

# **Path Planning in Non-Stationary Environment using RRTs**

*A Project Report*

*submitted by*

**SAI CHAITANYA MANCHIKATLA**

*in partial fulfilment of the requirements  
for the award of the degree of*

**MASTER OF TECHNOLOGY**

*under the guidance of*  
**Dr. Ravindran Balaraman**



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING  
INDIAN INSTITUTE OF TECHNOLOGY, MADRAS.**

**May 2015**

## THESIS CERTIFICATE

This is to certify that the thesis entitled **Path Planning in Non-Stationary Environment using RRTs**, submitted by **Sai Chaitanya Manchikatla**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Technology**, is a bona fide record of the research work carried out by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

**Ravindran Balaraman**  
Research Guide  
Assistant Professor  
Dept. of Computer Science and Engineering  
IIT-Madras, 600 036

Place: Chennai

Date:

## **ACKNOWLEDGEMENTS**

I would like to thank my friends, family for supporting me. I would also like to extend my gratitude to Ravi sir for his support and encouragement.

## ABSTRACT

Path planning is a widely researched area with applications in diverse fields. In its most plain setting, it involves finding a trajectory through a state space connecting free configurations and avoiding collisions. There are many proposed solutions that find a path if it exists. First we will look at various approaches attempted at tackling path planning under kino-dynamic constraints and other settings like obstacles with or without predetermined path. We will then present a popular and widely-used sampling based technique known as Rapidly exploring random trees (RRT). There have been many improvements to the vanilla RRT algorithm like growing trees from start and goal states, adding a bias towards goal, etc. They work well in some domains. The performance of RRTs majorly depend on the choice of distance metric we use. Standard metrics like Euclidean would not work in high dimensional domains and choosing a good distance metric is critical to make RRTs find a good solution.

Learning domain dependant metrics through repeated experience is looked in Reinforcement Learning (RL). We first present a framework in RL called Markov Decision Process (MDP). We then introduce RRTPI algorithm which learns the geodesic distance metric used in RRTs by posing the problem as a MDP. We introduce DYNA, a framework for integrating Planning, Acting, Model-Learning and Direct RL updates. We motivate how it can be used for solving the issue of path planning in non-stationary environment. We present DYNA-RRTPI algorithm to integrate Planning and Model-Learning into the original RRTPI algorithm and explain the use of it in continuous state domain. We then present our results in a continuous domain with kino-dynamic constraints.

# TABLE OF CONTENTS

|   |           |
|---|-----------|
| <b>ACKNOWLEDGEMENTS</b>                     | <b>i</b>  |
| <b>ABSTRACT</b>                             | <b>ii</b> |
| <b>LIST OF TABLES</b>                       | <b>v</b>  |
| <b>LIST OF FIGURES</b>                      | <b>vi</b> |
| <b>1 Introduction</b>                       | <b>1</b>  |
| <b>2 Path Planning</b>                      | <b>3</b>  |
| 2.1 Configuration space . . . . .           | 3         |
| 2.2 The Problem . . . . .                   | 4         |
| 2.3 Related Work . . . . .                  | 4         |
| 2.3.1 Search based algorithms . . . . .     | 4         |
| 2.3.2 Artificial Potential Fields . . . . . | 5         |
| 2.3.3 Sampling based planners . . . . .     | 6         |
| <b>3 Rapidly Exploring Random Trees</b>     | <b>8</b>  |
| 3.1 Algorithm . . . . .                     | 9         |
| 3.2 Improvements . . . . .                  | 10        |
| 3.2.1 Optimality . . . . .                  | 12        |
| 3.2.2 Issues . . . . .                      | 13        |
| <b>4 Planning in Dynamic Environment</b>    | <b>15</b> |
| 4.1 Predicting obstacles . . . . .          | 16        |
| 4.2 Replanning . . . . .                    | 17        |
| <b>5 RRTPI</b>                              | <b>19</b> |
| 5.0.1 MDP . . . . .                         | 19        |



## **LIST OF TABLES**

## LIST OF FIGURES

|     |   |    |
|-----|---|----|
| 3.1 | Process of building the RRT and extending the current tree to $x_r$ and . . . . .   | 9  |
| 3.2 | A snapshot of growth of RRT starting at [50;50] and goal at [95;95] the first column is at 200 samples and second column at 2000 samples. The first row is taken when sampling randomly , second row at 10 percentage bias to goal and third row at 20 percentage bias towards goal . . . . . | 11 |



# CHAPTER 1

## Introduction

Path planning has become a central problem in Robotics since the advent of Robots in Industry floors, daily life both directly and indirectly like robotic surgeries, autonomous self-driving cars, search and rescue operations involving unmanned drones, guided missiles etc. With increasing automation in previously what were completely human operated jobs and Robots moving autonomously in land, air and water, path planning is very much essential for it to complete its mission. Even though robots vary in sensors, actuators, applications and any other physical differences, the problem of navigation in a complex environment either known before hand or unknown is essential to all applications.

Figures; UAV, Clash of Clans and Arm

In its most generic form, Path Planning is the task of finding a valid path in a given configuration space. For example, finding out a sequence of steps to be taken to move a robot from initial point to a goal position in a maze is a standard problem looked in robotics. Other examples include finding out how to move each individual motors in a robotic arm to take the gripper from initial position to a required point to grip the object. The problem is not just specific to Robotics but is applicable in other domains like Computational Biology, Digital verification. In CAD software where one has to place the components on a chip in a way so that required connections between the components can be made, path planning is used. In animation and gaming industry, various efficient path planning techniques have been implemented to make the games more realistic, and at the same time finding out a path as quick as possible to make the game run on a mobile device in real time.

For example let's consider the famous mobile game clash of clans which involves a player's army trying to take down others village. It has various static obstacles like walls, buildings and other dynamic obstacles like troops to defend village as well. A deployed unit should figure out a path to the building to be attacked navigating through these static and dynamic obstacles. This is a classic problem that occurs in many game settings. Upon close observation one can notice the environment is discretized into grid of cells which are either free to navigate in or consist any obstacles. Each grid cell is connected to 8 surrounding cells. Such problems are relatively easier to tackle. When the search space is smaller and discrete, one can apply Dijkstras shortest path algorithm[put ref] or A\*[put ref] algorithm to find the

path from initial to final points avoiding collisions with obstacles.

A relatively challenging problem is when the state space is not discrete in the form of grid. For example in [figure 2], one can see a robotic arm with many joints and gripper at the end. in 3-dimensional continuous state space. To take the gripper from one position to desired position, all the joints have to be rotated appropriately. Each joint has constraints on angle it be rotated. Thus figuring out a sequence of states of angles of each joint to be followed in the *configuration space* to reach the final position avoiding collisions with obstacles in the environment and itself forms a path planning problem. Here one can notice that states are not merely the coordinates in euclidean space.

The path planning problem has been studied for decades now. One can see Choset [et al. [41], De Berg et al. [26], Latombe [96] and LaValle [97] ] for a detailed introduction to path planning and various attempts at solving the problem. It can be seen that most of the work has been done aimed at stationary or static environments. In such environments, there would be no obstacles or obstacles would be static if they are present. Also one can be sure that the environment doesnot change over time. Hence one has the advantage of doing a detailed plan once and using it for any future planning requirements. But in most of the real world scenarios it is rarely the case. Environment changes and obstacles are not fixed. We try and attempt solving the non-stationary environment case where the world may change over time. We do not treat the obstacles seperately but we model the obstacles also into the environment. For instance, consider a robot doing mapping task in a unknown building. It maps the environment when it enters a room and plans a way to get out of the room. The next time when the robot enters the room some objects might have been moved. Thus earlier plans are valid but necessary modifications have to be made to use them. We deal with such scenarios in which environments are non-stationary but not necessarily moving obstacles whose path can be calculated. Also we try to look at the kinematic and dynamic constraints on the movement of the robot and try and formulate an algorithm that would obey them.

In the next chapter we define the problem of path planning formally and list out different variations of the problem and some approaches that have been tried before. In the further chapters we would detail a sampling based approach called Rapidly Exploring Random Trees(RRTs) followed by our method that uses Reinforcement Learning (RL) techniques for solving the problem optimally. We conclude with experiments in various domains and future work.

## CHAPTER 2

### Path Planning

One of the earliest breakthroughs in motion planning was the introduction of the concept of configuration space (1979 lozano perez and wesley). It made possible to represent the robot by a single point. Now finding a solution became nothing but finding a path connecting those points.

#### 2.1 Configuration space

It allows us to specify the problem. Configuration of a robot in general includes the geometry of the robot, the environment setting and also the geometry of the obstacles. It describes the degrees of freedom the robot has. Configuration space is the set of all such possible configurations of the whole system which includes agent( or robot), the environment and obstacles. It is generally parametrized. For example, the configuration of a robot translating in a 2-D space can be described in terms of 2 parameters  $x, y$ . If the robot is allowed to rotate then an additional parameter is required to describe the angle or orientation. The minimal number of such parameter that are required to describe a configuration of the robot is called *degrees of freedom* of the robot. In general, there must be four components in configuration description.

- Description of the geometry of the moving agent or robot.
- Description of the geometry of the workspace.
- Description of robot's degrees of freedom.
- A start and goal configuration( can be set of configurations also) for the robot.

We denote configuration space by  $C$  and each parameter often represents a degree of freedom. The obstacles, static or moving, does not allow the robot to reach certain configurations. Such configurations which has collisions form forbidden configurations. In case of robotic arm, such forbidden configurations arise not only because of obstacles in the environment but also itself as each joint has limits on the rotating capacity. Let us denote them by  $C_{forbidden}$  or  $C_{obs}$  and rest of the states that can be reached by the robot as  $C_{free}$ .

## 2.2 The Problem

A path is defined as a continuous function with range in the configuration space  $C$  as follows

$$\pi : [0, L] \rightarrow C$$

where  $L$  is the length of the path. The problem of motion planning is to find a path in  $C_{free}$  configuration space between start and goal configurations.  $\pi(0) = s \in C_{free}$  and  $\pi(L) = g \in C_{free}$  and  $\forall i \in (0, L) \pi(i) \in C_{free}$ . A *complete* path planning algorithm should return the path in finite time if it exists and if such a path doesn't exist it should return failure. Also more often than not one would define a cost measure on the path which can be length of the path or time taken or domain specific, which is used to rank paths found. An optimal path is one with the least cost. The problem of finding the path is a PSPACE-hard [ Reif 1979 reference ], where complete planning algorithms ( put reference Lozano-Perez and Wesley, 1979; Schwartz and Sharir, 1983; Canny, 1988) exist but are not practically feasible due to their time complexity.

## 2.3 Related Work

The path planning problem can be categorised into static and dynamic environments. Further classification can be done based on whether we are operating in known or unknown environment.

### 2.3.1 Search based algorithms

For simple cases in a known environment, search based planning algorithms which discretise the configuration space to grid cells and find the path from start cell to goal cell work well. A lot of algorithms have been proposed in this genre including Dijkstra's algorithm ( put the fucking reference ) which is a breadth first search and with introduction of heuristics in algorithms like A\* the search space is reduced. These algorithms were even modified to suit dynamic environments D\* unlike A\* which was a static algorithm needed to be re-run again once environment changes. Also derivatives of D\* exist which includes Focused D\* which addressed the time complexity issues of D\* and quad tree D\* which reduced the space complexity. Other modern approaches like Anytime A\*, D\*-lite are proposed which reduces the search and time complexity even further.

## Issues

But the main limitation of this class of algorithms is that grid is a fixed topology whose resolution must be specified and paths are only optimal at level of the resolution of the grid. There is always a trade-off between the quality of the path generated and the computational power available. Hence to solve the problems efficiently in known environments other approaches involving combinatorially computing the explicit paths like Visibility Graph method [ insert reference] and Artificial Potential Field methods[ insert reference ] were introduced.

### 2.3.2 Artificial Potential Fields

Artificial Potential Field method is a practical approach( ge and cui reference 2002 ) was widely used on many real systems as it relaxed the completeness to *resolution completeness* which is returning a path if it exists provided that the resolution parameter of the algorithm is set correctly. A potential field is cast in the configuration space where destination applies the attractive force and obstacles cast repulsive force and the robot follows the steepest descent in the resulting potential field to reach the destination. But this faces issue that robot might get stuck in the local optima. There were proposed solutions [reference from mid term 6] to tackle this but it does not apply to all problems in general. ARF is particularly suited for applications where translation in any direction is major degree of freedom as one calculates the potential of a point based on the distance from the source in the workspace or configuration space.

## Issues

one can notice that it requires explicit representation of the obstacles in the configuration space which is not always the case in many real world tasks. Also even if they are explicitly mentioned, presence of large number of obstacles would give rise to lot of computational overhead as solution is build using obstacles representation directly. Hence only low dimensional C-spaces and simplified geometries have been solved efficiently using these methods. Avoiding such explicit representation was done in the next set of algorithms known as sampling based algorithms. These algorithms were ideally suited for unknown or partially known environments as they do not as for explicit representation of the obstacles in the configuration space.

### 2.3.3 Sampling based planners

In the sampling based motion planners, the configurations in  $C_{obs}$  are determined only by sampling C-space using *collision detectors* which would tell us whether a given configuration is a part of  $C_{free}$  or  $C_{obs}$ . Thus the modern collision detection algorithms keep track of simple metrics like distance between robot and obstacles in the environment to specify that. It makes even complicated models to be specified and solved efficiently using limited information about the configuration space. There is no need to enumerate all possible contact cases of our robot and obstacles nor compute the contact cases to solve for a trajectory. Also it makes the problem simpler as the collision detection can be treated as separate module. The simplicity and generality made the sampling based planners to be successfully applicable to wide range of motion planning problems even with high dimensional configuration spaces. For a full historical evolution of sampling based planners one may look into [reference current issues in sampling based planners]. One of the highly successful sampling based planners was Probabilistic Road Map method( insert reference ).

#### Probabilistic RoadMap

The most basic implementation of Probabilistic RoadMap is very simple and works in two phases. In the preprocessing phase the algorithm builds a road map( a graph  $G(V,E)$ ) by attempting to connect points sampled from  $C_{free}$  which it uses to query in the query phase. Algorithm begins with an empty graph. In preprocessing phase it samples a point  $x_{rand}$  from the state space and if it falls in  $C_{free}$  it is added to vertex set  $V$ . Then it attempts to connect all points that are in  $r$ -radius from the  $x_{rand}$  in the order of increasing distance from  $x_{rand}$  i.e nearest first using a simple local planner. The local planner checks if a path is possible by checking collisions with any obstacles. If there are no collisions then the points are added to the graph by joining edges between the point and  $x_{rand}$ . For sake of simplicity the connections are only added if the randomly sampled point is not connected to any connected components of the new point. This prevents any formation of cycles, resulting in a forest ( set of trees). There are other simpler modifications where initially the algorithm samples  $n$  points in the space and further it form trees among those  $n$  points only. In other work ( lavalley 2006), the selection of vertices has been modified from  $r$ -radius disc to  $k$ -nearest neighbours of the randomly sampled point. This forms a  $k$ -nearest graph. There were other modifications involving setting bounds on the number of vertices picked from the  $r$ -radius disc. Other ways of modifying the algorithm is to have a variable radius disc from which one tries to join points to the randomly sampled point. These versions are called  $k$ -Nearest PRM, bounded degree PRM,

and variable radius PRM respectively. If a roadmap was constructed for a particular configuration, it can be used to query for paths between any pairs of configurations in the environment in the Query phase. To implement this, we connect the two query configurations to the existing roadmap and treat them like new nodes inserted into the roadmap during the previously described construction phase. If it succeeds a path is searched for in the roadmap. It is usually the case that paths generated from Probabilistic Roadmap technique have to undergo some post processing to give smooth paths. Other variants of PRMs differ in the choice when joining two configurations using the local planner. In the simplest case we used a straight line. Also how we determine the neighbors to which connection is attempted can be varied. Local planners can be made to try complicated connections and ways of choosing the points to attempt connections to. One can choose all points in the neighbouring set but that takes time. One can choose to attempt to only k-nearest neighbours within a bounded distance  $d_{max}$ . Such things take time for such hefty computations. One cannot afford time on such steps. Thus one has to be careful in making a tradeoff between the real-time application of the planner vs the complexity we need. PRMs are not complete. That is, if a valid path does not exist between the start and end points in  $C_{free}$ , they would not be able to notify that. However, it is to be noted that PRMs are probabilistically complete. If there exists a path between start and goal configurations in  $C_{free}$ , then the probability that PRM-roadmap contains a solution tends to 1 as the roadmap creation time tends to infinity [insert reference 4]. Creating a roadmap takes lot of time and once it is created all future queries can make use of this preprocessing step to fetch results faster. Such problems are called multiple query problems. In contrast to this, when there is only one valid query to be made in a given setting it is called single-shot method. Frazzoli et al (Sampling-based Algorithms for Optimal Motion Planning) proposed PRM\*, an optimal version of the PRM where they limit the number of points in the neighbourhood to extend to either by setting bound on k in the k-nn approach or by setting the radius of the disc.

## Issues

This is of importance in fast and changing environments where the roadmap created might not be valid for future queries. Hence in dynamic environments, various techniques like single-query PRMs were introduced but the most successful and widely used algorithms for their simplicity are Rapidly Exploring Random Trees. We discuss RRTs and their modifications in the next section.

## CHAPTER 3

### Rapidly Exploring Random Trees

RRT is a single query sampling based algorithm. In the basic version the algorithm incrementally builds a tree of feasible trajectories with the start point as root. The algorithm initially samples a point  $x_{rand}$  randomly in the space. It then tries to connect the point to the already existing tree by trying to connect to the *nearest* point  $x_{near}$  in the tree using a local planner. The local planner connects them and adds an edge to the tree if the path is not colliding with any obstacle. To minimize the collisions the algorithm instead of joining  $x_{rand}$ , tries to move from  $x_{near}$  in the direction of  $x_{rand}$  by a small  $\epsilon$ -distance or moving in that direction for a small  $\delta$ -time using the local planner resulting in a new point  $x_{new}$ . We would later on see the other implications of this formulation in the kino-dynamic setting. If there are no collisions in joining  $x_{near}$  and  $x_{new}$  an edge is added between them and algorithm proceeds to pick another point. This continues until the goal is reached or maximum points are drawn

figure : RRT basic using the euclidean distance metric.

The probability of *extending* a point in the tree is directly proportional to the *voronoi* region of the point. A voronoi region of a point is all the set of points in the space to which it is the nearest neighbour. Thus RRTs have excellent space filling properties. This accounts for the *rapidly-exploring* aspect of the RRTs. The tree is thus stretched towards the lesser explored regions of the state space. If a big area is unexplored then the probability that the RRT will extent towards that region is higher than other regions. RRTs are shown to be probabilistically complete. The probability of finiding a solution if it exists tends to 1 as the number of samples picked tends to infinity. RRTs have been successfully implemented in many domains as seen in Rapidly-exploring Random Trees: Progress and prospects, Algorithmic and Computational Robotics: New Directions, pp. 293308, 2001. [3] J. Bruce and M. Veloso, Real-time randomized path planning for robot navigation, in Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS), 2002. [4] J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue, Motion planning for humanoid robots, in Proceedings of the International Symposium on Robotics Research (ISRR), 2003. [5] J. Kim and J. Ostrowski, Motion planning of aerial robots using Rapidly-exploring Random Trees with dynamic constraints, in Proceed- ings of the IEEE International Conference on Robotics and Automation (ICRA), 2003. [6] G. Oriolo, M. Vendittelli, L. Freda, and G. Troso, The SRT Method: Randomized strategies for exploration, in Proceedings of the



### 3.1 Algorithm

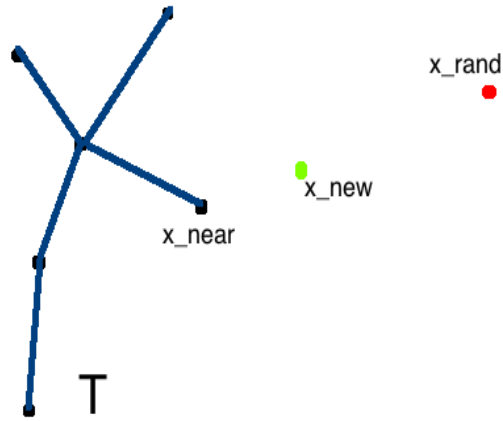


Figure 3.1: Process of building the RRT and extending the current tree to  $x_{rand}$

#### Algorithm *Build-RRT*

(\* builds an RRT in a state space \*)

1.  $V(G) \leftarrow x_{start}; E(G) \leftarrow \phi$
2. **while** goal is not reached
3.      $x_{rand} \leftarrow \text{Sample}(S);$
4.      $x_{near} \leftarrow \text{Nearest}(x_{rand}, V(G));$
5.      $x_{new} \leftarrow \text{Extend}(x_{near}, x_{rand}, S);$
6.     **if** Not-Colliding( $x_{new}, x_{near}, S$ )
7.         **then** Connect  $x_{new}$  to  $x_{near}$
8. **End of Algorithm**

#### Algorithm *Nearest( $x_{rand}, V(G)$ )*

(\* Finds a point nearest to given point in the graph \*)

1. return the vertex that minimises  $\|v - x_{rand}\|$ , where  $\|x-y\|$  is the euclidean norm
2. **End of Algorithm**

**Algorithm *Extend*( $x_{near}$ ,  $x_{rand}$ ,  $S$ )**

(\* Finds a point close to  $x_{rand}$  that is in  $\epsilon$ -neighbourhood of  $x_{near}$  \*)

1. return a point  $x_{new} \in S$  such that  $\|x_{near} - x_{new}\| < \epsilon$  which minimises  $\|x_{new} - x_{rand}\|$
2. End of Algorithm

### 3.2 Improvements

Thus RRTs incrementally build a tree sampling points from the state space. If the state space is large it takes long times to find the path to goal. Hence a little bias is added towards goal to direct the exploration making it faster. It is incorporated in the sample function as seen in line 3 of the following algo.

**Algorithm *Bias-Sample*( $S$ )**

(\* Samples a point  $x_{rand}$  in the state space  $S$  \*)

1.  $prob \leftarrow \text{Random}(0,1)$
2. **if**  $prob < p_{goal}$
3.     **then**  $x_{result} \leftarrow x_{goal}$
4.     **else**  $x_{result} \leftarrow$  Randomly sample a point from the state space
5. return  $x_{result}$
6. End of Algorithm

Other ways in which people have looked speeden up the process is to grow multiple trees towards a region and connect them. One approach was called RRT-Connect( put the fucking reference and add more info from the ppt!). This algorithm keeps track of two trees one from the start and other from the goal states. Initially one tree is begun from start state and it is extended by a small step towards a random point to reach a target point which is then added to the first tree. The second tree now is grown towards the target point and moves in that direction by a small step. This process is continued until the second tree finds an obstacle in trying to move towards the target set by the first tree. In that case, roles of both trees are swapped and now second tree sets target states and first tree tries to reach to the target states. This is continued till both trees meet. This method is usually way faster than the vanilla RRT which explores the whole state space. Hence in applications like piano movers problem where we have to compute a path in a very short limited time such approaches could be useful. The idea of bi-directional trees worked but it limited the way goals could be defined. In many cases the goals are not stationary and it had issues even if the environment is not stationary.

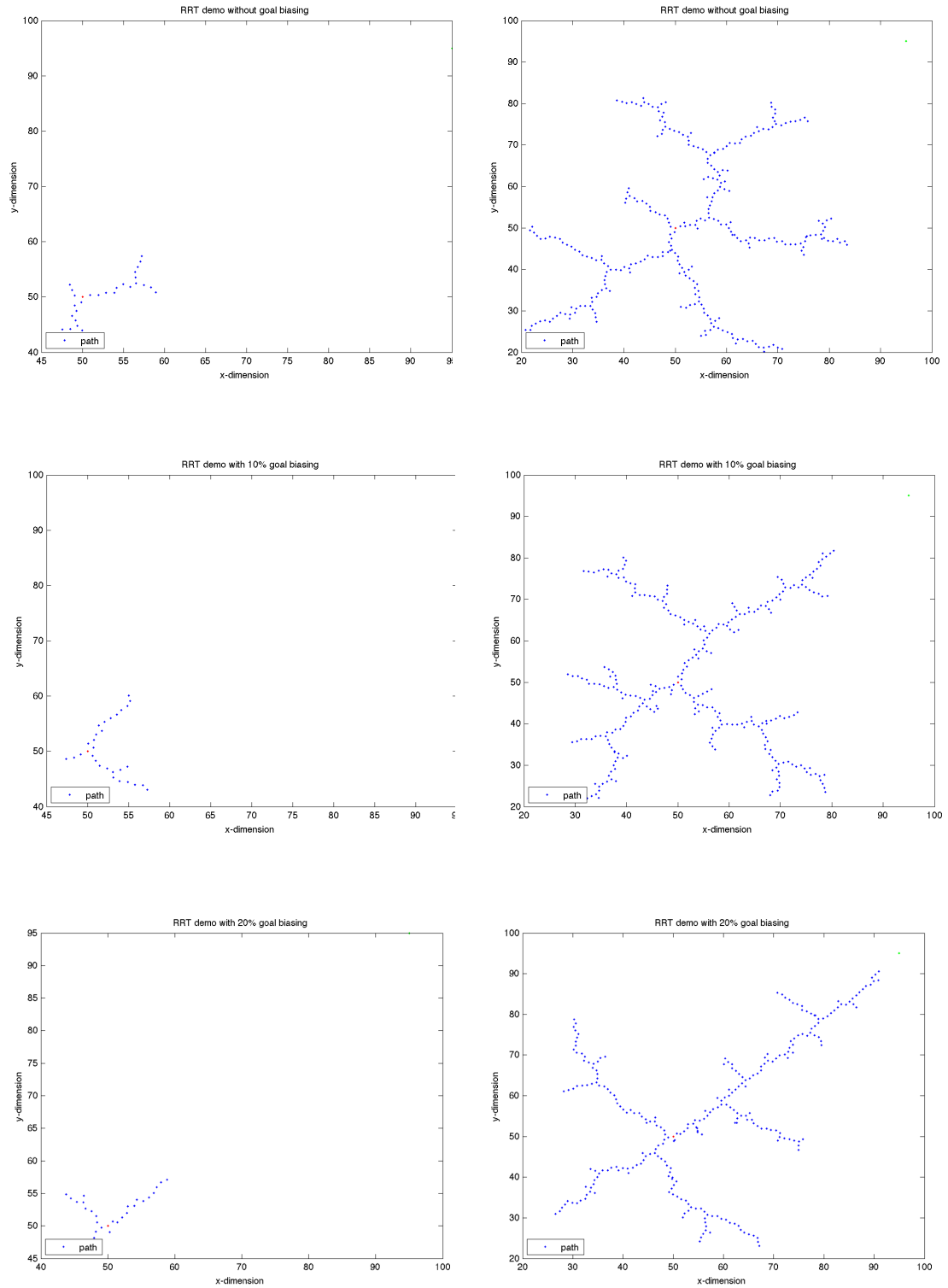


Figure 3.2: A snapshot of growth of RRT starting at [50;50] and goal at [95;95] the first column is at 200 samples and second column at 2000 samples. The first row is taken when sampling randomly, second row at 10 percentage bias to goal and third row at 20 percentage bias towards goal

Brooks et al (*A.Brooks, T.Kaupp, and A.Makarenko, Randomisedmpc-based motion-planning for mobile robots*) proposed a way to include dynamic model of the robot and a cost function to prune the tree and find out solution paths faster. Other approaches looked at finding a better set of vertices to use in extend procedure. Shkolnik et al. (*A.Shkolnik, M.Walter, and R.Tedrake, Reachability – guided sampling – planning for planning under differential constraints, in Robotics and Automation, 2009. ICRA09. IEEE International Conference on*) used reachable configuration set from a point to prune the possible list of vertices to be considered to extend to. Sometimes it is better to find a suboptimal solution quickly if we need online implementation of the algorithm. Ferguson et al (*D.Ferguson and A.Stentz, Anytime, dynamic planning in high – dimensional search spaces, in Proc. IEEE International Conference on Robotics and Automation, 2007, pp.1310-1315*) used the idea of anytime planning where the algorithm produces a path anytime and when it is being executed it iterates again to find better paths. Csaba et al (RRT++ paper reference) uses the same idea and proposed RRT++ algorithm which finds a path and optimises it by running the algorithm again minimising on cost metric.

Few other variants of RRTs in dynamic environments are discussed in the next chapter.

### 3.2.1 Optimality

The basic RRT algorithm is just probabilistically complete. The solution given by RRT algorithm is not guaranteed to be optimal. Detailed analysis of the optimality of the sampling based algorithms PRM and RRT were done by Frazzoli et al (Sampling-based Algorithms for Optimal Motion Planning) and proposed RRG, RRT\* optimal versions of the algorithms. In RRG ( rapidly exploring random graph) they build an incremental connected roadmap which might contain cycles also. It is similar to RRT except that everytime a new node  $x_{new}$  is added to the current tree, all the points that are in a small neighbourhood  $X_{near}$  ( ball of radius  $r$ ) of the new node are tried to connect to the tree if they pass the collision checks. They came up with a good approximation of what the  $r$  should be so that we get an optimal graph.  $r(card(V)) = \min(\gamma_{RRT^*}(\log(card(V))/card(V))^{1/d}, \eta)$ . RRT\* is an extension of the approach where we try and maintain tree property of the graph given by RRG when ever a new node is added by removing *redundant* edges. There is an additional *rewiring* procedure in RRT\* that does exactly this. Rewiring is done as storing tree structures is much easier and efficient from memory perspective and also searching for a path becomes trivial in case of tree data structures. They define a cost function for every point and the goal is to generate a path of minimum cost between start and end states. Let  $v$  be any point in the tree. Let  $Parent(v)$  denote the parent node in the tree data structure.  $Cost(v)$  denotes the cost and one can generate a child node cost from the parent node by  $Cost(v) =$

$\text{Cost}(\text{Parent}(v) + \text{Cost}(\text{Line}(\text{Parent}(v), v)))$  In RRT\* any node is only added to the tree if it passes through the low cost path. Everytime  $x_{new}$  is added, it considers connections to all points in  $X_{near}$ , the points in a small neighbourhood of  $x_{new}$  defined by them. But not all connections are accepted, only in the following two cases. i) If the point is already a part of the tree, then connection is made through  $x_{new}$  only if the cost of the path through the current parent of the point is greater than the path through  $x_{new}$ . ii) an edge is created to the vertex in  $X_{near}$  that can be connected to  $x_{new}$  along the minimum cost path. Another version called k-nearest RRT\* implements the same but considers only the k-nearest neighbours instead of the r-radius disc where k is governed by  $k(\text{card}(V)) = k_{RRG} \log(\text{card}(V))$  for populating the nearest nodes to be expanded  $X_{near}$ .

### 3.2.2 Issues

Even though it is an optimal extension of the vanilla RRT algorithm, it is difficult to apply it in domains with underactuated or complicated domains (LQR-RRT\*: Optimal Sampling-Based Motion Planning with Automatically Derived Extension Heuristics) as it requires us to design a distance metric cost and also we need to come up with the node extension method that has a lot of domain dependant terms. The entire optimality argument in Frazzoli et al ( Sampling-based Algorithms for Optimal Motion Planning ) depends on those two heuristics and if the values we use are off the optimal values, RRT\* would no longer produce good optimal paths. In most of the modern real world systems requiring motion planning like autonomous cars, unmanned drones, and other humanoid robots with complex linkages, more often than not we would tend to find underactuated dynamics ( the number of controllable degrees of freedom are less than the actual degrees of freedom ). Hence we need an algorithm that works well in high dimensional underactuated systems also. Attempts have been made by Perez et al( LQR-RRT\*: Optimal Sampling-Based Motion Planning with Automatically Derived Extension Heuristics ) to derive the two heuristics used in RRT\* by locally linearizing the domain and applying Linear Quadratic Regulation on that domain. But is not very intuitive why it would work in a complex domain where the dynamics are not linearizable. For example if there are obstacles are random parts of the domain one cannot expect to do LQR using data in some part of the domain and generalize it to the entire space. We might get a good estimate in some cases but there is no guarantee that it will work in all domains. This gives the motivation to our work where we would like to use some methodology to learn these domain dependant metrics from the trajectories instead of plainly applying regression. We would see in the future chapters that Reinforcement Learning(RL) is traditionally one field of Computer Science where domain dependant metrics have been learnt through experience with the environment.

## CHAPTER 4

### Planning in Dynamic Environment

Traditionally one way to look at Dynamic environment was to include the time also in the state space. So instead of simply planning in configuration  $C$  space now planning is done in Configuration-Time  $C-T$  space as introduced by Fraichard ( Fraichard. Trajectory planning in a dynamic workspace: a state-time approach. Advanced Robotics, 13(1):7594, 1999.). They address the problem of moving obstacles and dynamical constraints of the workspace in the robot in a unified way by solving in the state-time space. They model the problem with dynamical constraints as canonical trajectories which are trajectories with discrete and piecewise linear accelerations. We use the same formulation in our experiments later on. They use canonical trajectories in finding a shortest path in the state-time space. In (J. van den Berg and M. Overmars. Roadmap-based motion planning in dynamic environments. IEEE Transactions on Robotics, 21(5):885897, 2005 ) they use a two level planner to find safe trajectories in dynamic environment consisting of both static and dynamic domains. They use the roadmap constructed initially considering only the static obstacles and use two level search when querying a path from start to goal configurations. The local planner finds trajectories using depth first search on a state-time grid and and the global planner uses the local trajectories generated and and performs  $A^*$  search on them to find a path to goal configuration.

Svenstrup et al.(Trajectory Planning for Robots in Dynamic Human Environments) implemented a trajectory planning algorithm using RRTs dynamic human environments by modelling as  $C-T$  space with Model Predictive Control, where only a small part of the trajectory is executed when the new point is being calculated in the rrt iteration. They modelled dynamic human environment as a potential field with humans as obstacles emitting repulsive field and finding a minimum cost path according to the cost of traversing in the potential field. They have encoded the dynamic nature of the environment using the dynamic potential field equation which is

$$G(t) = g_1(x(t)) + g_2(x(t), P(t)) + g_3(x(t))$$

where  $G$  is the value of the potential field and  $g_1$  corresponds to cost because of the position of the robot in the environment neglecting the obstacles.  $g_2$  relates the cost associated with the robot based on its distance from the obstacles and  $g_3$  rewards the robot if its moving towards the goal. Based on this

dynamic potential function they have successfully implemented path planning using RRTs following this potential field in a simulated artificial human environment. This gives motivation for encoding the dynamic nature of the environment in *value* function like metric and following RRT procedure. The disadvantage of this method like any other potential field method is that it might run into local minima. Our approach which we define later uses similar value functions but it is learnt using another framework called Reinforcement Learning which we will introduce later. It also gives motivation for an anytime algorithmic framework where we could improve upon our solutions if there is time available during the online execution. We use another framework called DYNA (Put dyna reference here ! ) where we do planning in the time available and improve our solutions.

On the other hand PRMs could also be adapted to dynamic environment by working in Configuration - Time space. But since there is no necessity that motion of obstacles should be periodic or follow a certain path, the state space in which the RRT is growing could be highly transitional. Hence building the roadmap in preprocessing stage doesn't help much. Hence single shot methods like RRTs are preferred.

K. Fujimura. Motion planning in dynamic environments. Springer-Verlag, Tokyo, Japan, 1991.

K. Fujimura. Time-minimum routes in time-dependent networks. IEEE Transactions on Robotics and Automation, 11(3):343-351, 1995.

## 4.1 Predicting obstacles

Modelling the path of the obstacle in the environment can be classified into three categories. First is the worst case situation where the obstacle is trying to actively collide with the agent. Aoude et al (Aoude GS, Luders BD, How JP (2010a) Sampling-based threat assessment algorithms for intersection collisions involving errant drivers. In: IFAC Symposium on Intelligent Autonomous Vehicles, Lecce, Italy) proposed an algorithm which models the whole problem from the game theoretic standpoint as a pursuit evasion game where obstacle is the pursuer and agent is the evader. It uses RRTs to simulate possible trajectories of pursuer and thereby finding out reachability set in a given time frame. They have included timestamp in the state representation to keep track of time with respect to trajectories. The resulting algorithm, RRT-Reach is a two-phase exploration-pursuit modification to vanilla RRT. Kutawa et al. (Motion Planning for a Driving using RRT ; Yoshiaki Kuwata, Gaston A. Fiore, Justin Teo, Emilio Frazzoli, and Jonathan P. How) implemented similar ideas where they check if a trajectory is colliding with any obstacle and generate plans to avoid the obstacle using a map of the state space.

Another way of modelling the obstacles are by looking at their current dynamics and predicting their future trajectories assuming them to follow a fixed mode of operation i.e by propagating its dynamics forward. The prediction can be done using a Kalman filter or any continuous bayes filters. It works well for short term prediction but when the obstacles are not periodic or not following a certain fixed path then they do not predict the obstacles properly to be used by any path planning algorithm. Another way of modelling dynamic obstacles is by assuming them to follow different patterns in different parts of the state space. And we use previous trajectories across the state space to predict those patterns. Using discrete state space predicting techniques a motion model is developed where the object transitions from one state to other state with some transition probability which they attempt to learn. In clustering based techniques, previous trajectories are clubbed together into representative clusters and a new partial path is identified against its suitable representative cluster which would give the prediction on the future path of the obstacle. Due to the discretizations we need to perform to implement discrete state space techniques they may suffer from overfitting if the resolution is set too high unlike the clustering based techniques. Fulgenzi et al (Fulgenzi C, Tay C, Spalanzani A, Laugier C (2008) Probabilistic navigation in dynamic environment using rapidly-exploring random trees and gaussian processes. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Nice, France, pp 10561062) uses Gaussian Processes to model the dynamic obstacles in a RRT based path planner. Aoude et al (Probabilistically Safe Motion Planning to Avoid Dynamic Obstacles with Uncertain Motion Patterns Georges S. Aoude Brandon D. Luders Joshua M. Joseph Nicholas Roy Jonathan P. How) proposed a method that uses Chance constraint RRTs to calculate the reachability set and Gaussian Process regression to predict the future states of the obstacles. However with increasing number of obstacles and complex paths it becomes more difficult to keep track of all the obstacle trajectories.

In non-stationary and partially known environment where we do not have obstacle models, instead of predicting the path of obstacles efficient replanning can be done using RRTs. We look at some of the approaches to do that in the next section.

## 4.2 Replanning

A RRT that has been build for an environment would become invalid when the environment changes. One approach would be to do replanning whenever the environment changes. But it is highly time consuming in case of complex domains. Hence some improvements to replanning have been proposed based on temporal coherence assumptions. Bruce et al (J. Bruce and M. Veloso, Real-time randomized



path planning for robot navigation, in Proceedings of IROS-2002, Switzerland, October 2002, an earlier version of this paper appears in the Proceedings of the RoboCup-2002 Symposium.) proposed that the solutions discovered should be put in a waypoint cache which could be used in the later iteration of the RRT. In this version which they call Execution Extended RRT( ERRT), every time environment changes they do replanning from scratch. But with some probability  $p_{\text{waypoint}}$  the points that are in waypoint cache from the previous successful runs are used to guide the RRT to reach goal quicker. In practise ERRT performs much better than the naive RRT( put referene of ERRT) if there are small incremental changes to the configuration space  $C_{\text{free}}$  as the states that were in successful trajectories in the previous runs could be used for the current run also. Ferguson et al( D. Ferguson, N. Kalra, and A. Stentz, Replanning with rrts, in Proc. IEEE International Conference on Robotics and Automation ICRA 2006, 2006, pp. 12431248.) exploit temporal coherence assumption in a different way where they use large parts of previous trajectories itself instead of just guiding the path. After an RRT is build, if the environment changes they check all the current states of the trajectory if its still valid with respect to the changed conditions. If any part of the tree is invalid, the invalid part is pruned and the remaining portion of the tree is used to initialise the next episode of search and tree is regrown from that. This algorithm goes by the name Dynamic RRT(DRRT). It can be seen that at every iteration there is a large overhead of collision checking with all the nodes in the tree. But the advantage is that a large part of the tree is already planned from the previous iteration. Zucker et al( M. Zucker, J. Kuffner, and M. Branicky, Multipartite rrts for rapid replanning in dynamic environments, in Proc. IEEE International Conference on Robotics and Automation, 2007, pp. 16031609. ) combines ideas from both to propose MultiPartite RRT(MPRRT) algorithm which maintains a forest  $F$  of disconnected trees which lie in  $C_{\text{free}}$  but are not connected to the start node of the RRT  $T$ . Whenever the environment is changed any nodes that are still not in  $C_{\text{free}}$  are removed from the tree  $T$  and the disconnected subtree that are formed as a part of the removal are placed in the forest of subtrees  $F$ . After pruning the start point may lie in the tree  $T$  in which case RRT build operation is performed else the entire tree is put in the forest  $F$  and RRT is run from scratch again. The main difference between this and other approaches is during sampling one of the trees in the forest  $F$  are sampled and the root of the sampled tree is attempted to connect to the RRT by the standard RRT-Extend procedure described in Chapter 3. These algorithms motivate the effectiveness of replanning in dynamic environments without explicitly keeping track of all the obstacles in the environment with undefined motion patterns.

We present our algorithm which learns a value function across the state space and does replanning to tackle motion planning in non-stationary environments. We present few preliminaries in the next chapter

and introduce RRTPI algorithm.

P. Leven and S. Hutchinson. Real-time motion planning in changing environments. In Proc. International Symposium on Robotics Research, 2000.

B. Baginski. The Z3 method for fast path planning in dynamic environments. In Proceedings of IASTED Conference on Applications of Control and Robotics, pages 4752, 1996.

J. Kim and J. P. Ostrowski. Motion planning of aerial robot using rapidly-exploring random trees with dynamic constraints. In IEEE Int. Conf. Robot. & Autom., 2003.

P. Fiorini and Z. Shiller. Time optimal trajectory planning in dynamic environments. Journal of Applied Mathematics and Computer Science, 7(2):101126, 1997.

P. Fiorini and Z. Shiller. Motion planning in dynamic environments using velocity obstacles. International Journal of Robotics Research, 17(7):760772, 1998.

## **CHAPTER 5**

### **RRTPI**

#### **5.0.1 MDP**

J. Barraquand and J.-C. Latombe. A Monte-Carlo algorithm for path planning with many degrees of freedom. In IEEE Int. Conf. Robot. &Autom., pages 17121717, 1990.

## **CHAPTER 6**

### **Future Work**