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 finds 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 has 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 Markove Decisoin 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 continous state domain. We then present our results in a continous domain with kino-dynamic constraints.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

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, autonmous 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 lets consider the famous mobile game clash of clans which involves a players 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 kinamatic 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. Configurations 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

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

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 doesnt 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 Dijsktra'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 derivates 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 invloving 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 correclty. 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 is 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 genarality 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 incresing 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 (lavalle 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 prepossessing 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 algoritms 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 mininize the collisions the algorithm instead of joining x_{rand}, tries to move from x_{near} in the direction of x_{rand} by a small ϵ-distance or moving in that direction for a small δ-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 streched towards the lesser exploered 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

IEEE International Conference on Robotics and Automation (ICRA), 2004.)

3.1 Algorithm

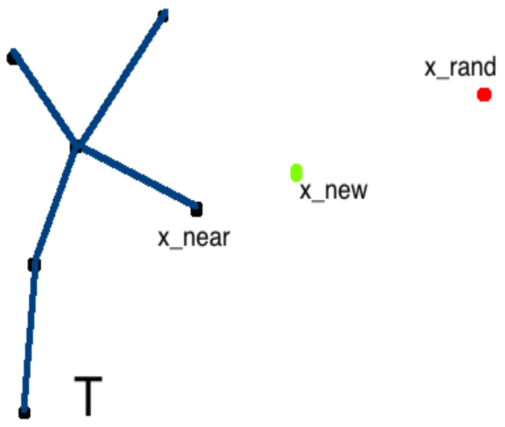

Figure 3.1: Process of building the RRT and extending the current tree to x_rand

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.        $x_{rand} \leftarrow$ Sample(S);
4.        $x_{near} \leftarrow$ Nearest($x_{rand}$, V(G));
5.        $x_{new} \leftarrow$ 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 follwing algo.

```
Algorithm Bias-Sample(S)
(* Samples a point $x_{rand}$ in the state space S *)
1.  prob ← 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.

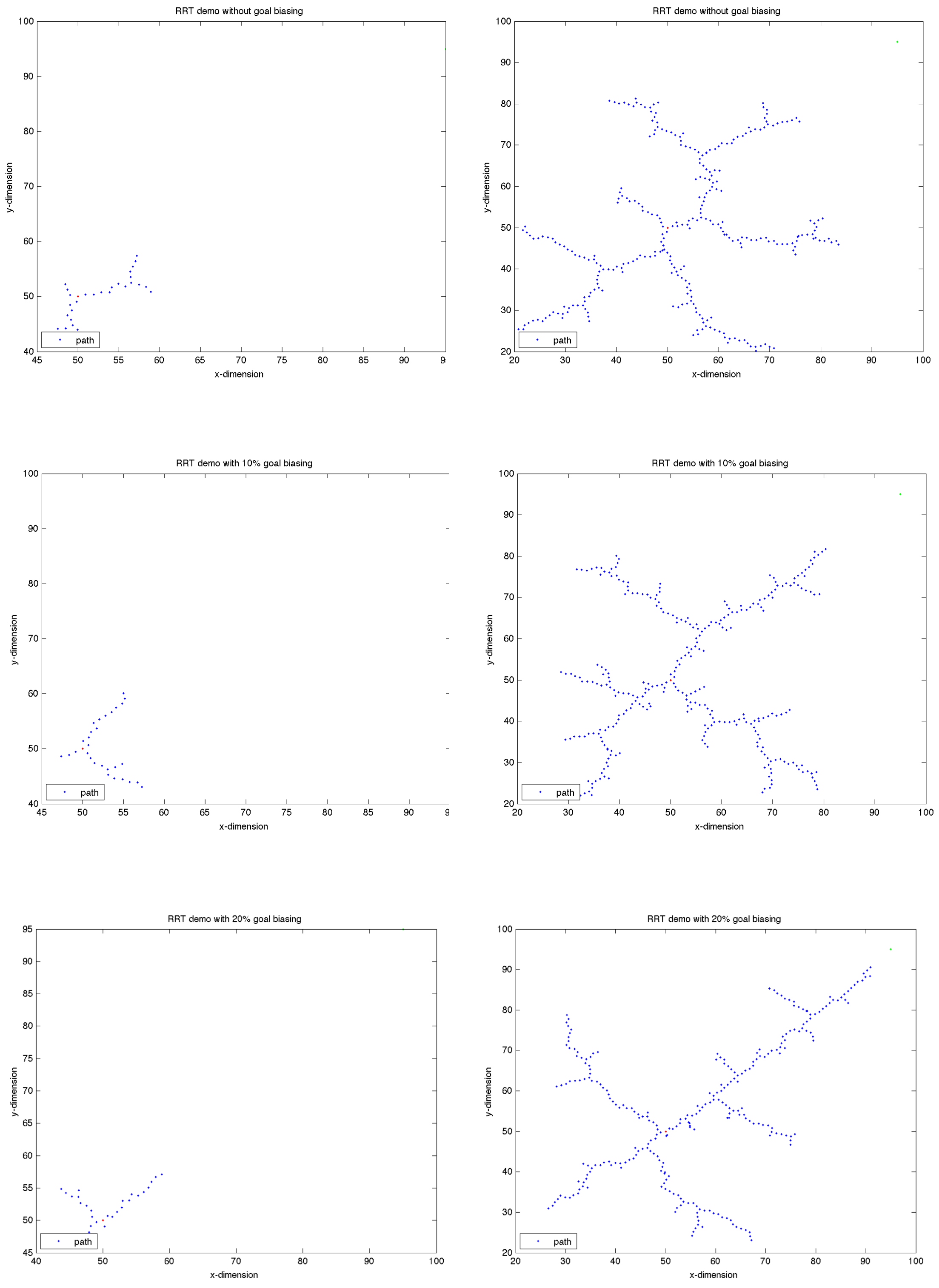

Figure 3.2: A snapshot of growth of RRT starting at [50;50] and goal at [95;95]the first column is at 200 samples and second column at 2000 samples.The first row is taken when sampling randomly , second row at 10 percentage bias to goal and third row at 20 percentage bias towards goal

Brooks et al $(A.Brooks, T.Kaupp, andA.Makarenko, Randomisedmpc{-}basedmotion{-}planningformobilero$
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.$(A.Shkolnik, M.Walter, andR.Tedrake, Reachability - guidedsam -$
$plingforplanningunderdifferentialconstraints, inRoboticsandAutomation,$ 2009.$ICRA$09.$IEEEInternatio$
used rechable confuration 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 $(D.FergusonandA.Stentz, Anytime, dynamicplanninginhigh -$
$dimensionalsearchspaces, inProc.IEEEInternationalConferenceonRoboticsandAutomation,$ 2007$, pp.$1310
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 probalistically complete. The solution given by RRT algorithm is not guarenteed 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*) =

Cost(Parent(v) + Cost(Line(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(card (V)) = k_{RRG} log(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 dufficult 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 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 ofthen 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 guarentee 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 int he 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 mininum 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 transitionary. Hence building the roadmap in preprocessing stage doesnt 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-mini mum routes in time-dependent networks. IEEE Transactions on Robotics and Automation, 11(3):343351, 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 Intel- ligent 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 there by 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 propogating its dynamics forward. The predicion 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 certiain 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. Fulgenzii 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 Guassian 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 Guassian 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 ot 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 proability $p_{waypoint}$ 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_{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_{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_{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 explicity 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 Sym- posium 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 dy- namic 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

The performance of RRTs heavily depend on the choice of the distance metric used in the calculation of the nearest points (S. M. Lavalle, From dynamic programming to RRTs: Algorithmic design of feasible trajectories, in Control Problems in Robotics. Springer-Verlag, 2002.). Even in $RRT*$ algorithm the optimality guarentees are valid only for the right choice of heuristics used in the algorithm. One field that has traditionally estimated domain dependant metrics using experience is Reinforcement Learning(RL). RRTPI alos tries to estimate the domain dependant distance metric by a value function using trajectories sampled from the domain and improving its estimate of the value function by Policy Iteration technique from Reinforcement Learning. We first define few important concepts that we use in RL here and then show how the distance metric is estimated from the trajectories.

5.0.1 Reinforcement Learning

We model the problem of finding the optimal path as solving an Markov Decision Process (MDP). A continuous MDP M is a tuple $< S, A, T, R, \gamma >$ where $S \in R^n$ is the n-dimensional state space or the configuration space. A is the action space. It is a set of actions in discrete domains. T(s,a,s') is the transition probability of reching state $s' \in S$ by performing action $a \in A$ on state $s \in S$. And for deterministic domains we can denote $T(s, a) = s'$. Note that $\sum_{\forall s' \in S} T(s, a, s') = 1$. $R(s, a, s')$ is the expectation of the real valued rewards for action $a \in A$ taken at state $s \in S$ thereby reaching state $s' \in S$. The rewards are usually bounded between $[R_{min}, R_{max}]$ and γ is the discount factor. We begin in starting state $s_o \in S$ and take action a_o and reach state s_1 according to the transition probability $T(s_o, a_o, s_1)$ and receive a reward r_1 drawn accordingly from the reward distribution $R(s_o, a_o, s_1)$. After a fixed time step, we pick the next action for the state s_1 which is a_1 according to $T(s_1, a_1, s_2)$ and get a reward r_2 according to $R(s_1, a_1, s_2)$. The procedure is continued till we reach our final state or maximum number of steps are reached resulting in the sequence of state, action, reward pairs $s_o, a_o, r_1, s_1, a_1.r_2, s_2, a_2, r_3, s_3...$ Our objective is to choose the sequence of actions $a_o, a_1, a_2, a_3, ...$ such that we maximize the cumulative reward or return. We define return as $\sum_{t=0}^{\infty} \gamma^t r_t$, where r_t is the reward at the time step t and $\gamma \in [0, 1]$

A policy π is a mapping from state space S to action space A $\pi : S\text{x}A \rightarrow [0,1]$, it represents the probability of choosing some action $a \in A$ for any state $s \in S$. A policy can be probabilistic or deterministic. A deterministic policy is represented as $\pi(s) = a$ where a is the action selected for state s. On the other hand $\pi(s,a) \in [0,1]$ gives the probability as defined above. The state value function $J^{\pi}(s)$ is the expected return from any state s while following a policy π. In discrete domains we can define the state value function using the Bellman equation as follows :

$$J^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma J^{\pi}(s')]$$

Optimal policy is defined as the one, in which for any other policy, $\forall s \in S, \pi$ $J^{\pi}(s) < J^{\pi^*}(s)$, where π^* is the optimal policy. In case of continuous state spaces. we approximate the value function either using parametric approximation or non-parametric or by nearest neighbour techniques.

Policy Evaluation

The bellman equation can be solved using Dynamic Programming techniques (R.S. Sutton and A.G. Barto. Reinforcement learning: An introduction, volume 28. MIT press, 1998.) if the transition probabilities are available for all possible states and actions. But it is never the case in real world systems. A popular class of algorithms solve this issue by sampling the trajectories according to a policy and estimating the value function. It is called policy evaluation.

Temporal Difference(TD) learning is a class of algorithms which estimate the value function using temporal differences (R. S. Sutton. Learning to predict by the methods of temporal differences. Machine Learning, 3, 1988.) of the trajectories. Given a set of N trajectories $\{s_o, a_o, r_1, s_1, a_1, r_2, s_2, ...s_M\}_{i=1}^{N}$, TD(0) method estimates the value function J^{π} using the following update rule

$$J^{\pi}(s_t) = (1-\alpha)J^{\pi}(s_t) + \alpha(r_t + \gamma J^{\pi}(s_{t+1}))$$

where $0 \leq i \leq N$ and $1 \leq t \leq M_i$. The parameter α is also known as the step size and it determines the learning rate. It controls the bias-variance tradeoff. Lower the alpha, lower the variance. The parameter γ determines how much importance is given to future rewards. When γ is set to 0 only the immediate rewards are considered and when γ is set higher future rewards are also given importance. Note that γ is not bounded in [0,1]. TD algorithms converge when alpha is gradually decayed.

In discrete state space, all the states can be enumerated and value function can be calculated for all the points and the above equation can be applied. But in continuous state spaces one has to do some approximation. The most common way of dealing with this is to approximate the value of a new sample to the value of its nearest neighbours assuming it is locally constant. Other ways include assuming the value function is locally linear within a neighbourhood and performing regression to find out the value of a given state. Other ways include function approximation and one could even use neural networks (Issues in Using Function Approximation for ReinforcementLearningSebastian Thrun Anton SchwartzInstitut fur Informatik III Dept. of Computer Science)to approximate the value function in continuous domains.

5.0.2 RRTPI Algorithm

RRTPI begins with uniform estimate of the value function J_o which would represent the cost-to-go metric .RRTPI works by first sampling trajectories from the state space according to a RRT algorithm. But the difference in which both algorithms build trees is that RRT uses euclidean distance to find out nearest points of any randomly sampled point where as RRTPI uses the value function over the state space. This forms roughly policy improvement step. The sampled trajectories are used to evaluate and update the value function using the TD(0) update rule. This forms the policy evaluation stage. RRTPI consists of a series of policy evaluation and improvement stages. Thus it is a kind of Policy Iteration using RRTs.

```
Algorithm Build-RRT(S, $||.||_J$)
($*$ samples trajectories using RRTs $*$)
1.  V(G) $\leftarrow x_{start}$; E(G) $\leftarrow \phi$
2.  n $\leftarrow$ 0
3.  while goal is not reached
4.          $x_{rand} \leftarrow$ Sample(S);
5.          $x_{near} \leftarrow$ Nearest($x_{rand}$, V(G) , $||.||_J$ );
6.          $(x_{new}, a, r) \leftarrow$ Extend($x_{near}$, $x_{rand}$, S);
7.          Connect $x_{new}$ to $x_{near}$
8.          V(G) $\leftarrow$ V(G) U $x_{new}$
9.          E(G) $\leftarrow$ E(G) U $(x_{new}, a, r)$
10. return G
11. End of Algorithm
```

Algorithm *Nearest(x_{rand}, V(G) , $||.||_J$)*

($*$ Finds a point nearest to given point in the graph with respect to the value function $*$)

```
1.  return $x_{near} \in V(G)$ such that $x_{near} = arg \max_{x \in V(G)} (||x - x_{rand}||)$
2.  End of Algorithm
```

$||x - y||_J$ = $J(x) - J(y)$ where x, y $\in$ S.

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

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

```
1.  $x_{new} \leftarrow arg \max_{a \in A} ||x - x_{rand}||_J$ and $(r, x_{new}) \leftarrow$ MDP($x_{near}$,a)
2.  return $(x_new, a, r)$
3.  End of Algorithm
```

Algorithm *NN-TD(G, J)*

($*$ Estimates the value function J from the trajectories $*$)

```
1.  Let $Y_n$ be the set of trajectories starting at $x_start$ till leaf nodes in G.
    for each trajectory $(s_o, a_o, r_1, s_1, a_1, r_2, s_2, ...)$ in $Y_n$
2.          for each pair $(s_i, a_i, r_{i+1}, s_{i+1})$
3.                  $J(s_i) = (1 - \alpha)J(s_i) + \alpha(r_i + \gamma J(s_{i+1})$
4.  return J
5.  End of Algorithm
```

Algorithm *RRTPI(N)*

($*$ builds RRT and implements policy iteration $*$)

```
1.  Initialize $J_o \leftarrow$ 0 uniformly for all i = 1..n
2.  n$\leftarrow$ 1
3.  while n ¡ N
4.          $G_n \leftarrow$ Build-RRT($M_n$, $||.||_{J_{n-1}}$)
5.          $J_n \leftarrow$ NN-TD($G_n$ , $J_{n-1}$)
6.          $n \leftarrow$ n+ 1
7.  End of Algorithm
```

(Efficient Continuous-Time Reinforcement Learning with Adaptive State Graphs) had used RL in graph based sampling algorithm. They show that value function based methods can be utilised to solve the motion planning tasks in unknown dynamic environments. Their approach lacks in a way that it is similar to PRM methods which has preprocessing overhead. Also it doesnt take the kinodynamics of the system into account when generating the plans.

To apply the algorithm to dynamic environment we also need to modify the algorithm for effective replanning. As we saw in many RRT improvements, anytime planners are more practical in online settings than single shot RRT planners as executing the time itself might take some time and the environment would have changed in that time interval. Hence we propose a online algorithm that would utilise the time when the plan is being executed in planning and one nice framework that looks at this issue is DYNA framework proposed by sutton et al(put the fucking reference !) in the next chapter.

5.1 DYNA-RRTPI

In case of on-line planning, interactions of agent with environment might change the environment model and thus further planning is impacted. DYNA framework [17] puts together the decision making and model learning. The real experience can be used to improve a model also known as model learning or improve the value function like in reinforcement learning, sometimes called direct reinforcement learning. In the above plot we summarized the possible relationship between experience, model learning and planning, direct reinforcement learning. It can be seen that experience can improve the value function both directly and indirectly. The indirect reinforcement learning stage is sometimes referred to as planning. These indirect methods are better in the sense that they make full use of experience and require fewer environmental interactions as planning is involved where as direct reinforcement learning is much simpler and are not affected if model is biased. DYNA framework brings together all the acting, model learning, planning and direct reinforcement learning into a single framework. The figure shows how various stages interact with each other in the framework. The middle column represents the interaction between agent and environment. Left side represents the direct reinforcement learning update. In case of TD learning the value function update is the direct RL step.The right side corresponds to model based learning.An explicit model is maintained, in which each transaction is recorded. Planning takes place on learnt model using simulated experiences. For simulating experiences starting states and actions are selected using the model. We use search control to represent that in figure. Learning and planning are thus integrated in this common framework where learning is done on actual experience where as planning is done on simulated one which is based on model. The first 3 stages of acting, model learning and direct reinforcement learning are done in very small fraction of the time and planning is done in the

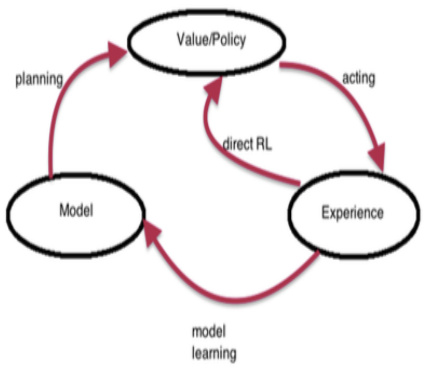

Figure 5.1: Dyna 1

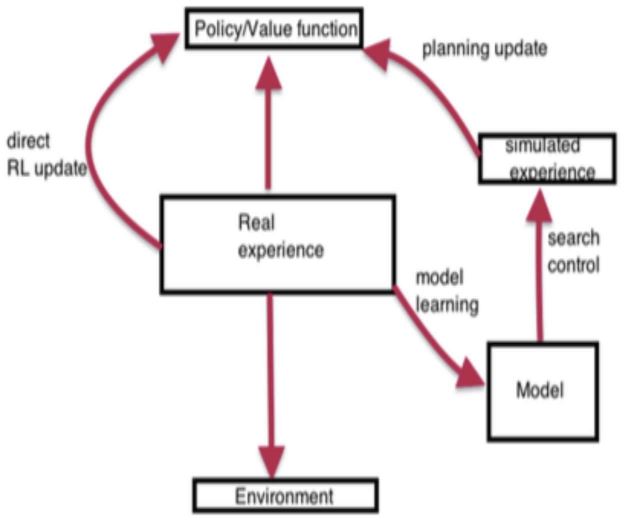

Figure 5.2: Dyna 1

rest of the time. An usual iteration of the agent would have it selecting an action to be performed in the current state based on some greedy approach. It then updates the value function and the transition is recorded in the model. Then N-iterations of planning take place on the existing transitions in the model. After each simulated transition, value function is updated which forms indirect RL update. Improvements have been suggested to Dyna framework like Dyna+ where the most oldest action that has not been selected gets more reward. Other methods like prioritized sweeping makes sure that planning stage is more efficient. Selection of state during simulated experience is based on the value increment they offer.

We have seen that domain dependant heuristics in RRT^* control the performance of the algorithm and efforts made to learn them using different methods(LQR paper reference). We would like to do TD learning on the trajectories to learn the appropriate value function across the state space. Also we have seen in (human environment paper) where they represent the dynamic behaviour of the environment using some sort of cost fucntion across the state space with good results.In most of the real world problems it is critical to maintain the kino-dynamic constraints of the agent. We incorporate a local planner in the Extend step to take care of the kino-dynamic constraints when generating the RRT. We also would like to make the algorithm online by interleaving planning and acting. In (Sampling based MDP Planning) it was shown that DYNA framework beats other model free learning algorithms in sampling based MDP planning. With this motivation ,we propose an algorithm that implements policy iteration in continuous domains using RRTs and does replanning effectively using DYNA framework. We first present the offline episodic version of the algorithm which computes path in less number of samples as it efficiently explores the state space and estimates the value function by planning using the current tree.

5.1.1 Algorithm

```
Algorithm DYNA-RRTPI(N)
(* Episodic offline version: builds RRT and implements policy iteration using DYNA framework *)
1.  G← Build a vanilla RRT using Euclidean distance metric
2.  $J_o \leftarrow$ NN-TD(G) i.e use the vanilla RRT over the space to initialize the J values
3.  n ← 0
4.  while n ¡ N
5.          $G_n \leftarrow$ Episode(S,$||.||_{J_{n-1}}$)
6.          $J_n \leftarrow$ NN-TD($G_n$ , $J_{n-1}$)
```

```
7.        n ← n+1
8.        End of Algorithm
```

Episode procedure builds the RRT from start state to goal state using the value function and performs planning at every step.

Algorithm *Episode(S,* $||.||_J$*)*

```
(∗ Build a RRT using value functions in dyna framework ∗)
1.  V(G) ← $x_{start}$; E(G) ←$\phi$
2.  while goal is not reached
3.        $x_{rand}$ ← Sample(S);
4.        $x_{near}$ ← Nearest($x_{rand}$, V(G) , $||.||_J$ );
5.        $(x_{new}, a, r)$ ← Extend($x_{near}$, $x_{rand}$, S);
6.        if Not-Colliding($x_{new}$, $x_{near}$,S)
7.          then Connect $x_{new}$ to $x_{near}$
8.              V(G) ← V(G) U $x_{new}$
9.              E(G) ← E(G) U $(x_{new}, a, r)$
10.             $J(x_{near}) \leftarrow (1-\alpha)J(x_{near} + \alpha(r + \gamma J(x_{new})))$
11.       J ← RRT-Planning(G, J, number of iteration)
12. return G
13. End of Algorithm
```

Steps 3,4,5 correspond to action selection in dyna framework. Steps 8,9 corresponds to model learning. Step 10 corresponds to direct RL update and step 11 is where planning is done.

Algorithm *NN-TD(G, J)*

```
(∗ Estimates the value function J from the trajectories ∗)
1.  Let $Y_n$ be the set of trajectories starting at $x_start$ till leaf nodes in G.
2.  for each trajectory $(s_o, a_o, r_1, s_1, a_1, r_2, s_2, ...)$ in $Y_n$
3.        for each pair $(s_i, a_i, r_{i+1}, s_{i+1})$
4.              $J(s_i) = (1-\alpha)J(s_i) + \alpha(r_i + \gamma J(s_{i+1})$
5.  return J
6.  End of Algorithm
```

Algorithm *Nearest(x_{rand}, V(G) , $||.||_J$)*

```
(∗ Finds a point nearest to given point in the graph with respect to the value function ∗)
1.  return $x_{near} \in V(G)$ such that $x_{near} = arg \max_{x \in V(G)} (||x - x_{rand}||)$
2.  End of Algorithm
```

$||x - y||_J$ = $J(x) - J(y)$ where x, y $\in$ S.

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

```
(∗ Finds a point close to $x_{rand}$ that is in $\epsilon-$neighbourhood of $x_{near}$ ∗)respecting the kinodynamic con-
    straints
1.  $x_{new} \leftarrow arg \max_{a \in A} ||x - x_{rand}||_J$—— $x_{new} - x_{near}||$ is within kinematic bounds and action $a_{max}$
    which generated $x_{new}$ is within the acceleration bounds
2.  $(r, x_{new}) \leftarrow$ MDP($x_{near}$,a)
3.  return $(x_new, a, r)$
4.  End of Algorithm
```

Algorithm *RRT-Planning(G, $||.||_J$, number of iterations)*

```
(∗ samples trajectories using RRTs ∗)
1.  n ← 0
2.  while n < number of iterations
3.          Randomly sample a point $x_{begin}$ from V(G)
4.          Find a trajectory $Y$ with $x_{begin}$ as the root
5.          for each pair $(s_i, a_i, r_{i+1}, s_{i+1}) \in Y$
6.                  $J(s_i) = (1 - \alpha)J(s_i) + \alpha(r_i + \gamma J(s_{i+1})$
7.  return J
8.  End of Algorithm
```

For online algorithms it is not possible to predetermine paths by running multiple episodes. But DYNA-RRTPI will perform planning when the robot is executing the action. For online setting, the DYNA-RRTPI could be modified as

Algorithm *Online-DYNA-RRTPI( )*

```
(∗ Online version: builds RRT and implements policy iteration using DYNA framework ∗)
1.  G← Build a vanilla RRT using Euclidean distance metric
```

```
2.  $J_o \leftarrow$ NN-TD(G) i.e use the vanilla RRT over the space to initialize the J values
3.  **while** goal is not reached
4.          $x_{rand} \leftarrow$ Sample(S);
5.          $x_{near} \leftarrow$ Nearest($x_{rand}$, V(G) , $||.||_J$ );
6.          $(x_{new}, a, r) \leftarrow$ Extend($x_{near}$, $x_{rand}$, S);
7.          **if** Not-Colliding($x_{new}$, $x_{near}$,S)
8.            **then** Connect $x_{new}$ to $x_{near}$
9.                  V(G) $\leftarrow$ V(G) U $x_{new}$
10.                 E(G) $\leftarrow$ E(G) U $(x_{new}, a, r)$
11.                 $J(x_{near}) \leftarrow (1-\alpha)J(x_{near} + \alpha(r + \gamma J(x_{new})))$
12.         J $\leftarrow$ RRT-Planning(G, J, number of iteration)
13. End of Algorithm
```

Number of iterations of planning could be modified as per the time available for planning during the execution phase of robot. Another important parameter to take care of is α which should typically be very small in stochastic environments for the value function to converge. γ value is domain dependant. The algorithm gradually builds a tree from the start to goal states in an online fashion. It uses RRT-based planning to improve the value function estimates by using the model already learnt. Even if dynamic environments the algorithm works well because the value function adapts to the moving environment at every time step and hence the around the obstacles the value function gives unfavourable values and guides the tree towards higher value goal regions.

If the environment has undergone drastic changes where replanning is required DYNA-RRTPI could be easily modified to include replanning efficiently. The modified algorithm is given as follows.

Algorithm *Replan-DYNA-RRTPI( )*

```
(∗ Online version: builds RRT and implements policy iteration using DYNA framework ∗)
1.  G$\leftarrow$ Build a vanilla RRT using Euclidean distance metric
2.  F $\leftarrow$ initialize to empty set of forest of disconnected trees. $J_o \leftarrow$ NN-TD(G) i.e use the vanilla
    RRT over the space to initialize the J values
3.  **while** Graph G doesnt contain goal
4.          **if** Graph is not empty
5.            **then** Prune(G,F)
6.            **else**  Intialize V(G) $\leftarrow \{x_{start}\}$
```

```
7.          $x_{rand}$ ← Modified-Sample(S,F);
8.          $x_{near}$ ← Nearest($x_{rand}$, V(G) , $||.||_J$ );
9.          $(x_{new}, a, r)$ ← Extend($x_{near}$, $x_{rand}$, S);
10.         if Not-Colliding($x_{new}$, $x_{near}$,S)
11.           then Connect $x_{new}$ to $x_{near}$
12.                V(G) ← V(G) U $x_{new}$
13.                E(G) ← E(G) U $(x_{new}, a, r)$
14.                $J(x_{near}) \leftarrow (1-\alpha)J(x_{near} + \alpha(r + \gamma J(x_{new})))$
15.         J ← RRT-Planning(G, J, number of iteration)
16.   End of Algorithm
```

Algorithm *Prune(Graph G, Forest F)*

```
(∗ Perfoms collision checks on all nodes and removes nodes that are not in $C_{free}$ ∗)
1.    for each node q ∈ V(G) and F
2.          if Not-valid (q)
3.            then remove q from G and F
4.                 split the tree at q and add subtrees to F
5.    if G is empty
6.      then Initialize G and add $x_{start}$ to V(G)
7.    End of Algorithm
```

Algorithm *Modified-Sample(S,F)*

```
(∗ Returns a point to which the RRT would try to expand to ∗)
1.    prob ← Random(0,1)
2.    if prob < $p_{goal}$
3.      then $x_{result}$ ← $x_{goal}$
4.      else  if prob < $p_{goal} + p_{forest}$
5.              then $x_{result}$ ← root of a randomly sampled tree from $F$ , the forest of disconnected trees.
6.              else  $x_{result}$ ← Randomly sample a point from the state space
7.    return $x_{result}$
8.    End of Algorithm
```

The above modification to the regular DYNA-RRTPI would use the previous successful trees regrow the trees at each instant. For every time step , the tree is inspected for any nodes in $C_{forbidden}$ and if it finds such nodes, the algorithm breaks the tree at that node and places the subtrees in the Forest F which would be used later on for sampling for points to extend the RRT. The algorithm for replanning can be made similar to both ERRT (put reference)or DRRT(put reference) based on the choice of parameter p_{forest}. In pruning stage if we maintain just a single tree instead of a forest of disconnected trees we would essentially get an analogue of ERRT where when selecting a node in *Modified-Sample* procedure we would have sampled a node from the single saved trajectory or waypoint cache we stored in pruning stage. The algorithm could be modified into an analogue of DRRT if instead of storing both the sub trees in pruning stage, we throw away the tree that is not connected to the start state. Hence in that case we would have to regrow from that sub-tree which is still connected to the start state. The suggested algorithm has the good properties of both ERRT and DRRT performs better that both ERRT and DRRT in replanning tasks (put reference of MPRRT).

5.2 Experiments

We evaluated our algorithm on discrete grid world domain, continous space without obstacles and continous puddle world domains.

CHAPTER 6

Conclusion and Future Work

We introduced a way of learning optimal solutions to motion planning problem in unknown non-stationary environments using RL and efficient replanning techniques. We assume just a minimum generative model which would tell if a given state is in C_{free} or not. Our algorithm builds a near optimal RRT in the state space with the learnt value function in continous state spaces. We have presented episodic, offline and online versions of our algorithm. We have showed use of nearest neighbour methods for tackling the value evaluation in continuous state space. We implemented our algorithm on continuous puddle world domain for an agent follwing kino-dynamic constraints using a local planner in the extend step of growing RRT. Our algorithm DYNA-RRTPI integrates good sampling based planning techniques into a framework interleaving directRL updates, model building and planning and thus solving problems in dynamic continuous unknown environments with efficient replanning techniques follwing kino-dynamic contraints of the agent and the environment.