An Improved RRT Based Path Planning with Safe Navigation

Article · February 2014

DOI: 10.4028/www.scientific.net/AMM.494-495.1080

Yang Liu
Linköping University
91 PUBLICATIONS **886** CITATIONS

Applied Mechanics and Materials Vols. 494-495 (2014) pp 1080-1083
© (2014) Trans Tech Publications, Switzerland
doi:10.4028/www.scientific.net/AMM.494-495.1080

An improved RRT based path planning with safe navigation

Bin Feng[1, a], Yang Liu*[2, b]

[1]Information Networking Institute, Carnegie Mellon University, Pittsburgh, 15213, USA

[2]School of Electrical and Electronic Engineering, Hubei University of Technology, Wuhan, China

[a]email: BinFeng@cmu.edu,*Corresponding author [b]email: yli@puv.fi

Keywords: RoboCup; Small-Size League (SSL); Rapidly-exploring Random Trees (RRT); Safe navigation.

Abstract. Path planning has been a crucial problem in robotics research. Some algorithms, such as Probabilistic Roadmaps (PRM) and Rapidly-exploring Random Trees (RRT), have been proposed to tackle this problem. However, these algorithms have their own limitations when applying in a real-world domain, such as RoboCup Small-Size League (SSL) competition. This paper raises a novel improvement to the existing RRT algorithm to make it more applicable in real-world.

Introduction

Path planning, focusing on finding a viable path between the start location and goal location, has always been a crucial problem in the field of robotics. Many algorithms have been proposed to solve this issue, such as neural network, ant colony system algorithm [1], potential field algorithm, A* heuristic algorithm [2], genetic algorithm, etc. Rapidly exploring Random Trees (RRT) algorithm is a data structure and algorithm that is designed for efficiently searching non convex high dimensional spaces. RRT algorithm is constructed incrementally in a way that quickly reduces the expected distance of a randomly-chosen point of the tree. Due to its stochastic sampling-based feature, RRT algorithm is current state-of-the-art algorithm suited for path planning problems that involves obstacles and kinodynamic nonholonomic domains, such as RoboCup Small-Size League (SSL) [3] domain. However, under a realistic dynamic environment, such as SSL competition, RRT alone cannot guarantee a practical safety among multiple robot agents, due to the robots' kinetics feature. Therefore, improvements regarding on the dynamics of robot have been made to make RRT more practical in a real field environment.

Approach

A. RRT

Rapidly-exploring Random Tree (RRT) is an efficient algorithm and data structure to explore and make path planning in a non-convex and high-dimensional space without colliding with obstacles. RRT algorithm was first developed by Steven M. Lavalle and James Kuffner. Approaches based on Rapidly-exploring Random Trees (RRTs) rely heavily on nearest neighbors [4]. RRT algorithm has many practical applications in the field of robotics, gaming and aeronautics. It can be applied to high-dimensional space; however, we only apply it in a two-dimensional domain in the RoboCup SSL. The main advantage of RRT algorithm is that it can find a valid path in a complex domain in most of the times. And it also has the advantage of not requiring explicit enumeration of constraints, but allow trajectory-wise checking of possibly very complex constraints [5] [6]. However, the path found does not guarantee to be the optimal one because it depends on the probability of choosing path points [7] [8] [9]. In order to find a valid path, here is how RRT algorithm does:

1) In a general configuration space C, we first set the root of the tree to be the initial point q_{init}.
2) Then we generate a random point q_{rand} in the collision-free space C_{free}.
3) Then we choose the nearest vertex $q_{nearest}$ to the random point q_{rand} in the tree.
4) Then we expand a certain distance v from $q_{nearest}$ directly to q_{rand}, thus we create the point q_{new}.

5) If the q_{new} is not locate in the obstacle space C_{obs}, we add this point and the new edge into the tree. Otherwise, we go back to step 2.

6) Keep looping from step 2 to step 5 until q_{new} is close enough goal position q_{goal}.

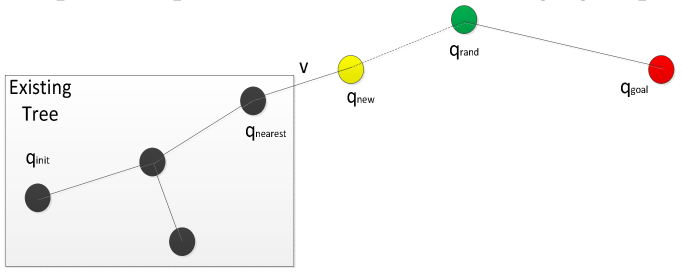


Fig.1. Illustration of RRT Algorithm

Figure 1 is a graphical illustration of how the RRT algorithm is executed. In the figure, the four black points in a rectangle box represent an existing RRT tree. The green point represents the generated random point in the collision-free space and the yellow point represents a new point generated in the direction from the nearest point in the RRT tree towards the random green point. Finally, the initial point and goal point are also represented in the figure [10].

Starting from a single initial point, the RRT tree will quickly expand the whole configuration space and find a valid path towards the goal position.

B. Robot model used in SSL

In the SSL competition, we used a holonomic robot model [11] which has the following assumptions [12].

1) The robot has a safety radius.
2) The robot has control over its acceleration within some set.
3) The robot has a maximum deceleration for emergency stop.
4) The robot has a maximum allowed velocity.

C. Safety-guaranteed RRT

Despite the fact that RRT algorithm has been an effective method when we are dealing with the path planning problem, it fits not that well when applying into a real field environment.

It has the following flaws when we apply the baseline RRT algorithm into RoboCup SSL competition.

1) In RRT, it presumes the moving object to be a particle without considering the physical feature of the moving object.

2) In RRT, it assumes the moving object is moving at a constant velocity, which is not feasible for a reality robot agent.

3) In RRT, it does not consider the dynamic feature of robot agent, thus the generated path cannot guarantee safety in the dynamic domain.

Based on the robot model mentioned in the previous part, the following dynamic formula can be listed:

$$v_f = v_i + at \tag{1}$$

where v_f is the final velocity, v_i is the initial velocity, a is the acceleration and t is the time of duration.

$$x_f = x_i + v_i t + \frac{1}{2}at^2 \tag{2}$$

where x_f is the final position, x_i is the initial position, v_i is the velocity, t is the time of duration, and a is the acceleration.

Obviously, the moving velocity of each robot agent cannot exceed the maximum allowed value V_{max}. Therefore, we have the formula:

$$\|v_i + Ca_i\| \leq V_{max} \tag{3}$$

where v_i is the velocity of i-th robot, a_i is the acceleration, and C is the time for a fixed control period.

Suppose the emergency stop maximum deceleration is D, then we have:

$$v_i + D \cdot t = 0 \tag{4}$$

where t is the time required to guarantee robot come to a stop under maximum deceleration and current velocity.

So, it is possible to calculate the safety distance S required to guarantee deceleration without hitting any obstacle.

$$S = \left\|\frac{v_i^2}{2 \cdot D}\right\| \tag{5}$$

In order to maintain safety, among moving robot with obstacles, the safety distance S is needed to be considered in the RRT algorithm.

Therefore, we have an improved version of RRT algorithm named safe_rrt which is described in pseudo-code:

```
procedure safe_rrt() : Path
    startNode ← new MyNode
    myTree ← new RRT Tree
    myTree.addNode(startNode, NULL)
    while not found do
        randomNode ← getRandomNode(myTree, env)
        nearestNode ← getNearestNode(myTree, random Node)
        if validSegment(nearestNode, randomNode) do
            newNode ← createNewNode(nearestNode, randomNode, maxLen)
            myTree.addNode(newNode, nearestNode)
            nearestNode ← newNode
            if hitTarget do
                found ← true
                return path
            end
        end
    end
```

The safe_rrt() procedure includes some sub-routines which we have applied safety check in it.

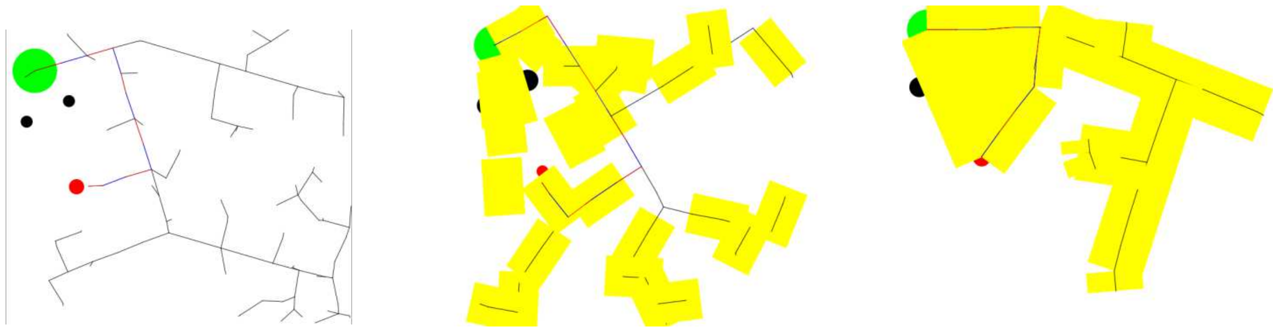
In getRandomNode() function, it returns a random node in the given environment domain without colliding with any obstacles. We construct a virtual rotated rectangle boundary based on the random node and the nearest node, then check it against each obstacle. If no collision happens, the function returns the random node.

In getNearestNode() function, we compare the randomized node against each node from the tree to calculate the distance between them. Therefore, we get the nearest node from the existing tree.

In validSegment() function, it calculates whether any collision may happen in the way between the nearest node and randomized node. It works by constructing a virtual rectangle which sweeps the rectangle area covered from the nearest node to the randomized node.

Experiments

We have implemented the novel algorithm with software simulation to test its performance.



(a) Demo 1 (b) Demo 2 (c) Demo 3

Fig.2. Demo results

Figure 2(a) demonstrates an example which employed the safety RRT algorithm. The green circle indicates the start position of the robot and its size. The red circle indicates the goal position for the robot, while two black circles indicate the obstacles existing in the field. By running the algorithm, it returns a valid highlight path for the robot navigation. Figure 2(b) shows how getRandomNode() function works in the algorithm. The virtual yellow rectangles confined by the safety distance are created to guarantee no collision happens to the new randomized node. Figure 2(c) shows how validSegment() works in the algorithm. The virtual yellow rectangles are confined by the areas which cover from the nearest node to the random node. Thus it is guaranteed that no collision may happen in the path.

Conclusion

RRT algorithm excels at exploring free space in larger environments and it is parallelizable. From the experiments and analysis, it can be verified that the new safety RRT algorithm is more applicable to the reality field environment such as RoboCup SSL competition because of the safety mechanism. The safety distance should be decided based on the physical configurations of the robot. In this paper, we use a rectangle-bound area to calculate the collision possibilities. For future work, a partial ellipse bound area can be used to make comparisons.

References

[1] N. Buniyamin, N. Sariff, W. A. J. Wan Ngah, and Z. Mohamad. Robot global path planning overview and a variation of ant colony system algorithm, International Journal of Mathematics and Computers in Simulation, 5(1), 2011.

[2] B. Akgun, M. Stilman. Sampling Heuristics for Optimal Motion Planning in High Dimensions, 2011.

[3] RoboCup SSL. http://robocupssl.cpe.ku.ac.th/

[4] Anna Yershova, Steven M. LaValle. Improving Motion Planning Algorithms by Efficient Nearest-Neighbor Searching, 2007.

[5] Yoshiaki Kuwata, Gaston A. Fiore, Justin Teo, Emilio Frazzoli, Jonathan P. How. Motion Planning for Urban Driving using RRT, Intelligent Robots and Systems, IEEE IROS, 2008.

[6] S. Karaman, E. Frazzoli. Sampling-based Algorithms for Optimal Motion Planning, 2011.

[7] Ahmad Abbadi, Radomil Matousek. RRTs Review and Statistical Analysis, 2012.

[8] Filipe Militao, Karl Naden, Bernardo Toninho. Improving RRT with Context Sensitivity, 2010.

[9] R. Alterovitz, S. Patil, A. Derbakova. Rapidly-exploring roadmaps: Weighing exploration vs. refinement in optimal motion planning, IEEE ICRA, 2011.

[10] J.J. Kuffner, S.M. LaValle. RRT-connect: An efficient approach to single-query path planning. IEEE Int. Conf. on Robotics and Automation, 2000.

[11] Botnia RoboCup team. http://robotics.puv.fi/

[12] James Robert Bruce. Real-Time Motion Planning and Safe Navigation in Dynamic Multi-Robot Environments, 2006.