost(x_c) + Cost(\sigma_c) < c_{min}) \wedge CollisionFree(\sigma_c)$ **then**
          $x_{min} \leftarrow x_c$;
          $c_{min} \leftarrow Cost(x_c) + Cost(\sigma_c)$;
        end
      end
      $E \leftarrow E \setminus \{(Parent(x_{near}), x_{near})\}$;
      $E \leftarrow E \cup \{(x_{min}, x_{near})\}$;
    end
  end
end
$T \leftarrow (V, E)$;
return T;

Algorithm 3. RRT*-Quick



(a) RRT (b) RRT* (c) RRT*-Quick

Fig. 1. Extended trees after 1000 iterations

4 Simulation

The effectiveness of the proposed algorithm was verified through computer simulations. RRT, RRT* and RRT*-Quick were run 50 times for each with the same set of random samples.

4.1 Simulation Environment

The simulation environment is shown in Fig. 2 with the start point and the goal area colored with red and blue, respectively. The size of the environment is 600×600 and an obstacle is blocking the direct path from the start point to the goal point. The optimal path lies on the left side of the map, but a tree tends to grow towards wider area, which is the right side of the environment; therefore local minima exist on the right side. The environment also has a narrow passage that makes harder to find a feasible path. The cost of the optimal solution is 1349.69.

4.2 Simulation Results

Major performance measures of a sampling-based motion planning algorithm are the quality of the initial solution and the convergence rate. Since a sampling-based motion planning algorithm works in a random way, the generated solution might be good or not at all. So total 50 runs of simulations were carried out and the results were analyzed statistically.

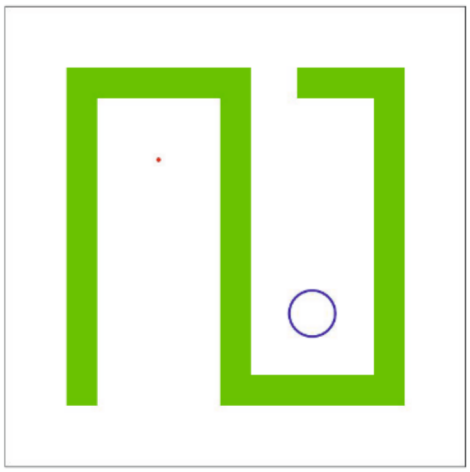


Fig. 2. Simulation environment

Initial Solution. The number of iterations and the time taken to find the initial solution were measured on each run with the cost of the initial solution.

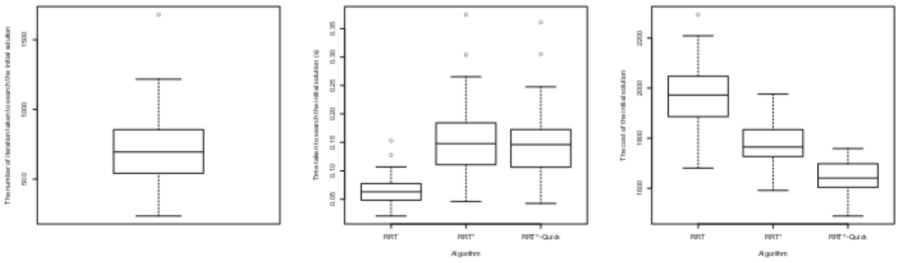
Table 1 shows the summary of the number of iterations to get the initial solution. It is also plotted as shown in Fig. 3.

Table 1. Statistical summary of the number of iterations needed to find the initial solution

Iterations to get the initial solution	
Mininum	236.0
1st Quartile	546.2
Median	695.0
Mean	715.4
3rd Quartile	852.8
Maximum	1680.0

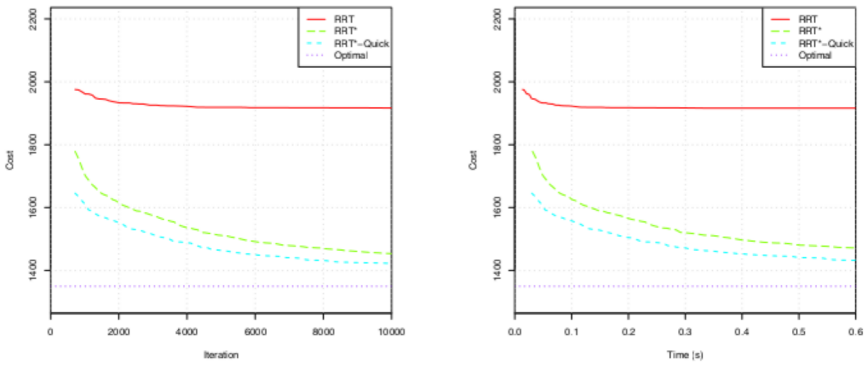
RRT found the initial solution most quickly, but the cost of the solution is the highest among the algorithms. RRT* and RRT*-Quick took more time to find the initial solution than RRT, but generated better solutions.

RRT*-Quick took slightly less time than RRT* and found much better solution than RRT*.



(a) Iteration taken to search the initial solution (b) Time taken to search the initial solution (c) Cost of the intial solution

Fig. 3. Statistical results of the initial solution

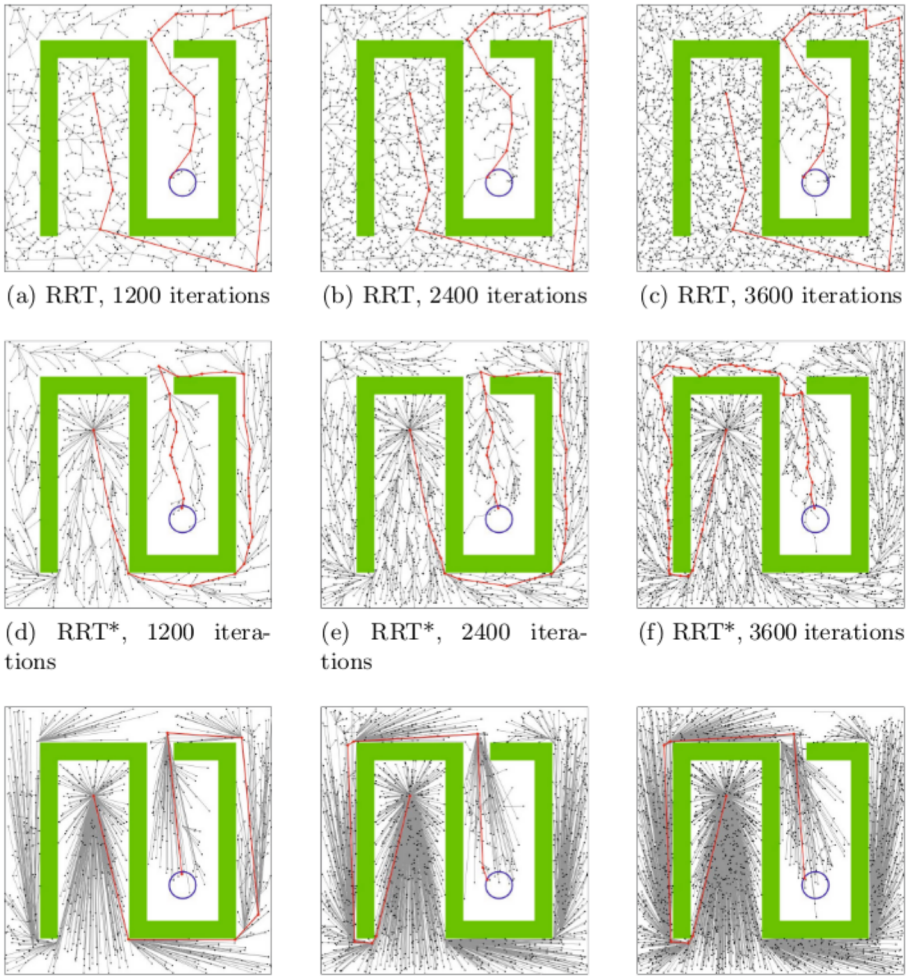


(a) Cost against iteration (b) Cost against time

Fig. 4. Costs are plotted against iteration/time

Convergence Rate. The cost of the best solution was measured on each iteration step and time step. Fig. 4 shows the cost against iterations and time, respectively, with the optimal cost. In each iteration, RRT*-Quick did more collision check, which consumes the most computation time, than RRT*. However, it turned out that it took only a half of time that RRT* took to get solutions with cost 1500.

The tree of each algorithm was drawn at every 1200 iterations as shown in Fig. 5. It is noted that the proposed algorithm converges more rapidly to the optimal and escaped from the local minima earlier than RRT*.



(a) RRT, 1200 iterations (b) RRT, 2400 iterations (c) RRT, 3600 iterations

(d) RRT*, 1200 iterations (e) RRT*, 2400 iterations (f) RRT*, 3600 iterations

(g) RRT*-Quick, 1200 iterations (h) RRT*-Quick, 2400 iterations (i) RRT*-Quick, 3600 iterations

Fig. 5. Generated tree

5 Conclusion

In this paper, RRT*-Quick algorithm that has a faster convergence rate than the conventional RRT* is proposed. The proposed algorithm enlarges the pool of effective parent candidates without increasing much computation time. Since it generates a motion plan with a lower cost, more samples are pruned and the computation time is reduced as a result. Simulations were performed to verify the performance of the proposed algorithm. It was demonstrated that the algorithm generated a shorter path compared to RRT and RRT*. Statistical analysis of the proposed algorithm and experiments with a robot in the real environment remain issues for further research.

Acknowledgement. This work was supported by the Technology Innovation Program, 10045252, Development of robot task intelligence technology, funded by the Ministry of Trade, Industry & Energy (MOTIE, Korea).

References

1. Bjornsson, Y., Enzenberger, M., Holte, R., Schaejfer, J., Yap, P.: Comparison of different grid abstractions for pathfinding on maps. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence, IJCAI 2003, pp. 1511–1512. Morgan Kaufmann Publishers Inc., San Francisco (2003)
2. Daniel, K., Nash, A., Koenig, S., Felner, A.: Theta*: Any-angle path planning on grids. Journal of Artificial Intelligence Research 39(1), 533–579 (2010)
3. Nash, A., Koenig, S., Likhachev, M.: Incremental phi*: Incremental any-angle path planning on grids. In: IJCAI, pp. 1824–1830 (2009)
4. Kavraki, L.E., Svestka, P., Latombe, J.C., Overmars, M.H.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Transactions on Robotics and Automation 12(4), 566–580 (1996)
5. LaValle, S.M., Kuffner, J.J.: Randomized kinodynamic planning. The International Journal of Robotics Research 20(5), 378–400 (2001)
6. Kuffner, J.J., LaValle, S.M.: Rrt-connect: An efficient approach to single-query path planning. In: Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2000, vol. 2, pp. 995–1001. IEEE (2000)
7. Karaman, S., Frazzoli, E.: Sampling-based algorithms for optimal motion planning. The International Journal of Robotics Research 30(7), 846–894 (2011)
8. Karaman, S., Frazzoli, E.: Optimal kinodynamic motion planning using incremental sampling-based methods. In: 2010 49th IEEE Conference on Decision and Control (CDC), pp. 7681–7687. IEEE (2010)
9. Karaman, S., Walter, M.R., Perez, A., Frazzoli, E., Teller, S.: Anytime motion planning using the rrt*. In: 2011 IEEE International Conference on Robotics and Automation (ICRA), pp. 1478–1483. IEEE (2011)