PATH PLANNING IMPLEMENTATION USING MATLAB

A. Abbadi, R. Matousek

Brno University of Technology, Institute of Automation and Computer Science

Technicka 2896/2, 616 69 Brno, Czech Republic

Ahmad.Abbadi@mail.com, Matousek@fme.vutbr.cz

Abstract

Motion planning is essential part in robotics science; it is responsible for planning paths for robots. The robot follow this plan in order to reach the goal from its position, while avoid collision with obstacles. Planning modules could be configured to check the optimality, completeness, power saving, shortness of path, minimal number of turn, or the turn sharpness, etc. In this technical paper we review the probabilistically planner RRT (rapidly exploring random tree) as local/global planner and Cell Decomposition as global planner guide the RRT. We focus on implementation of these methods in Matlab environments.

1 Introduction

The Idea of making machines "Robot" which can serve the human being proposed many time over the ages, from old ancient Egyptian and Chinese, until the first programmable humanoid robot invented by "Al-Jazari" in middle Ages about 1206. And then To Leonardo Da Vinci who designed humanoid automaton. A lot of proposal humanoids and automatons machine appear later, by 1913 Henry Ford installs the world's first moving conveyor belt-based assembly line in his car factory, and then the automation and automated machine and robot started to be more and more complicated [1].

In robot developments, motion design was one of the most challenging tasks. Many methods were proposed and a lot of researches were done in this domain. From work-principle point of view, some of these methods are exact and based on static environment [2], like Cell-decomposition algorithms Fig. 2. Another type of these methods based on sampling-based algorithms, for example, potential field algorithm [3], or probabilistic algorithm e.g. PRM (probabilistic Road Map) [4] and RRT (rapidly exploring random tree) [5].

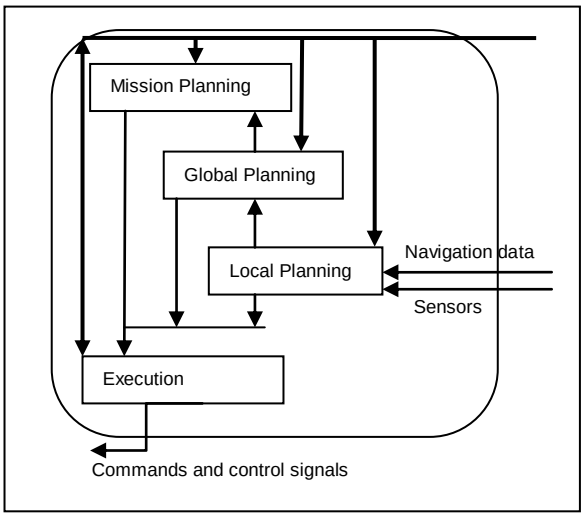


Figure 1: hybrid planner

In other hand, from generated-solution point of view we can divide the algorithm into multilevel (local, global or mission) levels or planners levels Fig. 1; which mean in local planner the algorithm produce solution locally based on surrounding space. Most of these algorithms are easy to implement and required low resource consuming like Bugs [6], VFH [7], Limited RRT, the main advantage is the tolerant to environment changing, and main drawback of this algorithm is local minima. In global

planner the algorithm produce full path form initial position to goal position. A lot of these algorithms require middle to high resource using. The main advantage is to avoid local minima but they have less tolerance to environment change. Mission planning is the stage where the algorithm has more than one goals and the planner in this level tries to optimize or find a way to satisfy all goals.

Recently many robots use hybrid planner (Fig. 1) to overcome the drawback of each type of this algorithms. In this article we will review shortly RRT and Cell-decomposition algorithms as an example of hybrid planner (local, global) [8] and focus on how we use Matlab to implement and test the proposed planner.

2 RRT

RRTs algorithm is probabilistic algorithm used for exploring the spaces rapidly and planning paths. Originally RRTs is a single query planner [5]. RRTs work efficiently for problems with high dimensions, nonlinear, dynamics or differential constraints. It is also good in discrete or continuous spaces. In RRTs we do not need to make any pre-computing for the configuration space. The tree is grown by connecting random samples in free space. A new edge will not be added to the tree if there is any collision with obstacles. Also RRTs is proofed to be probabilistically complete [9].

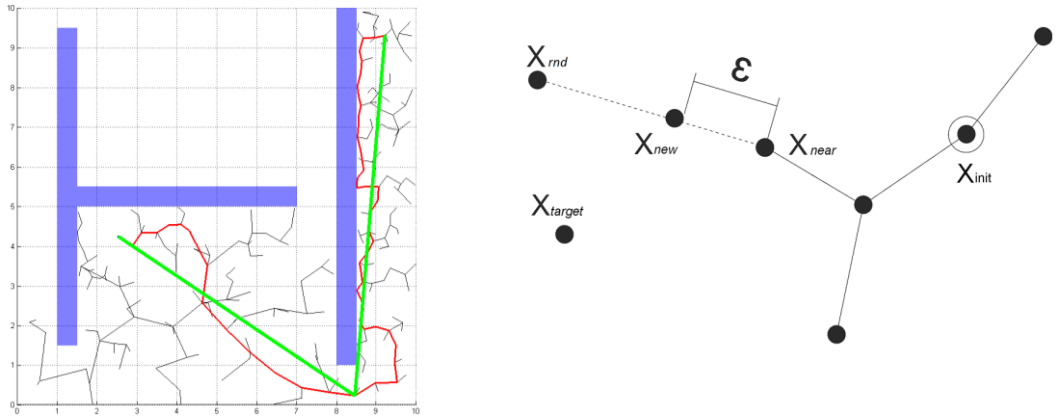


Figure 2: RRT principle.

The principle of RRT is shown in Fig. 2, we can summarize it as few steps, 1: Define the start point X_{init} and goal point X_{target}. 2: Make X_{init} as root of the tree. 3: choose random point of the space X_{rnd}. 4: Try to grow a branch [X_{near}, X_{new}] by length ε, where [X_{near}, X_{new}] is a segment on the line between the nearest point of tree X_{near} to X_{rnd}. 5: If no obstacle crossed by new branch add X_{new} as new point on tree. 6: Check if we reach X_{target}. 7: Repeat step 3 to 6 until we reach the goal point then return the path. We implement this algorithm in Matlab as shown in Fig. 3.

The main drawbacks in basic RRT are 1- It is not optimal, because it depends on probability and a random choice of next configuration. 2- A tree contains a great number of redundant nodes. 3- It has some difficulty to explore small areas and find the way throw narrow passages, and that come from the principle of RRT which depend on probability of choose a point in the space, which mean the small or narrow areas will have small possibility to choose a point form them. Moreover, the probability to connect the chosen configurations with a tree without collision is also small. 4- The produced path usually very tortuous and has a lot of turn and sharp angles.

Many improvements were done to develop the basic RRT [10]. One of them is to grow two or more trees from initial position, goal position, or any other points. And that will speed up the exploration speed, because instead of searching for single goal point we search for any connection between trees-points which has much more higher possibility. The connection between initial and goal trees will turn the problem to find path between two points on graph -tree could be represented by graph of nodes(points) and edges (segment between points)-.

```
%%RRT class
classdef RRTClass<handle & configurationSpace
   properties
     cords; parent; startPos,goalPos;
     extensionStep= E; rrtType='basic_RRT';
```

```
    bias.enable=1; bias.type=['biasToGoal', 'biasToTreePoints',…]; % bias to goal, other
trees, specific points,…
    bias.rangeVal=[0.05,0.07,…];%biasToGoal in range 0-0.05=5%,biasToTreePoints in range 0.05-
0.07=2% the rest is normal random point selection
  end
  methods
    function rrt=RRTClass(initialValues) …...
    function [objective, tElapsed]=rrtPlanner(rrt,MaxIteration,drawType)
      tRRTStart=tic;
      for iter=1:rrt.MaxIteration
          [objective]=grawTree(rrt);
          rrt.draw(iter ,drawType);% drawType :realtime draw, draw when success, don't draw
          If objective
             break;
          end
      end
      tElapsed=toc(tRRTStart);
    end
    function [objective]=grawTree(rrt)
     objective=0;
     [randomConfiguration]= rrt@configurationSpace.getRandomPoint(rrt.bias);
     [nearestConfiguration]=rrt.getNearestPoint(randomConfiguration);
     [newConfiguration]=rrt.branching(nearestConfiguration,randomConfiguration);
     IsCollid=rrt@configurationSpace.checkCollision(nearestConfiguration,newConfiguration);
     If IsCollid ; return ; end;
     rrt.addToTree(nearestConfiguration, newConfiguration);
     [objective]=rrt.checkGoal();
    End ….
```

Figure 3: selected pieces of RRT code in Matlab

Other improvements based on random point choosing strategies and sampling strategies. We can sample the space uniformly and choose one of these samples or use continuance or discrete distribution functions. Some other sampling strategy based on obstacle driven; which mean when a collision with obstacle occur the new samples are created and used to guide the planner. Another approach to develop RRT efficiency is sampling strategy which depends on kd-tree, voronoi [11] or filling-space algorithms [12]. In our implementation we use mainly uniformly distribution as shown in Fig. 4 "getBiasPoint" function; where we take a random point from work space. Another technique used in RRT planner is to bias to point/points with some probability, e.g. bias to goal point, to other trees-points, to point around the goal, old successful path points, points from important region or areas which taken from available knowledge about working environment. We implement the function "getBiasPoint" to give the user freedom to specify the bias methods and the probability value to these biases.

```
%%%%%%%% configurationSpace Class(CSpace) Methods%%%%%%%
function newPnt= getRandomPoint(CSpace ,bias)
    newPnt=[];
    if bias.enable
        newPnt = CSpace.getBiasPoint(bias);
    end
    if empty(newPnt)
           range=abs(CSpace.dimensions(:,2)- CSpace.dimensions(:,1));
           for i=1:size(CSpace.dimensions,1)
              newPnt (1,i)= CSpace.dimensions(i,1)+range(i)*rand;
           end
    end
end
function newPoint=getBiasPoint(CSpace ,bias)
    randVal=rand;
    methodIndex=find(randVal <= bias.rangeVal,1,'first');
    biasMethod= bias.type(methodIndex);
    switch biasMethod
           case 'biasToGoal'
                newPnt =goalPos;
           case 'biasToTreePoints' % bias to one point in other trees
        …..
           case 'biasToGivenPoints'
                …..
                randVal=randperm(size(CSpace.biasGivenPoints,1));
                newPnt = CSpace.biasGivenPoints(randVal(1),:); …..
    end
end
…..
```

Figure 4: selected pieces of getRandomPoint function code

3 Cells decomposition

Cell decomposition is one of the first applicable solutions for path planning [13] (Fig. 5). The algorithm aims to find free areas (cells not occupied by obstacles) in the configuration space, and build a graph of adjacency for these cells (Fig. 6). Finding free cells is not easy task especially in high dimensions. Many approximation solutions were proposed to increase the efficiency of this algorithm. One of these solutions in (2d) is to use a sweep line on x-axis (Fig. 5), then find the intersections with obstacles-edges, in order to generate the cells. The algorithm output is horizontal free areas (see Fig. 2-left). But for better use as spatial global planner we add another step to divide the cells horizontally Fig. 5-right. Base on generated cells we build graph of adjacency; nodes are the cells itself and edges are the connection between the cells. By this way the problem of navigation and path planning turned into graph search problem. For example, when we want to plan a path between two positions, we find cells which contain the positions, and then we search over the graph for a path (Fig. 7).

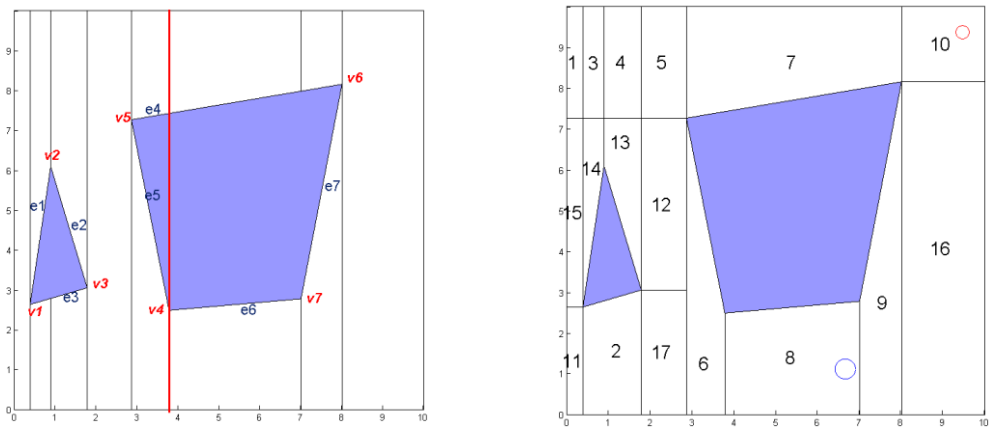


Figure 5: space cell decomposition

The drawbacks of Cell-decomposition algorithm are the number of generated cells on high dimension, computation cost, and the algorithm does not work probably in dynamic environments. The idea of dividing the space into manageable sections is presented in many researches [14] using different techniques. In [2] the authors review some cells decomposition methods i.e. approximate cell decomposition which based on drawing a mesh grid in the space, then exclude the cells on obstacle areas. This method is efficient but it generates large number of cells, and it is not complete in all cases. The next improvement for CD was by dividing the space quarterly, and tests every cell, if it has obstacle then we re-divide it quarterly again. This method is easy to implement. And we can control the minimum size of generated cell.

From Matlab point of view the graph of adjacency is generated based on sweep line. We use Bioinformatics graph function to deal with generated graph. The first function "*graphshortestpath*" of this toolbox is to search over the graph for shortest path between initial and goal position. This function could be configure to use 'Bellman-Ford', 'BFS', 'Acyclic', or 'Dijkstra' algorithms, we use in our implementation 'Dijkstra' algorithm which is also the default of this function. The other useful function in this task is "graphallshortestpaths" which give all available shortest paths. For visualization of graph we use "biograph" function which create graph object, and then we draw it using "view" function Fig. 6 shows piece of code which we use for search and visualize the graph of adjacency, the result of this code could be seen in Fig. 7-left.

```
%prepare Undirected Graph
wieght=1;
DG=sparse(graph.edges(:,1),graph.edges(:,2),wieght);
UG=tril(DG+DG');
% Graph search functions in Graph Theory, Bioinformatics Toolbox
[dist,path] = graphshortestpath(graph,InitialPosition,GoalPosition,'directed', false);
%draw graph of adjacency
h=view(biograph(UG,cellstr(num2str([1:size(UG,1)]')),'ShowArrows','off','ShowWeights','on'));
…..
```

Figure 6: search and draw graph, based on bioinformatics toolbox function

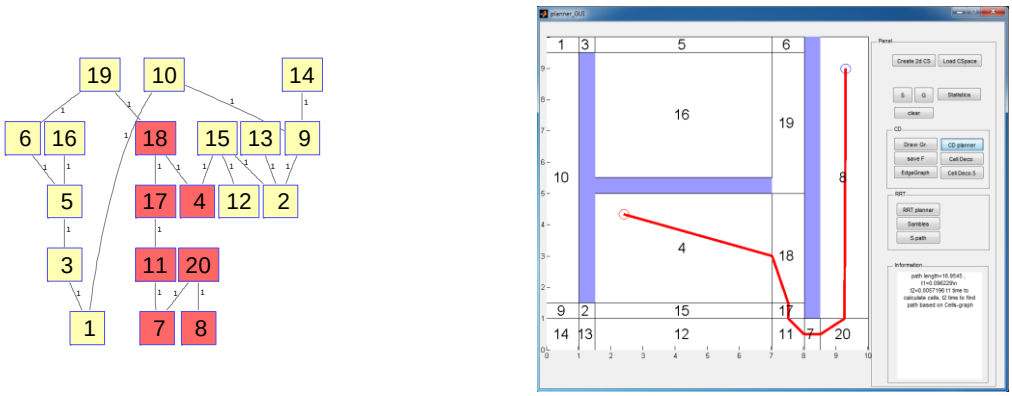
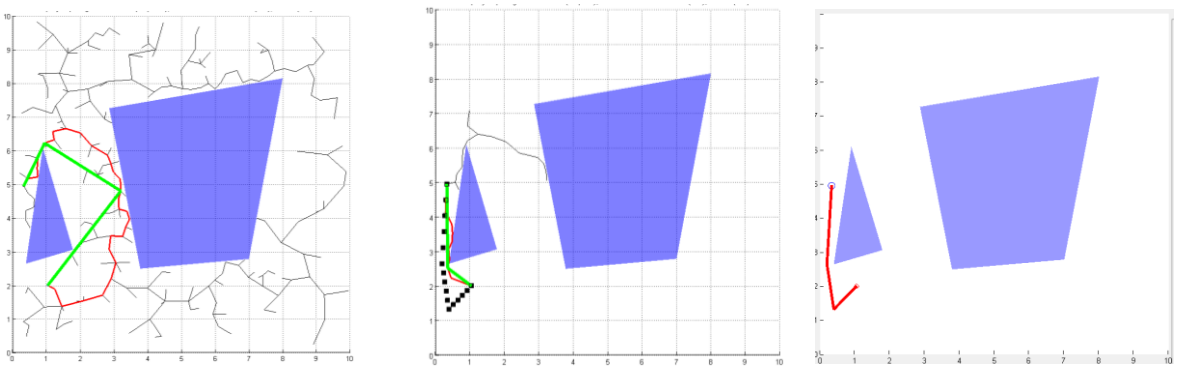


Figure 7: Cell-decomposition planner GUI and generated graph.

4 Results

In our work we integrate the results from Cell-decomposition and RRT algorithms in order to overcome some drawback of these two methods. Fig. 8 show the difference between RRT planner tree in normal work Fig. 8-a, and RRT planner when it use Cell-decomposition path points (Fig. 8-c) as bias-points Fig. 8-b. The planner in Fig. 8-a has to explore wide areas before it find the goal, while in second figure we can see the bias-point and the reduction of RRT tree. For implementing these algorithms, calculate results and make some statistical analysis we use Matlab, and we use Matlab GUI as user interface to enter desire value or options, also we generate graphs and apply searches and operation on them using some functions from Bioinformatics Toolbox.



a b c

Figure 8: RRT planner without /with bias to Cell-decomposition points and Cell-decomposition path

We repeat the test 100 times for RRTs on simple scenario Fig. 8, and take the mean of the results for successful tries to reach the goal. In each time we setup RRTs planner to extending branch by length of $E = 0.5$. And we consider that RRT fails to reach the goal after 3000 tries of branching, we use Intel Xeon(R) PC with CPU of 2.67 GHz, and 6 GB of memory, and Windows 7 64-bit. We implement the algorithm in MATLAB R2010b environment. The CD planner use Dijkstra algorithm for search on the graph. Dijkstra in this case has $O(log(N)*E)$ time complexity. Where N is the nodes number and E is the edges number in the graph. Table 1 show the difference between RRT planner using CD path as bias points and without using bias. The results show better timing and better length of generated path.

Table 1: COMPARISON OF MEAN VALUES OF RRT WITH/WITHOUT BIAS TO CELL-DECOMPOSITION POINTS

	Mean Time	*Mean Path Length*
RRT planner without bias	0.0427	15.590
RRT planner with bias	0.0307	9.8900

5 Conclusion

In this paper we implement and test RRT planners in simple configuration space andtest Cell-decomposition which generates graph of adjacency. The results show that using RRT planner will be more efficient when we integrate it with Cell-Decomposition path-points; the integration is done by passing path-points as external bias points to RRT planner. The idea of using spatial information given by cell-decomposition and RRTs planner gives a good result but it need more tests and enhancement of efficiency for algorithms. The future work will focus on using available information to speedup complex motion planning process for robots in uncertainty and dynamic environments.

References

[1] K. F. Uyanik, "*Social Robot Partners: Still Sci-fi?*"

[2] N. H. Sleumer and N. Tschichold-Gürman, *Exact Cell Decomposition of Arrangements used for Path Planning in Robotics*. 1999.

[3] O. Khatib, "*Real-time obstacle avoidance for manipulators and mobile robots*," in *1985 IEEE International Conference on Robotics and Automation. Proceedings*, 1985, vol. 2, pp. 500–505.

[4] L. Kavraki, M. N. Kolountzakis, J.-C. Latombe, and J.-C. Latombe, "*Analysis of Probabilistic Roadmaps for Path Planning*," *IEEE Trans. Robot. Autom.*, vol. 14, no. 1, pp. 166–171, 1998.

[5] S. M. Lavalle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning," 1998.

[6] Choset, Howie, Lynch, Kevin M., and Hutchinson, Seth, *Principles of Robot Motion : Theory, Algorithms, and Implementation*. MIT Press, 2005.

[7] J. Borenstein and Y. Koren, "The vector field histogram-fast obstacle avoidance for mobile robots," *IEEE Trans. Robot. Autom.*, vol. 7, no. 3, pp. 278–288, Jun. 1991.

[8] Ahmad Abbadi, Radomil Matousek, Pavel Osmera, and Lukas Knispel, "*SPATIAL GUIDANCE TO RRT PLANNER USING CELL-DECOMPOSITION ALGORITHM*," presented at the 20th International Conference on Soft Computing, MENDEL 2014, 2014.

[9] D. Berenson and S. S. Srinivasa, "*Probabilistically complete planning with end-effector pose constraints*," in *2010 IEEE International Conference on Robotics and Automation (ICRA)*, 2010, pp. 2724–2730.

[10] A. Abbadi and R. Matousek, "*RRTs Review and Statistical Analysis*," *Int. J. Math. Comput. Simul.*, vol. 6, no. 1, 2012.

[11] C. Urmson and R. Simmons, "*Approaches for heuristically biasing RRT growth*," in *2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings*, 2003, vol. 2, pp. 1178–1183 vol.2.

[12] J. J. Kuffner and S. M. LaValle, "*Space-filling trees: A new perspective on incremental search for motion planning*," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011, pp. 2199–2206.

[13] S. M. LaValle, *Planning algorithms*. Cambridge ; New York: Cambridge University Press, 2006.

[14] J.-C. Latombe, *Robot motion planning*. Boston: Kluwer Academic Publishers, 1991.