GENERATING AN ADAPTIVE PATH USING RRT SAMPLING AND POTENTIAL FUNCTIONS WITH DIRECTIONAL NEAREST NEIGHBORS:

by

SANDEEP CHAHAL

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2018

To Mom and Dad for their sacrifices and immeasurable love.

Acknowledgements

Firstly, I would like to express my gratitude to my advisor Dr. Manfred Huber, for allowing me to conduct research and for his interminable guidance and support. I shall eternally be appreciative to numerous conversations furnishing me invaluable knowledge pertaining to myriad of fields not to mention robotics. These perorations were conveyed with vitality, candor and patience that assisted in essential assimilation of complex ideas.

I must thank my committee members Dr. Vassilis Athitsos and David Levine for taking the time to serve in my committee.

I would like to thank Shriiesh, Bhupender, Sourabh, Arun, Azmat, Shirin, Brian, Taoran, Shreesha, Renon who have been part of journey so far at LEARN Lab. I am also thankful to Akash, Anil, Jai who have been family away from home.

Lastly, I thank my parents for their quintessence and being mainstay of strength that has kept me going.

November 27, 2018

Abstract

GENERATING AN ADAPTIVE PATH USING RRT SAMPLING AND POTENTIAL FUNCTIONS WITH DIRECTIONAL NEAREST NEIGHBORS

Sandeep Chahal, M.S.

The University of Texas at Arlington, 2018

Supervising Professor: Manfred Huber

Planning algorithms have attained omnipresent successes in several fields including robotics, animation, manufacturing, drug design, computational biology and aerospace applications. Path Planning is an essential component for autonomous robots. The problem involves searching the configuration space and constructing a desired collision-free path that connects two states (the start and the goal) for a robot to gradually navigate from one state to another. In global path planners, the complete path is computed prior to the robot set off. Sampling based planning like Rapidly Expanding Random Trees (RRT) and Probabilistic Road Maps (PRM) used for single or multi-query planning has gained popularity since it is probabilistic complete and scales well to complex configuration spaces. However, re-planning (re-calculating the complete path) is almost unavoidable as path execution is inherently uncertain since a robot will deviate from the path due to slippage and other uncertainties in the environment. Local path planners which only calculate the path direction at the current location partially alleviate this problem since they do not pre-calculate a complete path and are thus less affected by deviations. However, local path planners are either not complete or, if they use navigation functions, do not scale well to complex environments.

To address this, this work presents an approach that combines the advantages of sampling-based global path planning with the benefit of a local, navigation function-based

path planning on the generated sample space. This reduces the need for re-planning if the robot diverges from the original path by utilizing a harmonic function potential field computed over the RRT sample set and directional nearest neighbors. The proposed work derives the samples in the environment using a simple randomized algorithm and systematically sampled obstacles that are hit during random sampling of the space. It therefore avoids sampling of the complete space. Additionally, samples generated during one planning phase can be exploited further for new goals in the environment.

Table of Contents

List of Illustrations

Chapter 1

INTRODUCTION AND RELATED WORK

1.1 Introduction

In recent years, research in the area of robotics has proliferated as use of intelligent robots has advanced from warehouses to aiding in surgeries. Of the key requirements of autonomous robots, navigation is the most well-known problem which inherently requires path planning. Consider a robot that needs to navigate in an environment that contains obstacles. The challenge for this robot is to move from state A to state B without colliding with obstacles. To address this, research has been proposed continuously in the area of path planning since the 1970s.

Humans always come up with a plan in all the situations they encounter throughout their life. Given an environment, a plan is a sequence of actions that can be taken to achieve a solution. One question that often arises is whether a plan is optimized. As criteria of optimization differ, the formulation of an optimized plan can be difficult. Even if the right criteria can be formulated, computation of an optimized plan might not be tractable. Instead, one strives to achieve a solution that is acceptable (or preferred) within the defined criterion. This preferred or acceptable solution is referred to as a satisficing plan. For problems that involve probabilistic uncertainty, the need for optimization arises more frequently than in deterministic domains, and utilization of probabilities is often reflected in terms of expected costs [1]. An optimal path does not strictly have be the shortest path to reach the goal as various criteria like time, expense, distance from obstacle etc. affect the notion of an optimal path. Cost is an important aspect of optimality and is often included in the optimality criterion. Some research neglects the cost to reach the goal [4]. Besides the optimality of a path itself, it is also important to consider the cost of the planning process and in particular the fact that a robot should not waste time due to re-planning of the path

or other factors when it gets stuck at any place during path execution. This consideration often leads to a tradeoff between the optimization of a path and the avoidance of re-planning costs.

In global path planning approaches like Roadmaps and Cell Decomposition, the complete path is calculated before the robot executes it. In such planners, determination of goal reachability is straightforward and known to a robot prior to path execution. Since a robot needs to operate in a domain with the presence of uncertainties, however, the robot will in general deviate from the path. This results in re-calculation of the entire path where the robot is immobile until re-calculation is completed. While the re-calculation is potentially significantly faster than the initial calculation since a correction of the existing path might be possible, it still introduces additional costs and potentially risks for the robot during re-calculation. Local path planners tend to avoid this problem as they only compute the next step instead of the entire path. However, local path planners are either not complete, i.e. it is not known whether the goal is reachable, or, if using navigation functions to avoid this, do not scale well to complex domains.

The approach presented in this thesis is focused on reducing the need of re-planning by combining aspects of global and local path planning. The proposed system samples the domain by using a simple bidirectional randomized algorithm that generates paths from start to goal through random walks in the domain. The resulting sample space is then augmented by replacing the existing path connections with new directional connections and by generating virtual samples to represent the unsampled, unknown part of the domain. These improved samples can then be interpolated using a navigation function approach in a way such that the strict following of the initially sampled path is not necessary. A harmonic function potential field is here computed over the space described by the samples to form a robust potential function that avoids formation of local minima

while providing flexibility in terms of the local paths that can be chosen. If a robot deviates from the original path, it computes the gradient of the potential of its current state by finding the directional neighbors and continues following the gradient to reach the goal. In other words, the robot does not have to strictly follow the original path and can still reach the goal by utilizing the potential value which is minimal at the goal. Besides addressing local deviations from the path, this allows the system to also include secondary path objectives at run-time as long as the system does not leave the part of the domain described by the generated samples. The sampling-based mechanism here generates a tractable representation of the relevant parts of the space while the potential function based on local point neighborhoods provides the task flexibility. In this form, the combined approach addresses part of the scaling challenge for the navigation function path planner by allowing it to be computed on a sample space covering the relevant parts of the domain, and also addresses the challenge for the sampling-based global path planner arising from deviations from the path and dynamic changes in the task objectives.

The main contribution of the approach presented in this thesis is to provide a way of calculating the potential value and generating virtual samples. The advantages of this method include reducing the need of re-planning in case of a robot deviating from the original path and if a new goal is within the space that has been already explored. In the latter situation, the values of the already generated samples over the domain can be interpolated with regards to the new destination to reach that goal. If a new goal lies outside this space, additional random sampling will be utilized to extend the sample set into relevant portions of the space that have not been explored.

The remainder of this thesis is structured as follows: The rest of this chapter reviews related work dealing with approaches used by this thesis and similar methods that focus on similar problem used in the field of planning. Chapter 2 introduces the

background, underlying notion and formalism of Path Planning. Chapter 3 describes the approach used by this thesis and presents the formalization. It also discusses the details and module descriptions for an experimental implementation of the proposed method. Finally, in Chapter 4 we present the conclusion and future work.

1.2 Related Work

The Path Planning problem is well known in the field of robotics [1]. Over the past decades the problem of motion planning has been studied to a large degree with the increase in use of robots. Discrete and continuous state space planning are two categories that exists in path planning. Discrete search methods such as A-star [27], Breadth-First Search or Bidirectional Search [28] are some of the methods used that are complete and thus guaranteed to find a feasible path in a finite set of states. Planning in a continuous state space has been explored since the 1970s and initially mostly referred to planning for a robot in 2D or 3D world, but more recently has moved into higher-dimensional, kino-dynamic state spaces.

Depending on the availability of information about the domain, path planning can be divided into two categories which are global path planning and local path planning. In a local path planner, only information about the next step of a potential path is computed without the derivation of a complete path to the goal. In contrast to local path planning, complete path information about the domain is provided in global path planning. This leads to obtaining a complete path from the start state to the goal state and thus contains direct information regarding reachability of the goal and path cost which are often not available in local path planning approaches. A wide range of traditional path planning methods have been developed an applied successfully, including deterministic Roadmap methods, such as Visibility graphs and Voronoi Diagrams, and Convex Cell Decomposition approaches

[???]. Many of these, however, do not scale well to higher dimensions and larger domains with complex obstacle geometries even if the relevant part of that space is locally confined around the robot system. Besides deterministic, complete methods, global path planners have also used Genetic Algorithms to derive a path which considers if the path is traversable and generates a new path if it is not traversable [3] . To address scalability issues in larger, more complex domains, sample-based planning has been exploited for global path planning and has been very successful for solving some problems in robotics. To decouple the motion planning approach from the geometric and kinematic models of the robot, most of these system use collision detection as “black box”[1], allowing them to perform sampling in a lower-dimensional space with the robot constraints leading to a rejection of some of the generated samples. In situations where most samples are still valid, this can dramatically reduce the complexity of finding a path while still allowing the planner to find a path through areas where the robot constraints are relevant.

Since first proposed in 1998 by Steven LaVelle, the randomized algorithm Rapidly-Exploring Random Tree (RRT) [5][6], considered as part of Monte Carlo methods to bias search in Voronoi regions, has attained great success and is widely used as the basis for path planning techniques since it easily handle the problems with obstacles, nonholonomic and kino-dynamic constraints. There have been various improvements and modifications of RRT over time [7]-[13]. Randomly exploring random graphs (RRG) [14] were proposed in 1996 which have a learning and a query phase. RRG’s initially construct a probabilistic roadmap graph whose nodes correspond to collision-free configurations and edges represent the feasible paths between these nodes and are computed using a simple local planner. Though RRT and RRG work in a similar fashion by generating random node and connecting them, RRTs are used to answer a goal single query whereas RRGs are used for multi query in a given state space. A tree version of RRG called RRT* preserves

asymptotic optimality of the RRG while maintaining a tree structure of RRT [14], and thus the computational advantages for path formation. Bidirectional RRTs have also been explored that use two RRTs, one rooted at the start and one rooted at the goal. The main advantage of this being that it will generally focus more efficiently on the relevant parts of the domain.

A variety of potential field methods have also been proposed where a potential field is applied to the goal and the obstacles and then the robot makes use of the resultant field from the obstacles and the goal. Artificial potential fields were initially used by Khatib [15] which involved the use of a potential field in Cartesian space. Newman and Hogan [16] extended this to Configuration-space. However, many potential methods suffered from local minima which resulted in a robot being trapped or stuck in a place. Various approaches have been suggested to address this issue. One of those notable approaches is to design the potential field as a harmonic function. Other approaches include treating local minima as obstacles [17], designing potential functions such that it avoids creation of local minima [18] etc. Harmonic functions govern a wide range of physical processes, including fluid flow or oscillation motions where the restoring force is directly proportional to the displacement and acts in the direction opposite to that of displacement [19]. In general, harmonic functions satisfy Laplace's equation [20] and therefore do not contain local minima.

To address the rate of convergence of RRT, potential function-based planning has been used in recent work in conjunction with RRT to guide the samples towards the goal [21] [22]. The resulting approaches proved to have more efficient memory utilization and accelerated convergence rate. The focus of the work in [22] is to change the function that connects two samples. Even though [22] makes the individual connection formation slower,

it links two samples more effectively, thus resulting in an overall acceleration of the planning process.

Focused D* [23], an extension of A* addresses the problem of path planning in a dynamic environment. Anytime Dynamic A* [24] presents a graph based planning and re-planning algorithm to produce bounded suboptimal solutions within the available time.

An important issue in practical path planning is to reduce the need of re-planning. To address this, the objective of this thesis is to propose a technique that can reduce the need of re-planning when a robot moves away from the original path that was derived by the planner. Founded on some of the recent works presented in this section, the proposed approach accomplishes this by initially using a bidirectional randomized algorithm and then computing a potential field harmonic function over those samples by utilizing directional nearest neighbors. As a result, if a robot deviates from path, the potential value that has been computed based on its neighbors can direct the robot towards the goa, traversing the area spanned by the freespace samples instead of strictly following the initially computed path. In addition to this reduction in re-planning in the case of deviations from the intended path, this approach also allows the samples generated during one planning phase to be utilized in later query phase when other goal locations are presented.

Chapter 2

TECHNICAL BACKGROUND

2.1 Configuration Space and Map Representation

Planning in a robot's workspace is hard due to the potentially complex structure of a robot and its kino-dynamic constraints. To overcome this, configuration space, also called C-Space, was first introduced by Lozano-Perez in 1987 [2]. A robot in C-Space (Configuration Space) is represented by a point and legal positions for a robot configuration space are denoted by C_{FREE} and illegal positions or obstacles are denoted by C_{OBS}. As shown in Figure 2-1.

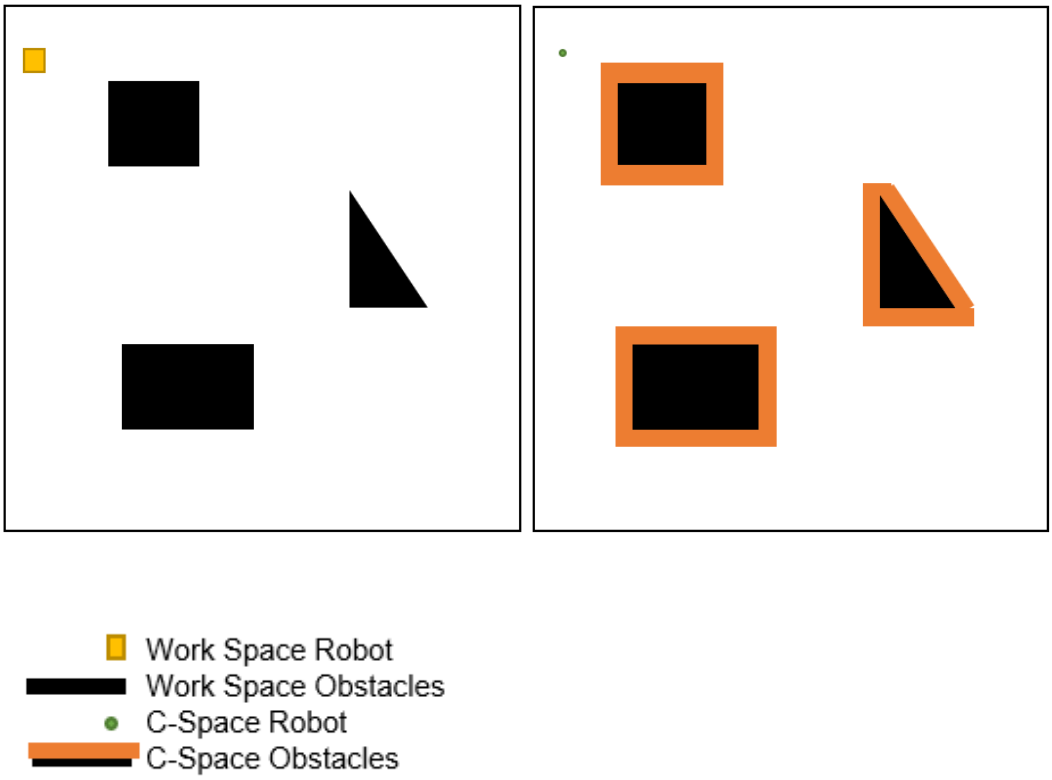

Figure 2-1 Workspace(left), C-Space (right)

$$C_{SPACE} = C_{FREE} \cup C_{OBS}$$

The key ingredient in constructing configuration space is extension of obstacles from workspace to C_{SPACE}. In other words, configuration space is the space of all possible configurations of a robot and is described as topological manifold. C-Space is obtained by marking any configuration in which any part of the robot collides with an obstacle as an obstacle configuration. In simple situations where the robot is a solid body, this corresponds to extending the obstacles by sliding the robot along the edges of obstacle regions, extending them by the size of the robot.

2.2 Path Planning Problem Formulation

The basic idea is that each situation of the world is a state x, such that $x \in X$ where X is the state space. The start state of the robot is denoted by X_S and the goal state is the location to be reached by a robot, denoted by X_G, where $X_S, X_G \in X$.

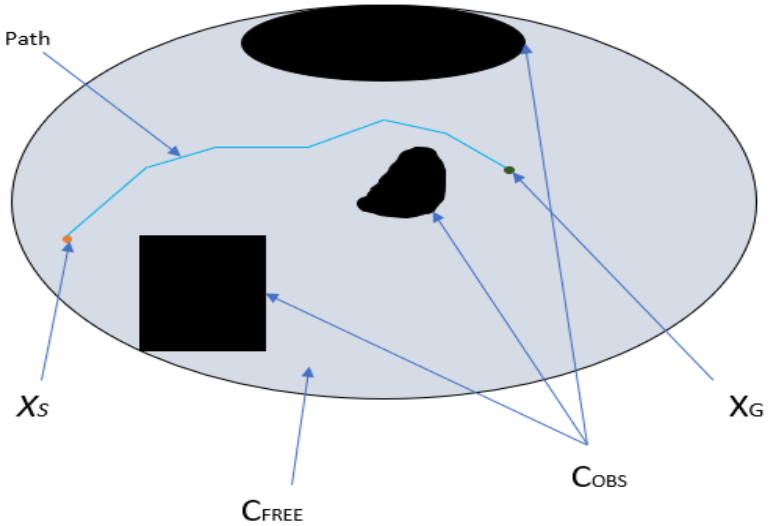

Figure 2-2. Configuration Space and Path

The path planning problem can be now formulated as finding a sequence of states leading from start state X_S, to goal state X_G in C-Space (Configuration space) such that all connections of states from X_S to X_G to be traversed by the robot belong to the free space as shown in Figure 2-2.

2.3 Sampling based motion planning

Sample based motion planners are not complete but are efficient. They find path from start to goal by using collision detection to make sure it avoids hitting any obstacle in the path. These algorithms are known as resolution complete or probabilistic complete which guarantees to find a solution if it exists, provided it runs for an infinite amount of time. Although the state space for motion planning is infinite, sampling-based planning can consider at most a finite number of states if it is to find a solution in given amount of time. As a consequence, it is important how samples are generated and evaluated.

RRT (Rapidly-exploring random trees) [21] is a sample based path planner which uses a specific exploration strategy where the objective is to get close to the goal within a threshold distance. RRT builds a tree which is rooted at a point from which grow branches in random directions. To extend the tree at a given point in time, they generate random samples and then grow the closes branch towards that point by some fixed step size (s) to a new sample which is considered instead of the initial random sample. To determine the node to extend using a newly drawn random sample, its distance is calculated with respect to each sample in the tree and then connected to the nearest sample. The base algorithm is given below:

$Function: Rrt$

$Input: initial\ config\ Xinit, number\ of\ vertices\ in\ RRT\ n, step\ size\ s$

$Output: Tree$

1. $T.init(Xinit)$
2. $T.init\ n = 1\ to\ n$
3. $Xrand < -rand\_config()$
4. $Xnear < -nearest\_vertex(Xrand, T)$
5. $Xnew < -new\_config(Xnear, Xrand, s)$
6. $T.add\_vertex(Xnew)$
7. $T.add\_edge(Xnear, Xnew)$
8. $Return\ T$

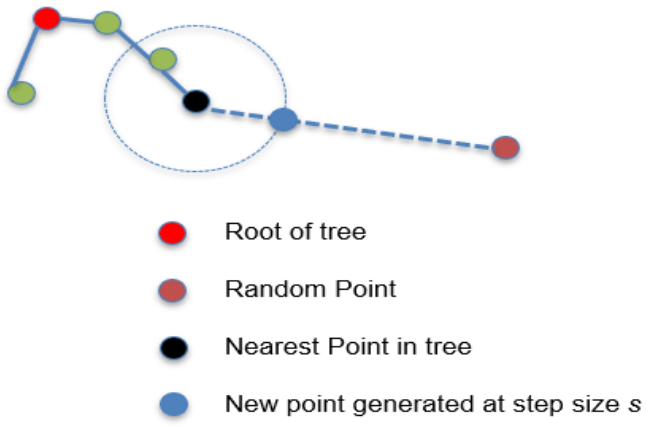

Figure 2-3. Adding New Node in RRT

2.4 Space Sampling

Sampling can be categorized into two categories as probabilistic and non probablistic sampling. probabilistic sampling is used to reduce bias in sampling. Hence,

providing an unbiased representation of the population. Probabilistic sampling is cost-effective and does not require any technical knowledge given the simplicity with which it can be done. Simple random sampling and systematic sampling are types of sampling techniques.

Random sampling can be done using a random number generator once you have the size of the population as shown in Figure 2-4. The probability density function for a continuous uniform distribution can be written as

$$f(a) = \frac{1}{y - x} \; if \; x \leq a \leq y$$
$$f(a) = 0 \; if \; a < x \; or \; x > b$$

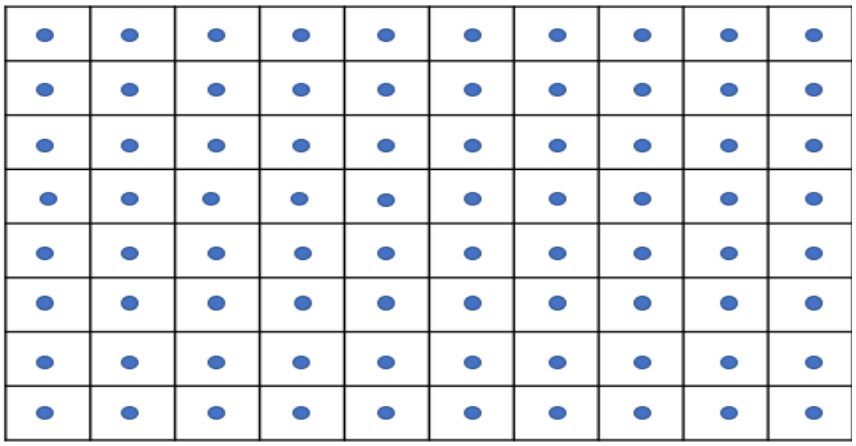

Figure 2-4. Uniform Sampling

A uniformly distributed random variable depends on the size of the interval but not on the location of the interval. Uniform random variable sampling techniques include sampling a random variable from a uniform distribution following the steps mentioned below:

1. Determine the sampling interval size of the population.
2. Get the upper and lower bound.

3. Use a uniform random number generator to generate a random number between lower and upper bound.

Systematic sampling is better for representing a population in a faster and simpler manner involving element selection from an ordered sample frame. A systematic sampling technique works by choosing the nth sample of the population as in Figure 2-5.

$$n = \frac{N}{x} \; where \; N \; total \; popultion \; and \; x \; is \; sample \; size$$

Systematic sampling works as follows:

1. Arrange population in a sequence (N).
2. Select the sample size (x).
3. Calculate the sampling interval n = N/x.
4. Select random number *r* between 1 to n including n.
5. Add the sampling interval x to the chosen random number to add the next member to a sample and repeat this to add the remaining members.

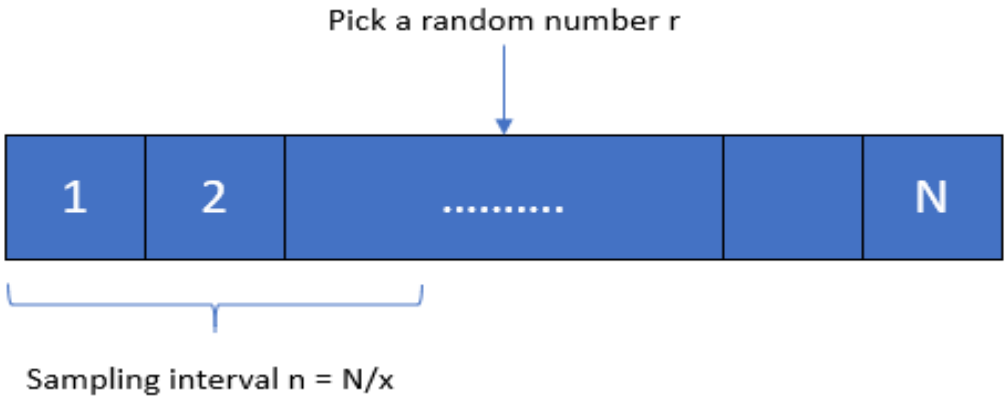

Figure 2-5. Linear Systematic Sampling

Linear and Circular systematic sampling work in a similar fashion except that the start point and end point in linear systematic sampling are distinct while it restarts from the start point in circular systematic sampling once the entire population has been sampled.

2.5 Potential Field

Initially proposed by Khatib [15], a goal and obstacle potential field represents two imaginary forces acting in the Configuration Space (C-space) called attractive force and repulsive force as in Figure 2-6. The attractive force is produced by the goal whereas the repulsive force is produced by obstacles. The direction to move for a robot can be computed by calculating the combined force at any point in in the C-space. The goal has minimum potential and obstacles have the highest potential in C-space where a robot follows gradient descent to reach the goal. In a mixture of potentials approach, the attractive force is

$$F_{attractive}(u) = -\nabla U_{attractive}(u)$$

and the repulsive force is

$$F_{repulsive}(u) = -\nabla U_{repulsive}(u)$$

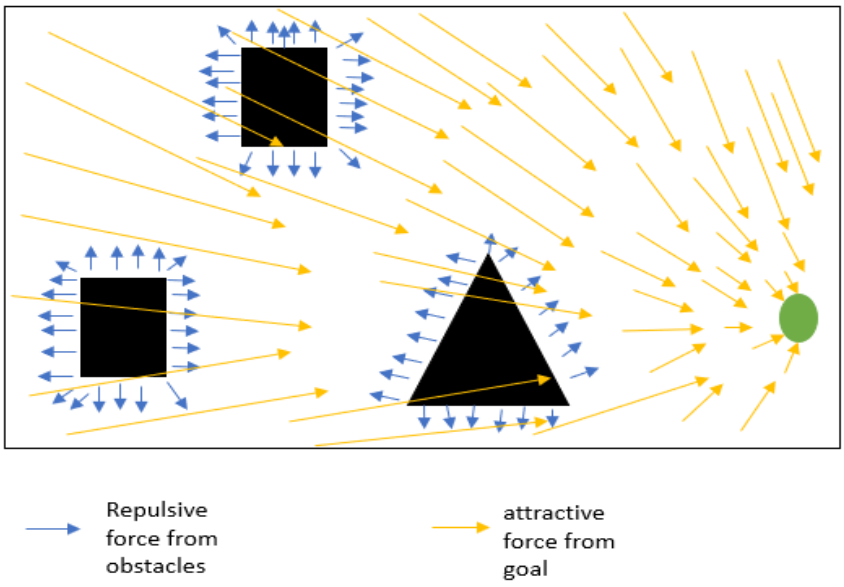

Figure 2-6. Imaginary Combined Forces

The resultant force is the combination of these two forces as in Figure 2-7

$$F_{total}(u) = F_{attractive}(u) + F_{repulsive}(u)$$

where the potential value of u is given as

$$U(u) = U_{attractive}(u) + U_{repulsive}(u)$$

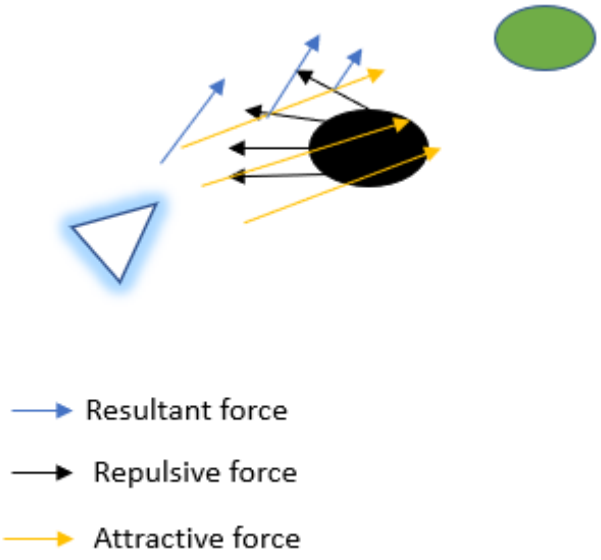

Figure 2-7. Resultant Force at Any Location

2.6 Harmonic Functions

As discussed in related work, potential fields, and in particular mixtures of potentials approaches, can have local minima where a robot can get stuck. Navigation functions can be used to generate potential fields that do not suffer from this issue. One class of these functions are harmonic functions. One property of a harmonic function is that it satisfies Lapace's Equation which means it is twice continuiously differentiable such that the sum of second order derivatives is 0.

$$\Delta \mathrm{u} = \sum_{i=1}^{n} \frac{\partial^2 u}{\partial {x_i}^2} = 0$$

which can also be written as

$$\Delta u = \frac{\partial^2 u}{\partial {x_1}^2} + \frac{\partial^2 u}{\partial {x_2}^2} + \cdots + \frac{\partial^2 u}{\partial {x_n}^2} = 0$$

An example with variables x,y z would be

$$\Delta u(x, y, z) = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = 0$$

Solutions to Lapace's equation can be computed by boundary conditions. One of the methods is to treat boundaries as obstacles and assign them the same value as that of real obstacles. In contrast to that the other method assigns boundary conditions such that the gradient of the potential field is tangential to the obstacle boundary which results in a robot staying close to obstacle surfaces [16]. Some of the techniques used for numerical calculation of the value of the harmonic function at each location [20] are Jacobi Iteration and successive over-relaxation, all of which require descretization of the environment into a finite number of locations to make computation tractable. Jacobi Iteration requires a higher number of iterations to converge and is more effective on SIMD architectures. It works by traverse through the state space and the value of a location is calculated as the average of its neighbours for relaxation. For a grid value which considers only four neighbours the Jacobi iteration for each grid can be written as

$$u^k(x, y) = \frac{1}{4}[u^{k-1}(x_{i+1}, y_i) + u^{k-1}(x_{i-1}, y_i) + u^{k-1}(x_i, y_{i+1}) + u^{k-1}(x_i, y_{i-1})]$$

where k is iteration number.

The Gauss- Seidel approach uses a similar iteration but neighboring values used for relaxation are partially from the previous iteration and partially from the current iteration.

$$u^k(x, y) = \frac{1}{4}[u^{k-1}(x_{i+1}, y_i) + u^{k}(x_{i-1}, y_i) + u^{k-1}(x_i, y_{i+1}) + u^{k}(x_i, y_{i-1})]$$

SOR relaxation also know as Successive Over Relaxation converges more quickly compared to the previous methods by anticipating the effect tha the change of the value will have on itself.

$$u^k(x,y) = u^k(x,y) + \frac{w}{4}[u^{k-1}(x_{i+1},y_i) + u^k(x_{i-1},y_i) + u^{k-1}(x_i,y_{i+1}) + u^k(x_i,y_{i-1})]$$

where k is the iteration number and w is a weight larger than 1, sometimes also referred to as the relaxation factor.

The iterations used in the above methods are repeated until the values at each location stop changing. In other words, until it converges.

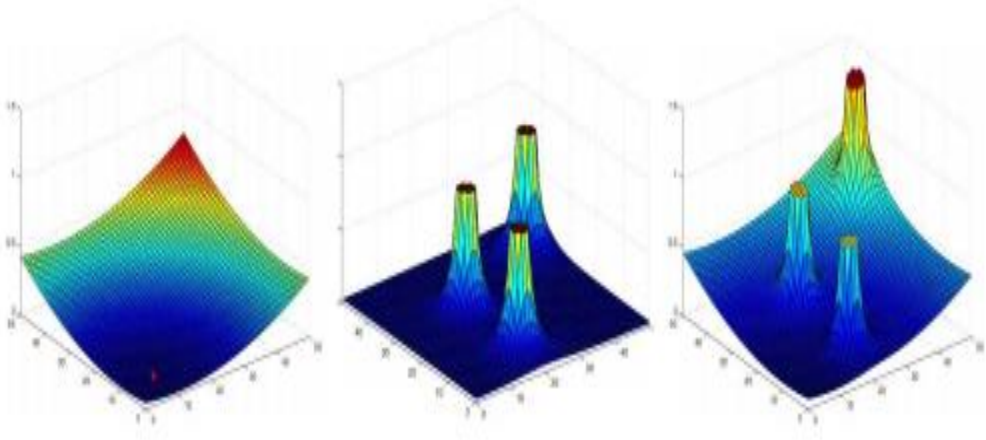

Figure 2-8. Attractive Potential (left), Repulsive Potential (center), Combined Potential (right)

2.7 Nearest Neighbor

Neareset neighbor is a proximity search that involves finding a point closest to the query point in terms of distance. The nearest neighbor search problem is well known in many fields, including computer vision, statistical classification, computaional geometry, DNA sequence, chemical similarity and sample based motion planning. Two types of approaches exits which are exact nearest neighbor and approximate nearest neighbors. The discussion in this section is about data structures used for efficient nearest neighbors, in particular kd-trees [29]. It will discuss space partitioning, building kd-trees, and finding

nearest neighbors within them as these techniques can be used for efficient nearest neighbor computation.

The time for evaluation of nearest neighbors in a naïve implementation with a set of samples and without additional data structures is linear in the number of vertices for each computation. Running time increases with the number of vertices in tree. However, increasing vertices improves approximation quality. KD-tree is a data structure to efficiently represent multidimensional data in a binary tree. KD-tree construction uses either sliding distance or a median node to split the nodes in the binary tree based on the location of the nodes. Construction time for a KD-tree is O *(dn log n)* time, where d is the number of dimensions of the underlying locations and n is the number of points.

Example of construction of a KD tree where d =2 which is (x, y) is given below:

1. Find the minimum and maximum along the x axis and assign this and all data points to the root node.
2. For the node:
    a. Determine the minimum, maximum, and median of the x-axis values in the node.
        i. Assign the points with values below the median to the left child node and the ones with values larger than the median to the right child
3. For each of the children which contains more than one data point:
    a. Determine the minimum, maximum, and median along the y axis of all data points in the node
        i. Assign the points with values below the median to the left child node and the ones with values larger than the median to the right child.

4. For each of the children which contains more than one data point go to step 2
5. Repeat steps 2-4 until all the data is placed in the leaf nodes.

Searching in a KD-tree for nearest neighbors is relatively efficient since it allows to eliminate large portions of the data points relatively rapidly. To find neighbors, the tree is traversed for the given data point coordinates and only the partitions surrounding the partition containing the data point have to be examined for potential neighbors. Query for the node is here performed by branching based on the medians stored in the nodes.

Chapter 3

PROPOSED APPROACH AND IMPLEMENTATION

3.1 Architecture Overview and Details

In this work we propose an approach that reduces the need of re-planning if a robot diverges from the original path. The architecture contains four sub-sections, a depiction of which is as given in the architecture layout in Figure 3-1.

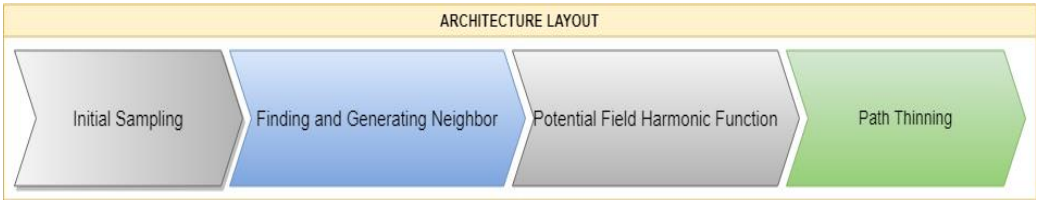

Figure 3-1. Architecture Layout

The planner assumes that the map of the environment or the state space with all the information, including the obstacle states and the start and goal states, is available. It then generates samples in the state space using a simple version of the randomized algorithm RRT and builds two random trees rooted at the start state and the goal state respectively which guarantees to never generate a point on an obstacle. The path is found when these trees hit or connect with each other. The randomized algorithm is stopped once a path to the goal is found. The nodes in the resulting path tree are then treated as free space samples that describe the neighborhood of the found path. In addition to the freespace samples, the algorithm systematically samples all obstacles that are encountered to produce an obstacle samples. Figure 3-2 shows the details of this sampel generation part of the proposed approach.

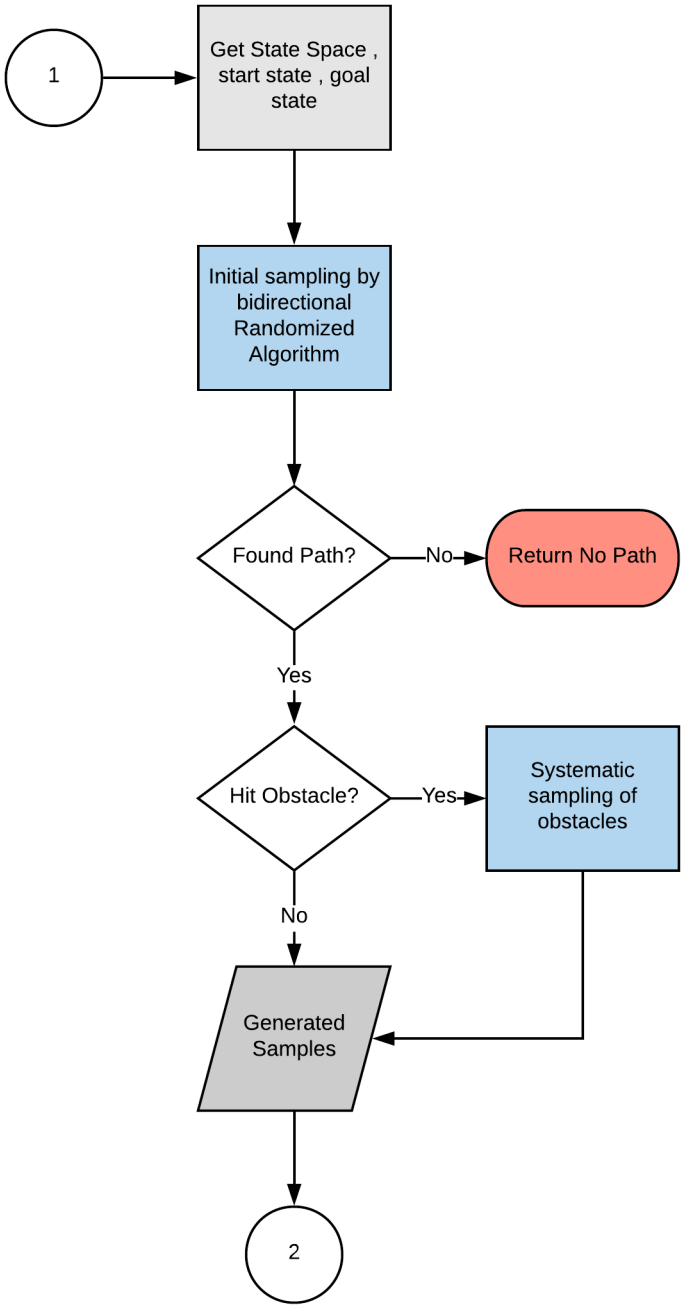

Figure 3-2. Detailed Overview of Part-1 of the Architecture

Once the sample set has been generated, all the connections produced by the trees in the previous step are ignored and instead a nearest neighbor connectivity structure is established. For this, The points are then stored in a data structure (KD-tree) for nearest neighbor query and one nearest neighbor is found for every node in each of a pre-determined set of directions within an upper distance threshold η. The closest neighbor is considered in each direction and if no sample is found a virtual sample is generated and preserved in the data structure. The next step involves propagating potential values for each of these samples using Successive Over-Relaxation until it converges to form a harmonic function potential. The last step involves reducing the number of samples in the free space that have been generated by the first step of the process. A sample, *S,* is removed if all neighbors of *S* are free samples and all the samples which have *S* as neighbor have another free sample within threshold η in the same directions. This second part of the approach effectively overlays the space described by the samples generated previously with a harmonic potential, establishing the ability to generate paths that do not go strictly through sample points. This process is detailed in Figure 3-3.

This approach results in an area for a robot to reach the goal by following the gradient of the potential at any point which reduces re-planning as at any given point if the robot diverges its potential value can be calculated by finding its neighbors which results in the gradient for a robot to follow. In the following the approach is described in more detail.

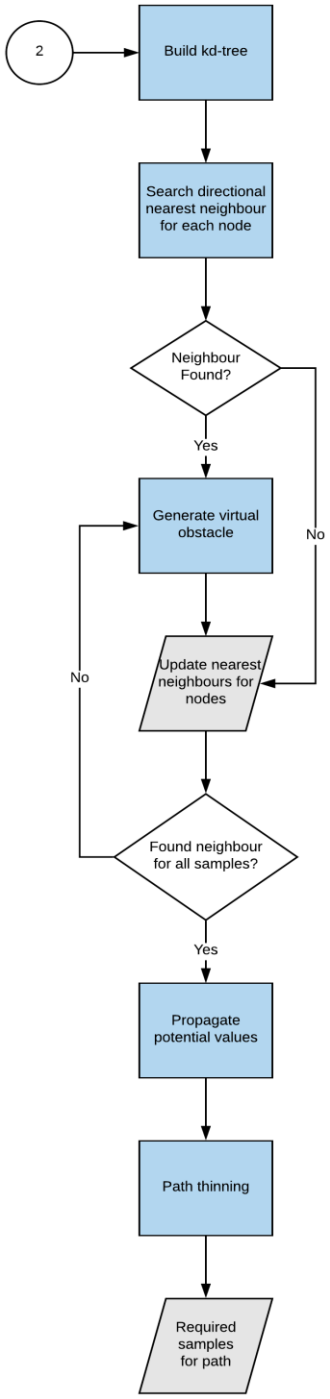

Figure 3-3. Detailed Overview of Part-2 of the Architecture

3.2 Initial Sampling

The initial sampling of the space is done using a simple bidirectional randomized RRT algorithm that uses a front and a back tree to explore relevant parts of the configuration space. The front tree is rooted at the start node and the back tree is rooted at the goal state. A new random sample is generated from a uniform distribution over the space using a density function $f(a_i)$ for each dimension, i, of the configuration space:

$$f(a_i) = \frac{1}{i_{max} - i_{min}} \; if \; i_{min} \leq a_i \leq i_{max}$$

$$f(a_i) = 0 \; if \; a_i < i_{min} \; or \; a_i > i_{max}$$

The next step is to find the closest sample (*CS*) for this random sample in the front tree by computing the distance of each node from the tree and keeping track of the node found at the minimum distance so far. The distance is calculated as the Euclidian distance of the two points *p* and *q* which is given by

$$dist(p,q) = \sqrt{(p_{\mathrm{n}} - q_{\mathrm{n}})^2 + \cdots + (p_2 - q_1)^2 + (p_1 - q_1)^2}$$

which connects two points by a straight-line segment given as in 2D [26]

$$dist(X,Y) = \sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2} \; where \; X = (x_1, y_1), Y = (x_2, y_2)$$

which in 3D can be given as

$$dist(p,q) = \sqrt{(p_3 - q_3)^2 + (p_2 - q_2)^2 + (p_1 - q_1)^2}$$

A new sample *NS* is then generated in the direction of this point from the closest point at a distance of *s*, which is called the step size. This is illustrated in Figure 3-4.

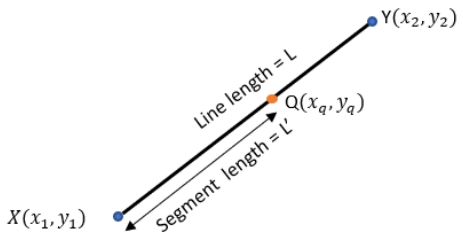

Figure 3-4. Point on Line

In Figure 3-4 the actual point, $Q(x_q,y_q)$, is given by

$$Q(x_q,y_q) = \left(\left(1 - \frac{L}{L'}\right)x_1 + \frac{L}{L'}x_2\right), \left(\left(1 - \frac{L}{L'}\right)y_1 + \frac{L}{L'}y_1\right)$$

$$where\ 0 < t < 1$$

This new sample is then added to the front tree as a child of CS. A new sample is then generated with respect to the back tree and similar steps are repeated. This process is repeated until a path is found. In general, a maximum number of nodes is defined, and the process is repeated until it hits that limit. If it hits the maximum limit and no path is found it is assumed that there exists no path.

If the RRT hits any obstacle during the process of building the tree, the surface of the encountered obstacles is systematically sampled as mentioned in Section 2.4.

3.3 Nearest Neighbor and KD-tree

All the freespace samples and obstacles sample generated are then used to construct a KD-tree. After construction of the KD-tree, each sample is queried for its nearest neighbors in $2 * 2^d$ directions to establish spatial connectivity for the potential field calculation. In a 2D Configuration space this leads to connections in 8 directions, resulting in 8 closest neighbors within the threshold range *H.* If no free sample or obstacle sample in a given direction is found within the distance threshold, it is considered to lie on the edge

of the known workspace in that direction and a virtual obstacle is generated in that direction at threshold = η. At the same time, it is verified that the space spanned by the neighbors does not contain an obstacle and if an obstacle is encountered in that space, it is systematically sampled, those samples are added, and the process continues. The resulting samples are then stored in a data structure that also contains all the neighbors.

An example of a query sample is as shown in Figure 3-5 left where the query sample is orange with a threshold radius η showed with a red circle from the sample.

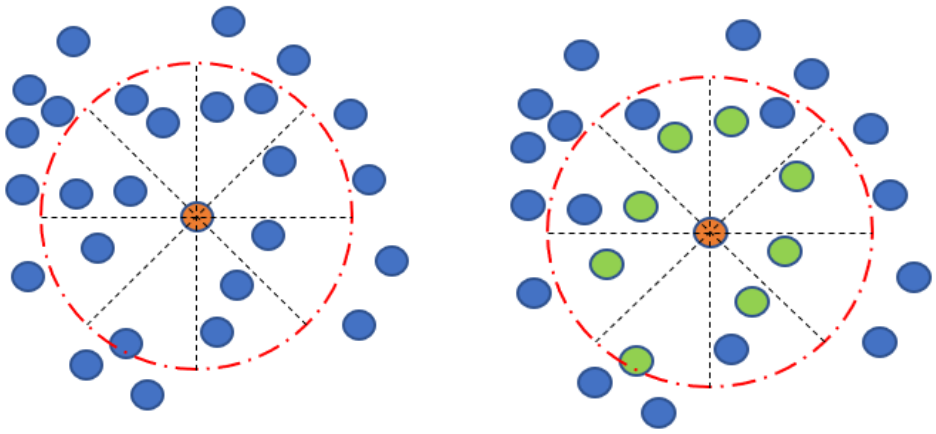

Figure 3-5. Query Sample (orange, left), Nearest neighbor in each direction (green, right)

The picture on the right shows the neighbors found in each direction in green. In the case when no sample exist in a direction, virtual obstacle (workspace boundary) samples are generated in that direction at distance η as shown with the yellow samples in Figure 3-6.

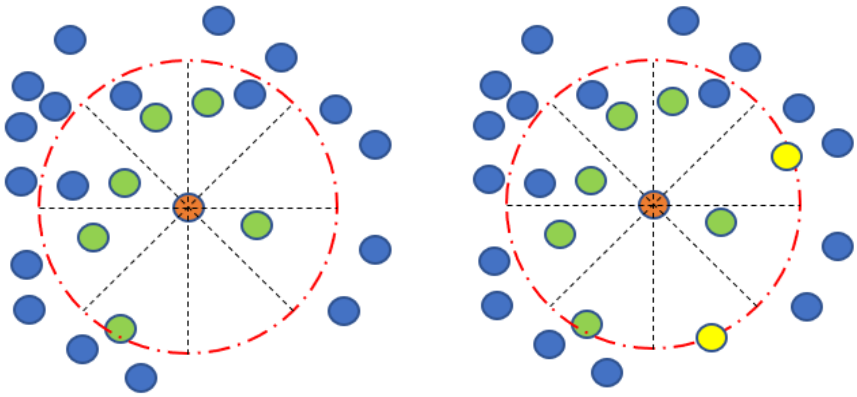

Figure 3-6. Query Sample and neighbors in each direction(left), Generated virtual obstacles (yellow, right)

The KD- tree construction used here is based on a sliding distance and builds an unbalanced tree [29] . An example of how the KD- tree is constructed from data samples is as shown below.

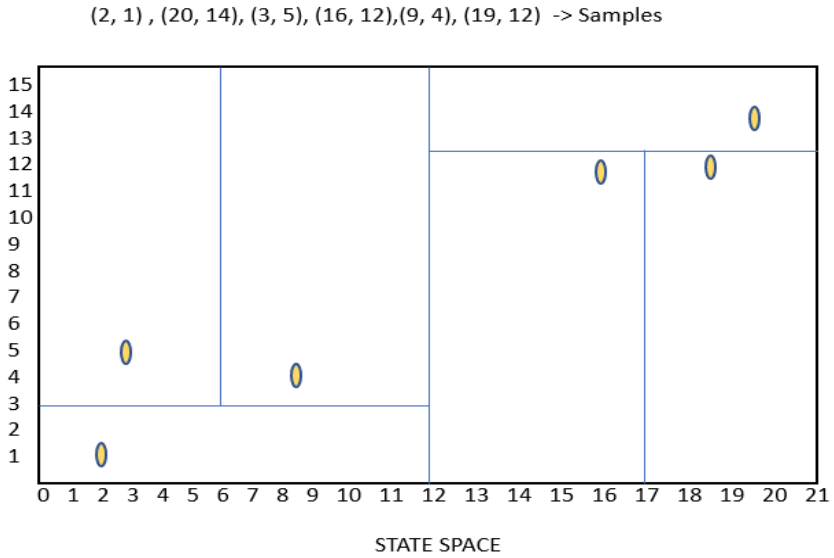

Figure 3-7. Space Partition KD-Tree Based on Sliding Distance

Figure 3-7 shows the space partition based on the sliding distance and Figure 3-8 shows a KD-tree from the space partition of the state space. Leaf size in this example is 1 show as yellow in Figure 3-8.

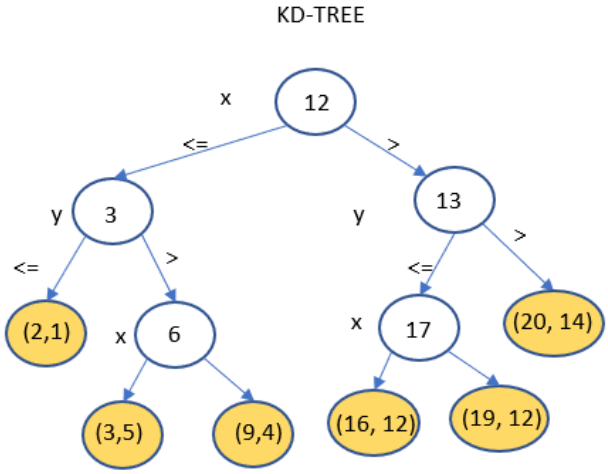

Figure 3-8. KD-Tree Based on Space Partition

3.4 Potential Field Harmonic Function

After all the samples have been queried for neighbors, potential values are propagated with the goal being at a minimum and the obstacles being at a maximum. Each value is then updated by

$$U(u) = \frac{\sum_{i=1}^{n} W_i \ U(i)}{\sum_{i=1}^{n} W_i}$$

where the weight W_i of each sample in the equation is related to its distance from the point u with $W_i = \frac{1}{d(u,i)^2}$ and $U(u) = potential\ value\ of\ sample$, $U(i) = value\ of\ neighbor\ sample$. In other words, the value of each sample is the weighted average of the sample values of its nearest neighbors. The potential value of the sample is affected more by the potential value of the samples that are near to it and affected less

by the potential value of sample that is far from it. This process is repeated until it converges which means until values stop changing.

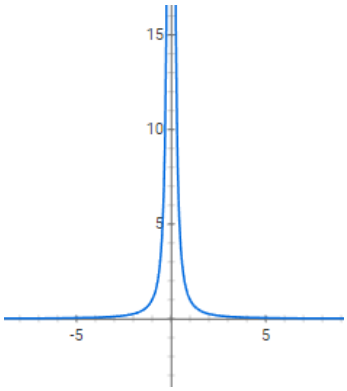

Figure 3-9. Potential Field effect of a neighbor based on distance $1/X^2$

3.5 Thinning Path

Since the sample density in the space is uneven and, in particular, the density along the sampled path is very high, the sample density in many freespace regions could be reduced without significantly affecting the potential. To do so, some of the unnecessary samples in free space can be removed. This approach uses two criteria to remove a sample; in particular, a sample *S* can be removed only if

1. All the neighbor sample of sample *S* are free samples.
2. All the neighbor samples of *S* contain all free neighbors.
3. All the samples that contains *S* as neighbor have another free sample in that direction within distance threshold *H*.

A KD-tree for this reduced sample set can be constructed with all obstacle samples and remaining free samples after thinning out the path. A robot can now reach the location of the goal by simply following the gradient. If a robot diverges from path its potential value

can be calculated by querying the neighbors using the KD-tree which is O (k logN) time and navigate along the negative potential gradient.

3.6 New Goal

If a new goal is generated and lies in the space which has been already explored, the harmonic function can be used to recalculate potential values for each sample and the resulting values can be interpolated across the samples to allow the robot to navigate by following the negative potential gradient to the goal.

If the new goal in in an area which is unexplored, a new biased tree can be built from the goal which generates a new random goal tree and is expanded until it hits the samples which that have been already generated. Then the samples for this tree are added to the KD-tree and the potential field is recalculated to drive the robot to the new goal.

Chapter 4

IMPLEMENTATION AND EXPERIMENTS

4.1 Implementation

The experiments introduced here assume that the world map, start sate, goal state and obstacles are available to the robot. For the purpose of execution this map is generated by an algorithm. The world or map for experimental purpose is generated by first setting the size of the environment. Then the obstacles in the map or domain are generated and returned to the simple bidirectional randomized path planner. It is worth mentioning that this C_{SPACE} is not generated by the robot but instead is randomly generated by $generate\_map()$.

$Function: generate\_map$

$input: space\ size\ S, number\ of\ obstacles\ n$

$generate\_map(S, n)$:

1. $map < -generate\_space(S)$
2. $map < -generate\_obstacles(map,\ n)$
3. $return\ generated_{map}$

Once this data is available, the next step is to utilize a bidirectional algorithm to generate the initial samples to find a path between the start state and the goal state. This randomized bidirectional algorithm generates two different sample sets originating from start and goal states. These samples may or may not be biased towards converging either with the root of another sample set or towards the last sample generated in the other sample set. These two sample sets are stored in a tree data structure. The function returns

no path found and exists, if no path is found within a specified threshold of maximum number of samples generated. The algorithm for this is given below.

$Function: bidirection\_randomized\_algorithm$

$input: start\ state\ X_S, goal\ state\ X_G, tree\ roote\ at\ start\ state\ T_s, tree\ roote\ at\ goal,$

$\cdot\ step\ size\ \blacktriangle, maximum\ number\ of\ nodes\ ⋔, state\ space\ S,$

$goal\ threshold\ €$

$output: T_S, T_G$

1. $while\ n \leq ⋔:$
2. T= T_S
3. $X_{RAND} < -random\_configuration(S)$
4. $X_{NEAR} < -nearest\_vertex(X_{RAND},\ T)$
5. $X_{NEW} < -new\_config(Xnear,\ Xrand,\ \blacktriangle)$
6. $hit\ =\ T.add\_vertex(X_{NEW})$
7. If hit: systemetic_sample(obstacle)
8. $T.add\_edge(X_{NEAR},\ X_{NEW})$
9. $if\ goal\_check(T_S, T_G)\ is\ true: return\ T_s, T_G$
10. $Swap\ T_s, T_G$
11. $exit(no\ path)$

With the generated samples a KD-tree is built to store the information such that it can be queried quickly when needed. For each point in the sample set, the function searches directional nearest neighbors and generates virtual obstacles. If there are no nearest neighbors found it updates the nearest neighbors for nodes. Continuing within this loop it generates and update neighbors for each sample in the generated sample set.

$Function: nearest\_neighbors$

$input$: Samples S (from $T_s, T_{G,}$), threshold Ħ,

$data\ structure\ with\ neighbors\ and\ potential\ value\ D_N$

$output$: $data\ structure\ with\ neighbors\ and\ potential\ value\ D_N$

1. $K_D = build\_kdtree(S, balanced = bool, leafsize = 1)$
2. $for\ all\ free\ samples\ in\ S$:
3. $s_N <- K_D\ .find\_kdir\_neigbhor(s)$
4. $if\ neighbor\ not\ found\ in\ any\ dir$)
5. $hit = generate\_virtual\_neighbor(s_N,\ dir,$ Ħ)
6. If hit: systemetic_sample(obstacle)
7. $D_N\ .add\_sample(s_N)$
8. $return\ K_D\ , D_N$

Given the set of samples, there exists a potential field from the start state to the goal state which the robot can follow to reach the goal. The issue with this potential field is that if looked at strictly from the perspective of the path generated in the RRT formation, it may contain local minima and thus would not lend itself to navigation. To address this issue, this approach utilizes a harmonic function to propagate potential values in a way that completely removes local minima from potential field. For this, the potential value for each freespace point is iteratively updated as the weighted average of its neighbors and this way the potential function is relaxed until it converges.

$Function: harmonic\_function$

$input: D_N$, goal_state()

$output: D_N$

1. $Untill\ converges$
2. $for\ all\ free\ samples\ in\ S$:
3. $if\ goal: s_N(value) = min$
4. $else\ if\ obstacle: s_N(value) = max$
5. $else\ s_N(value) = s_N(value) + alpha\left(\frac{\sum_{i=1}^{n} W_i s_N}{W_i} - s_N(value)\right)$
6. $return\ D_N$

Now there the system has a potential field which leads to the goal. In an attempt to further optimize this path, this approach utilizes a method for path thinning. This leads to a smooth path connecting start and goal state, which the robot can now follow.

$Function: thin\_path$

$input: D_N$

$output: D_N$

1. $for\ sample\ in\ D_N$
2. $if\ \ all\ neighbors\ are\ free\ for\ sample\ and\ neighbors\ of\ neighors\ are\ free$
3. $for\ samples\ that\ have\ sample\ as\ neighbor\ con\ be\ replaced\ by\ free\ samples$
4. $remove\ sample\ from\ D_N$

The next location for the robot at any time step is calculated by

$Function: next\_loc$

$input: D_N, current\ location\ of\ robot$

$output: D_N$

1. $for\ current\ location$
2. $if\ \ current\ location\ on\ path$
3. $next\_loc = -\diamond D_N\big(argmin(neighbors)\big).value$
4. $return\ nextloc$
5. $else$
6. $return\ gradient\_next\_loc()$

$function: gradient\_next\_loc$

$input: K_D, current\_location, gradient\ step\ D_N, goal\ state X_G$

$output: next_{loc}$

1. $K_D\,.find\_kdir\_neigbhor(current\_location\ )$
2. $next_{loc} = -\diamond D_N\big(argmin(neighbors)\big).value$
3. $return\ next_{loc}$

When the robot navigates to a given goal state, it might receive a new goal state. In this case, the function new_goal checks if the new goal state resides within the already sampled space. If not, a biased randomized algorithm is invoked from the new goal state that samples the space until it finds any of the old samples. Once it finds a path, the harmonic function is reinvoked and potential values for each sample are re calculated in a similar fashion.

$function: new\_goal$

$input: D_N, K_D,$ new$_goal_state\ X_{NG}, biased\ b$

$output: D_N or\ N_T$

4. $in\_explored\_space\ = K_D.find\_kdir\_neigbhor(X_{NG})$
5. $If\ in\_explored\_space$:
6. $D_N.add(X_{NG}), K_D.add(X_{NG})$
7. $harmonic\_function(D_N, X_G)$
8. $else$
9. while max $number\ of\ node$
10. $R_{RAND} < -random\_uniform\_number(0,1)$
11. $if\ R_{RAND} < b: X_{RAND} =\ robot\_curr\_location$
12. $else: X_{RAND} < -random\_configuration(S)$
13. $X_{NEAR} < -nearest\_vertex(X_{RAND},\ T)$
14. $X_{NEW} < -new\_config(Xnear,\ Xrand,\ s)$
15. $hit =\ T.add\_vertex(X_{NEW})$
16. $If\ hit: systemetic\_sample(obstacle)$
17. $if\ goal_{found}\ return\ N_T$
12. $exit(no\ path)$

4.2 Experiments

The environment for the experiments is created in a 1000x1000 2-dimensional continuous space. Some example location samples in this space can be (30, 30), (12.678, 16.00001458). To illustrate this environment, a visualization of it is shown in Figure 4-1 where the right half of the picture depicts the domain or environment for planning which contains random obstacles as shown in blue. The right half of the picture shows the uniform samples generated while building RRT during initial phase.

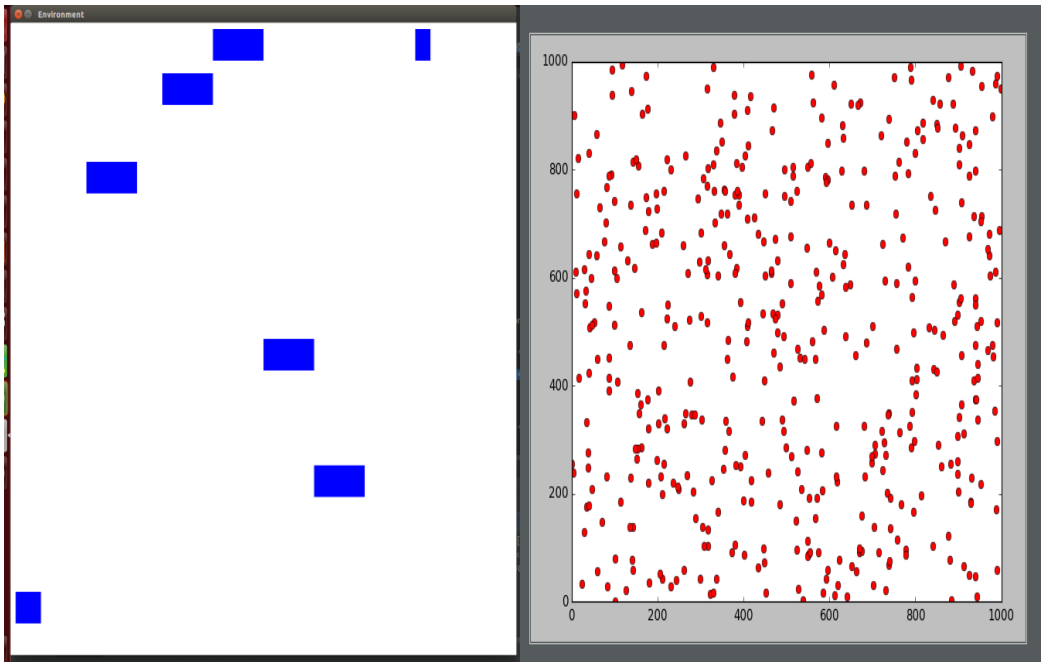

Figure 4-1. An example of environment set up (left), random sample generated (right)

Figure 4-2 represents a built Bi-directional RRT in the domain (C-space). The green points indicate the locations on obstacles hit during exploration of the Configuration space by the bidirectional RRT which are then systematically sampled into obstacle samples as shown in Figure 4-3.

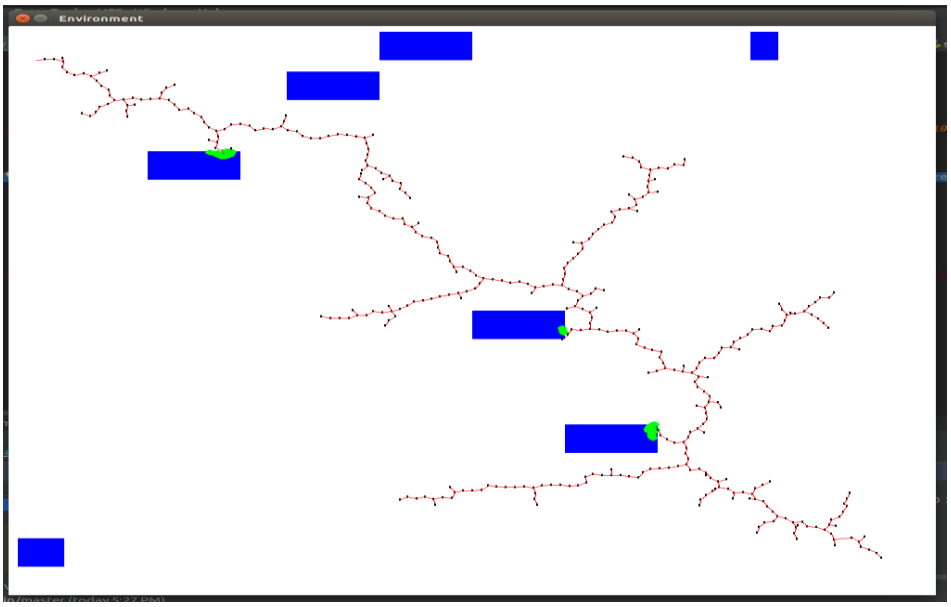

Figure 4-2. Path from Bidirectional RRT with Obstacles (Blue) and Obstacle Hits(green)

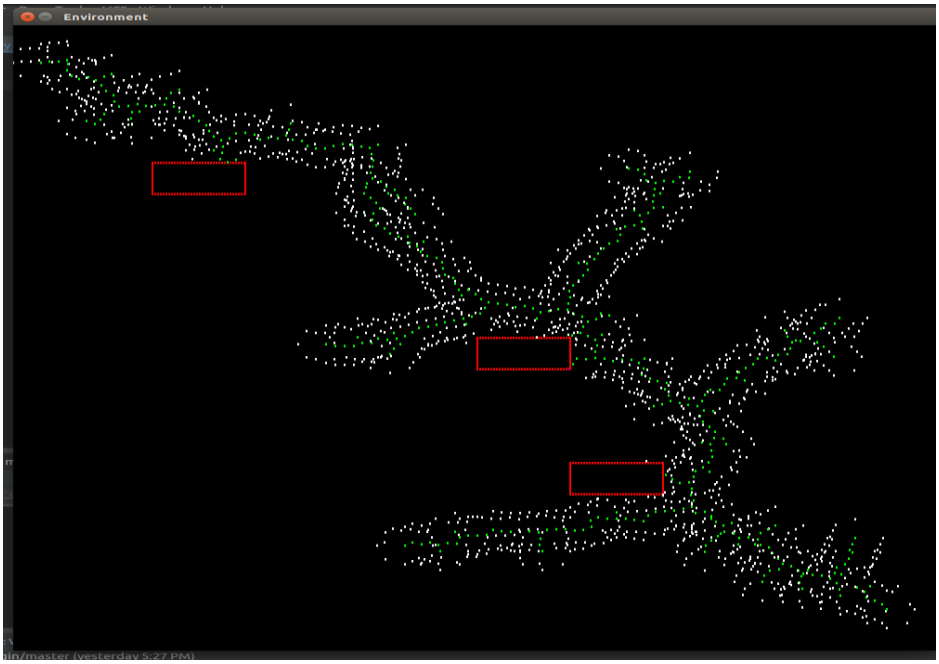

Figure 4-3. Systematic Samples on Obstacle Surface (Red), Virtual Obstacles (White) and Original Path Samples (Green)

Once the RRT is built and surfaces of obstacles that were hit during the RRT building phase are systematically sampled, all these samples are used to construct a kd-tree as mentioned in Section 3.3. The resulting samples are then stored in the data structure that also contains neighbors as shown in Figure 4-3, where white points indicate virtual obstacles which represent the boundary of the space described by the available samples. The virtual obstacles (workspace boundary) of one sample does not affect other samples and is thus only linked to a single freespace sample.

Figure 4-4and Figure 4-5 show an example of the change in the potential of a sample set derived on the same environment during the relaxation process at iteration 1 and iteration 100, respectively. The goal is at left bottom.

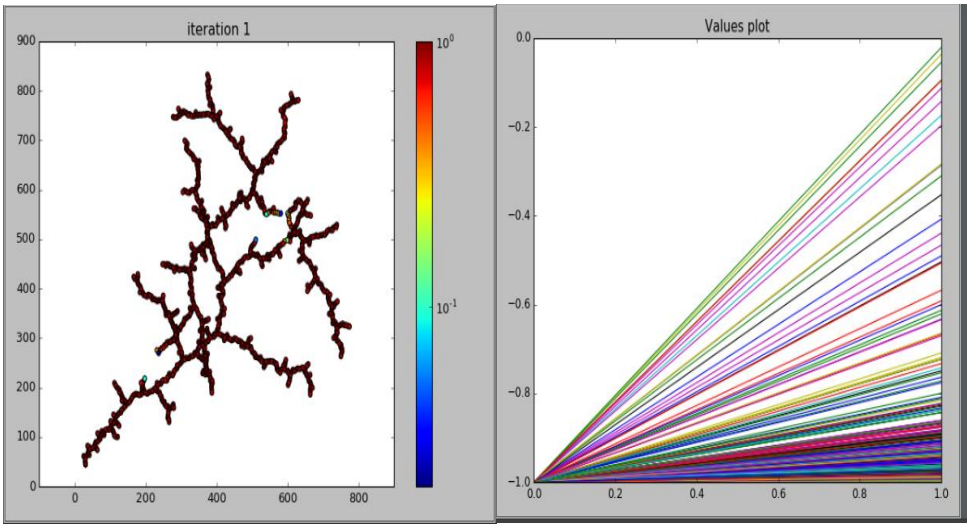

Figure 4-4. Freespace sample values at iteration 1 with the goal at the left bottom and the start at the right top (left), and change in values (right)

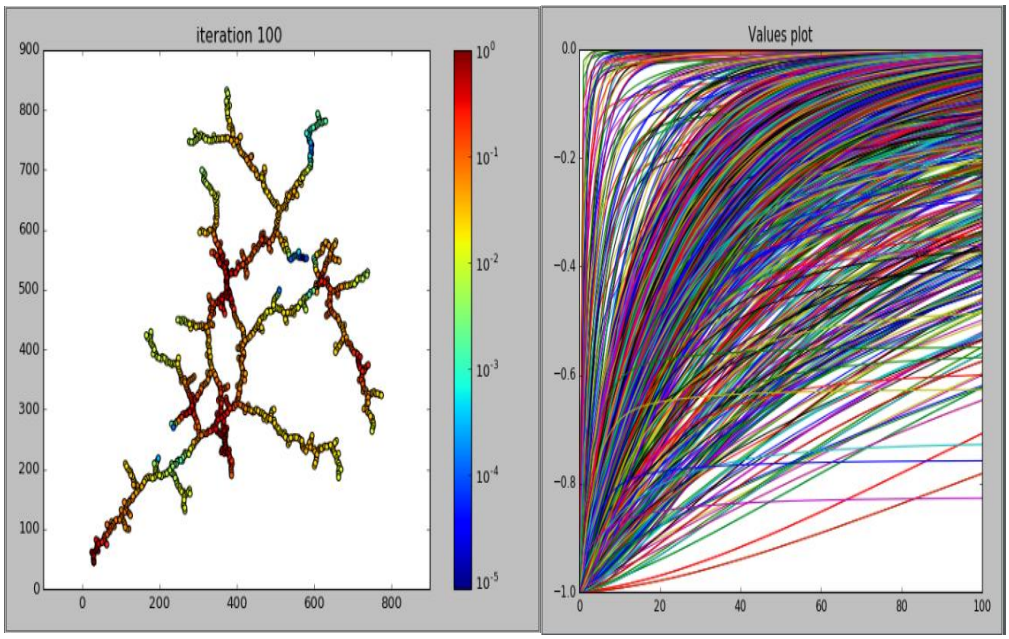

Figure 4-5., Freespace sample values at iteration 100 (left) and change in values (right)

The values are stable after 3000 iterations and, as can be observed in the value plot of Figure 4-6 the value function behavior is exponential as mentioned in Section 3.4. This figure also represents the gradient along the path after the values are stable. Note that the values are plotted in linear scale. Since the function values resulting from a harmonic function relaxation behave like an exponential, some of the differences in the gradient values, specifically near the start location, are not easily discernable.

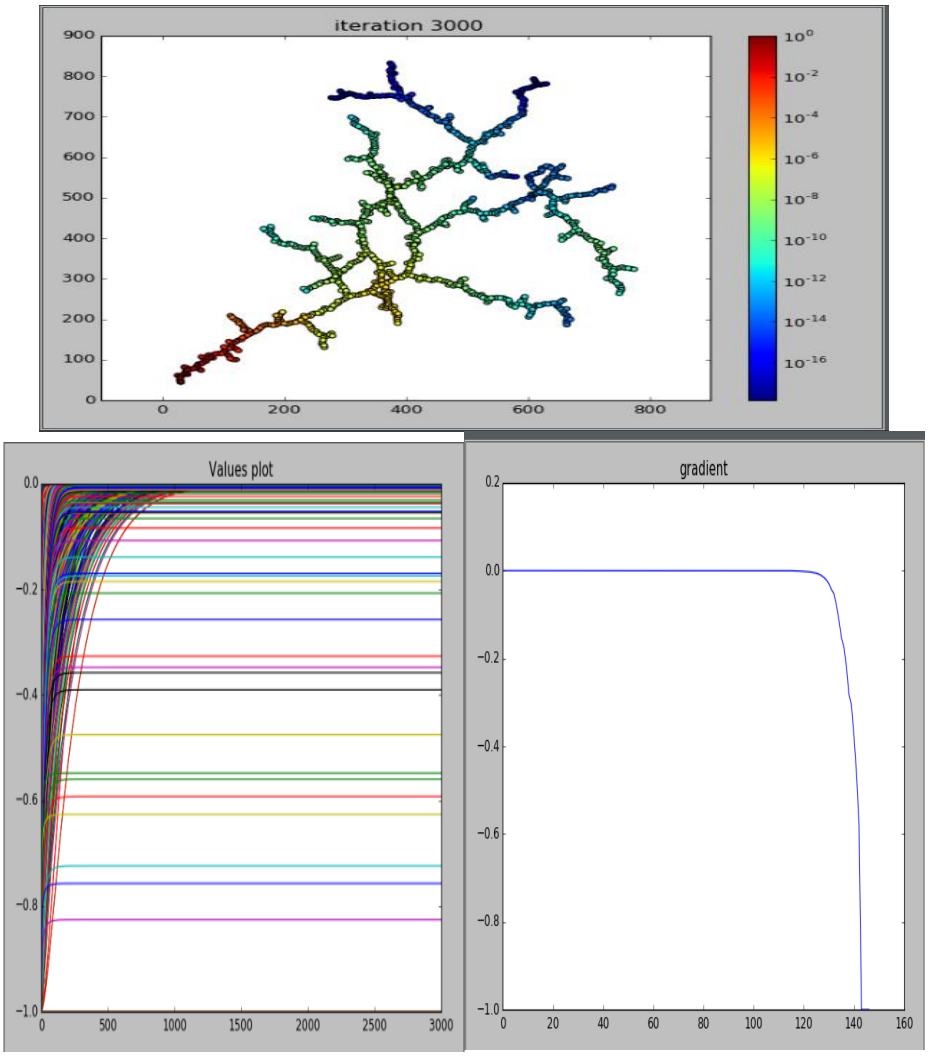

Figure 4-6. Freespace sample values at iteration 3000 (top), change in values (bottom left) and gradient of a random path (bottom right)

A similar example with a different random path is as shown in Figure 4-7 and Figure 4-8 at 4, 80, 1000 and 3000 iterations.Figure 4-7. At iteration 4

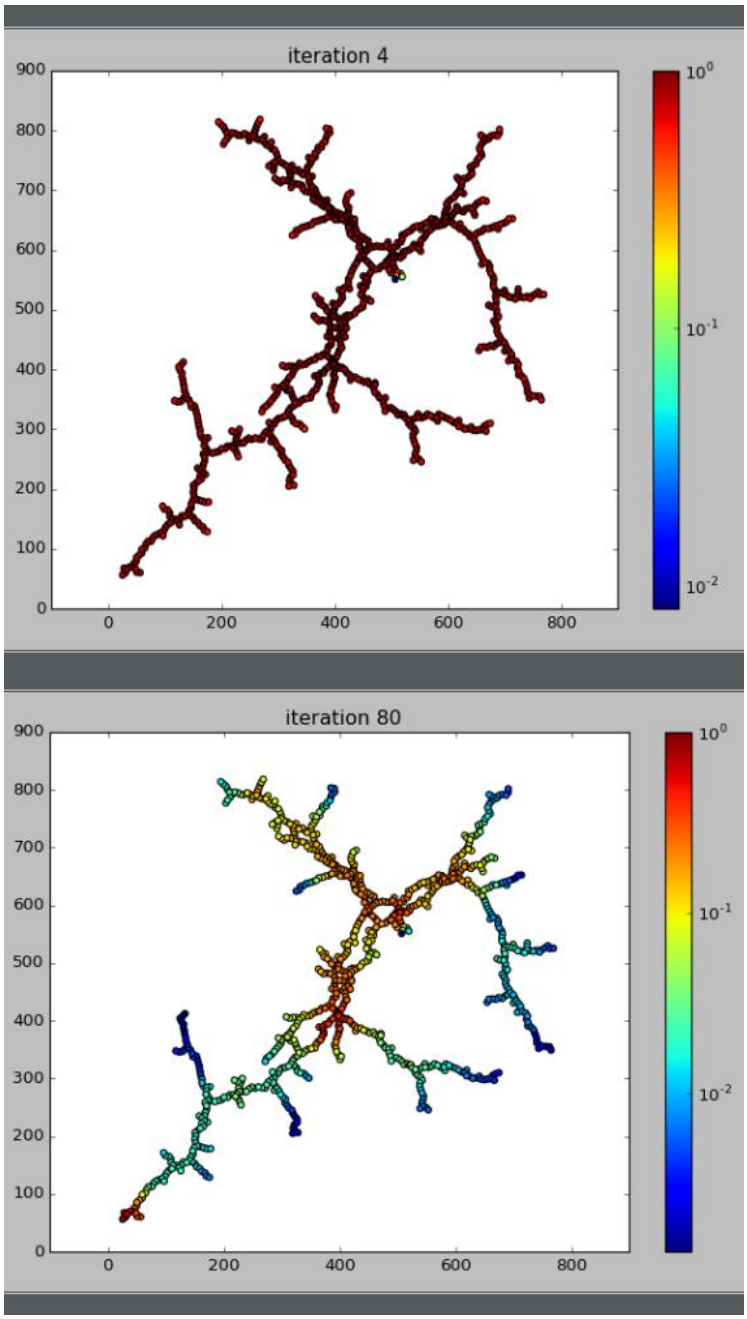

Figure 4-7. At iteration 4(top), iteration 80(bottom)

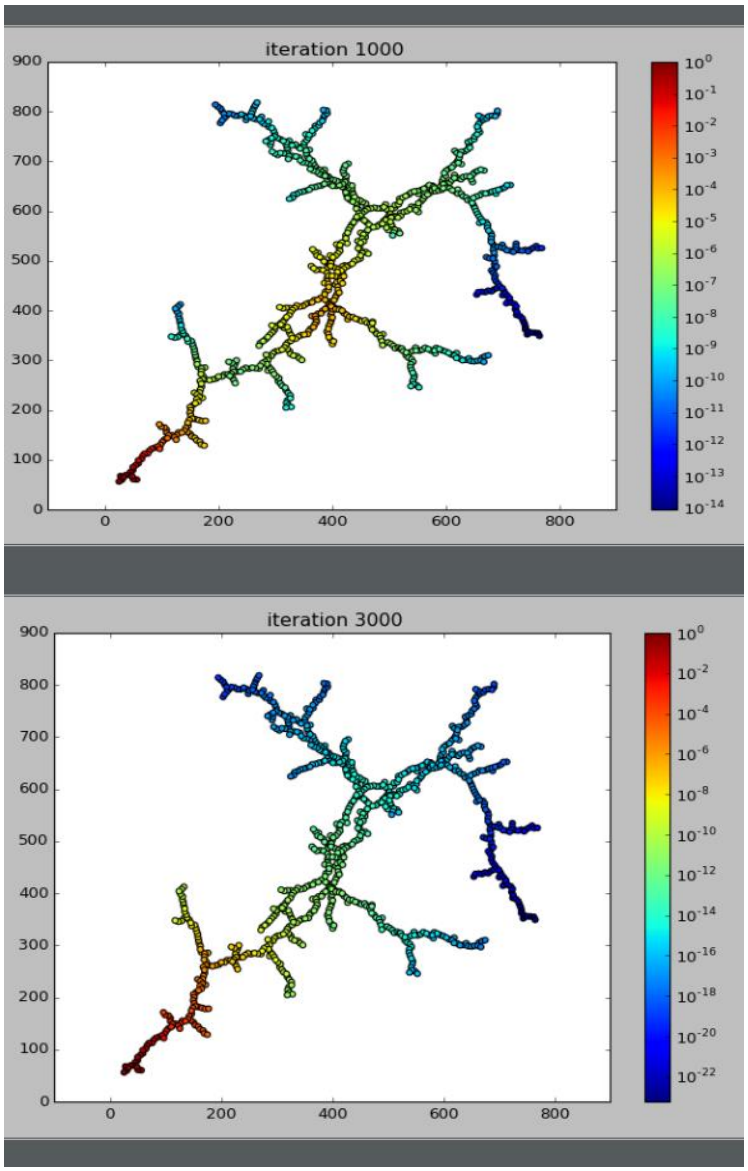

Figure 4-8. At iteration 1000(top), iteration 3000(bottom)

The path generated by the randomized algorithm and the potential value gradient along the path are shown in Figure 4-9 and the alternate path that the robot can follow to reach the goal when following the potential by moving to the directional nearest neighbor sample that has the lowest potential value is as shown in Figure 4-10.

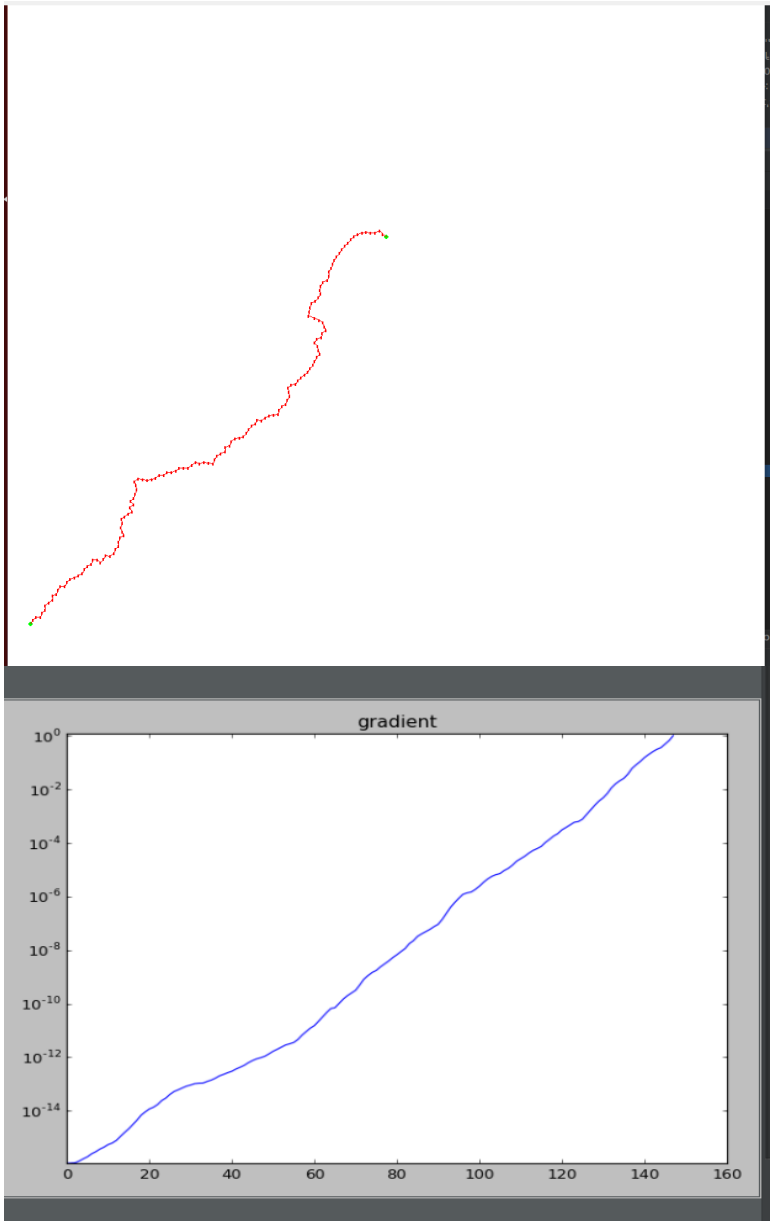

Figure 4-9. Random path generated by RRT (Top, Red), log scale of the negative gradient of the potential along the RRT path (Bottom blue)

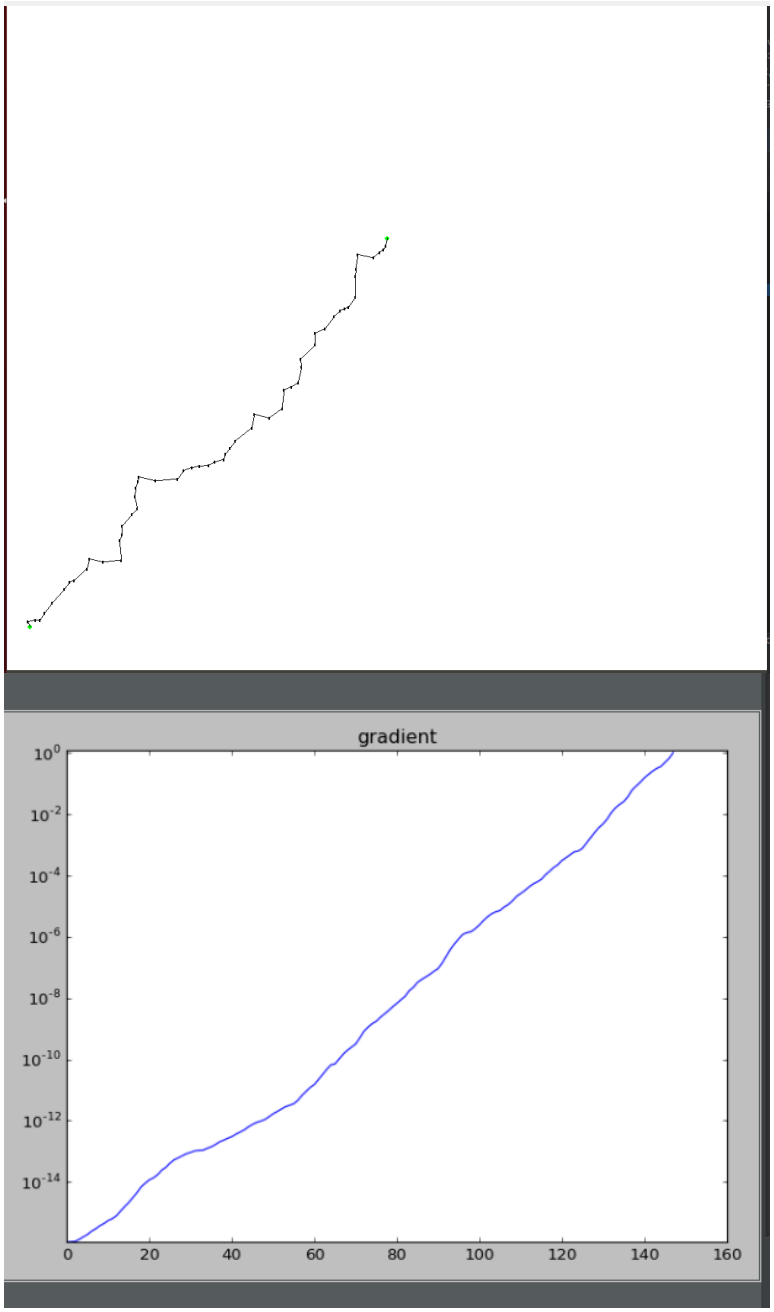

Figure 4-10. Potential value path generated by visiting the minimum value neighbor at each node (Top, Black), log of the negative gradient along the path (Bottom, blue)

Figure 4-11 and Figure 4-12 display the alternate paths the robot can take to reach the goal. Besides the random RRT path and the path following the lowest potential directional nearest neighbor, this also shows the path generated when following the gradient of the potential function. This path, which does not have to go through any of the sampled points, is generated by taking steps along the negative gradient calculated based on the nearest neighbors of the current location of the robot, resulting in a smoother path.

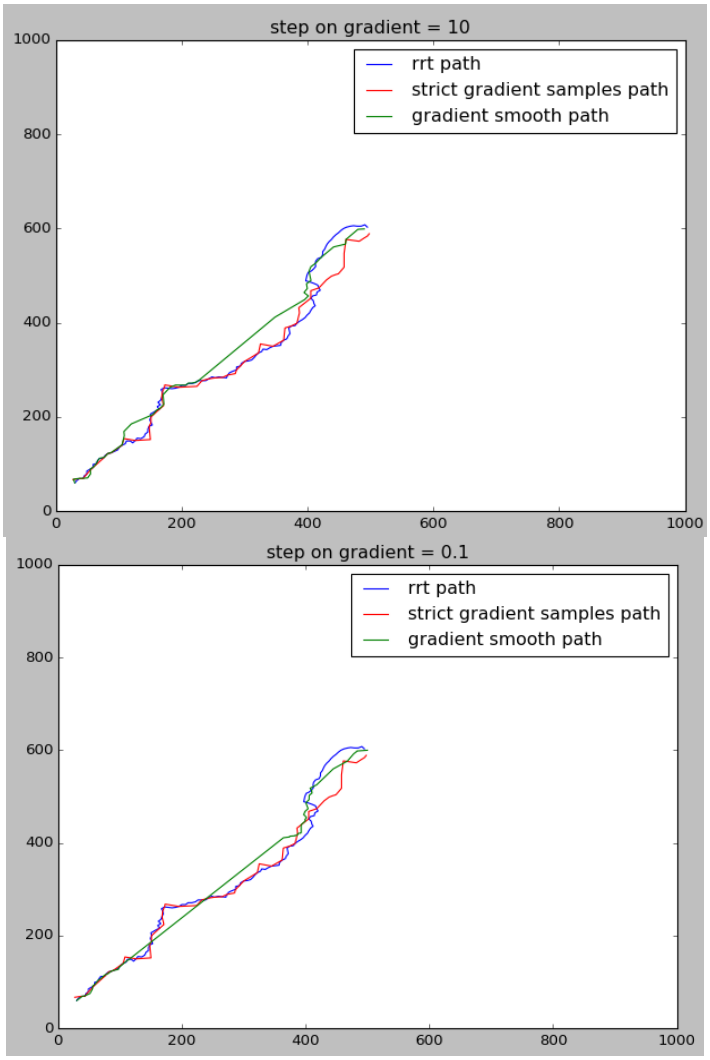

Figure 4-11. Smooth path along the gradient (Green), traversing samples based on gradient value (Red) and RRT(Blue) for step size 10(top) and step size 0.1(bottom)

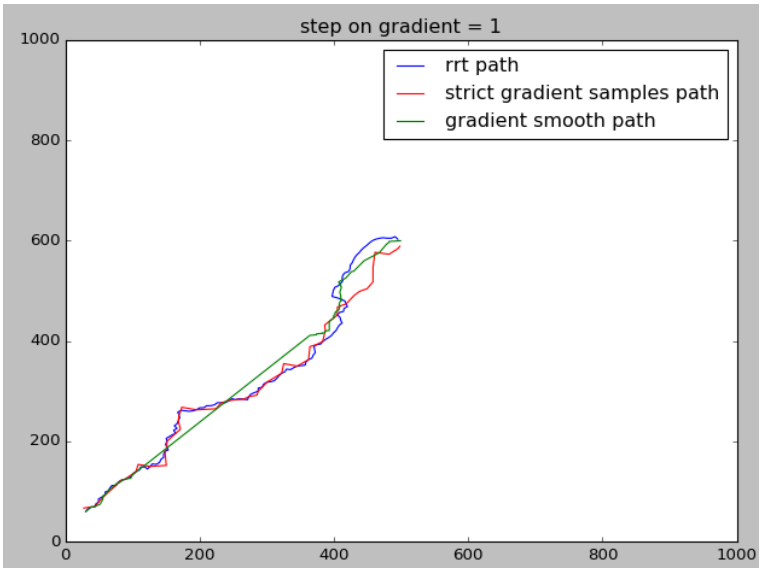

Figure 4-12. Smooth path along the gradient (Green), traversing samples based on gradient value (Red) and RRT(Blue) for step size 1

To illustrate the benefits of the potential function superimposed on the sample set, the following example shows a case where RRT sampling resulted in a larger number of branches in close proximity to each other. As a result, samples on different branches form neighbors and thus connect regions of freespace, yielding larger numbers of paths. This experiment shows the effect of the harmonic function potential which prefers paths with lower collision probabilities when utilizing the potential field and moving down the gradient. The RRT here finds a first path that moves close to an obstacle and thus includes higher collision risk. Figure 4-13 shows the extended sample set generated from the RRT run. Here the path found by the RRT planner passes close to obstacle. Figure 4-14 shows that the path generated by moving along the directional nearest neighbors with the smallest potential value follows a similar but alternate path to the one initially generated by the RRT.

However, if the path is computed based on the gradient of the potential function calculated based on the nearest neighbors at each step (step size here is 1), the resulting path takes a different route which stays further away from the obstacle but is slightly longer, reflecting the tradeoff made by the harmonic potential. Note, that in this example, both routes strictly move downhill on the potential function and would be a valid path from the perspective of the potential function

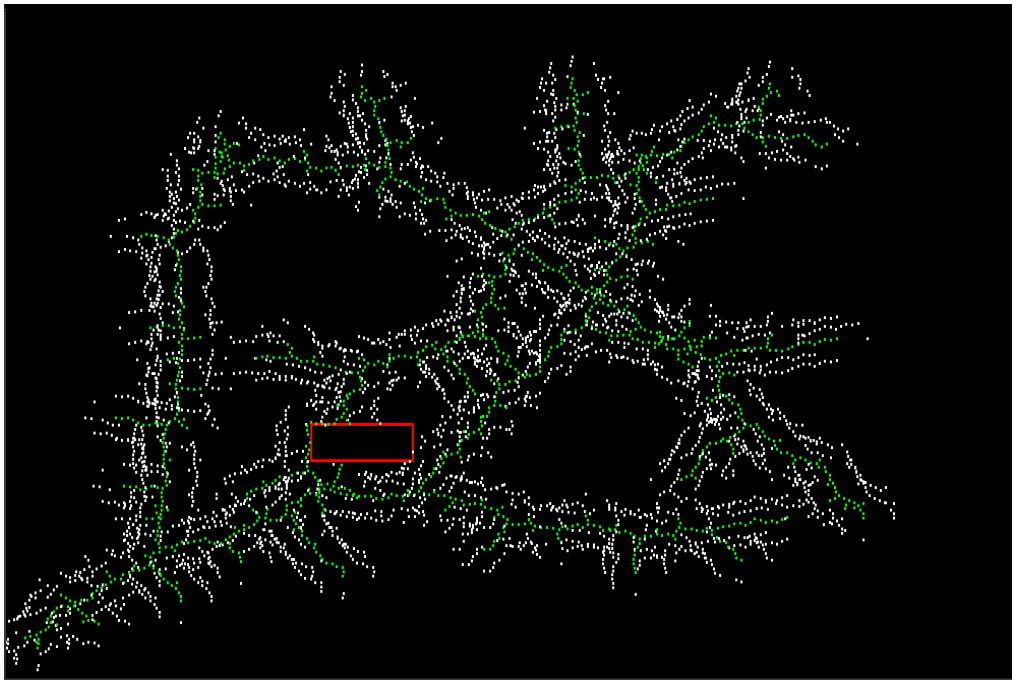

Figure 4-13. Smooth path along the gradient (Green), traversing samples based on gradient value (Red) and RRT(Blue) for step size 1

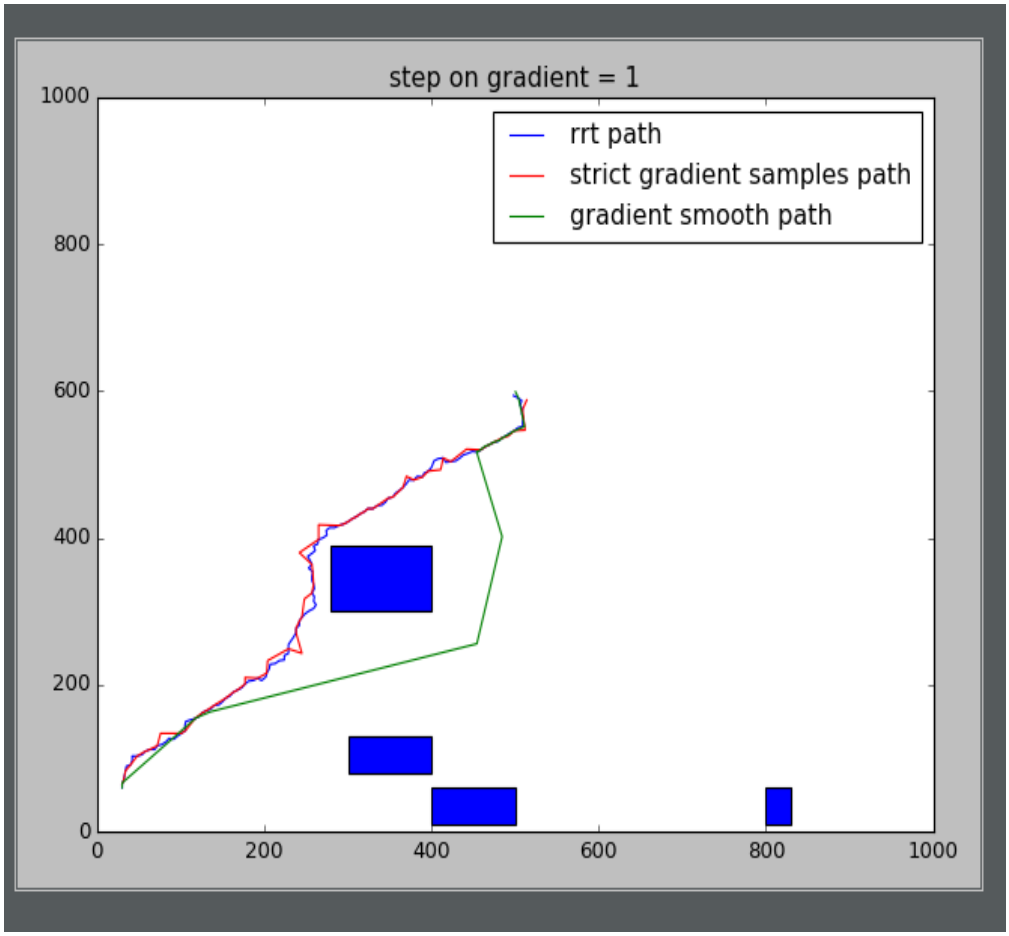

Figure 4-14. Imposed Potential Field

Chapter 5

CONCLUSION AND FUTURE WORK

Combining sampling-based global path planning with the benefits of navigation function based potential fields is one way to reduce re-planning when a robot deviates from the path. In addition, by using these navigation functions, secondary runtime constraints can be incorporated on the fly using a control composition scheme akin to Nullspace control. In the approach proposed in this thesis, RRT is exploited to address the scalability issue of harmonic functions by generating a targeted sample set based on the star and goal location on which the harmonic potential is computed using nearest neighbor relations. The use of RRT for sample generation here tends to generate fewer points even in larger spaces compared with structured sampling techniques (such as grid-based techniques) while being fast and efficient. The advantage of computing a potential over the resulting sample space in this method is the inherent availability of multiple paths to achieve the goal within the potential function which addresses deviations from the path and makes it easier to incorporate secondary conditions at runtime. The proposed technique is effective in handling the deviation of the robot from the original path and exploits the swiftness with which the RRT finds a path to generate a sample set and potential for subsequent smooth paths. Different versions of RRTs can be in cooperated in the proposed method.

As autonomous robots gain popularity. There needs to be continuous efforts towards increasing their efficiency. In striving to achieve more flexible path gerneation, this research has made a small contribution towards path plannig. Future work will focus on optimizing this technique by reducing the number of freespace samples along the originally generated path and adding additional sampling techniques to add samples that can bridge regions between different branches of the RRT tree, thus yielding a larger area that is

traversable using the harmonic potential. In addition to the sample generation, traversal in the KD-tree could be further improved to yield faster nearest neighbor calculations. In addition to that, extending the work to non-holonomic robots and higher dimensions could be a focus in future.

References

[1]. Steven M. LaValle 2006, Cambridge University Press

[2]. Lozano-Pérez, T. 1987. A Simple Motion-Planning Algorithm for General Robot Manipulators\ IEEE Transactions on Robotics and Automation, 3(3), 224-238.

[3]. Global Path Planning for Autonomous Mobile Robot using Genetic Algorithm, 2013 International Conference on Signal-Image Technology & Internet-Based Systems

[4]. M. Othman, M. Samadi, and M. Asl, “Simulation of dynamic path planning for real-time vision-base robots,” in Intelligent Robotics Systems: Inspiring the NEXT, ser. Communications in Computer and Information Science, K. Omar, M. Nordin, P. Vadakkepat, A. Prabuwono, S. Abdullah, J. Baltes, S. Amin, W. Hassan, and M. Nasrudin, Eds. Springer Berlin Heidelberg, 2013, vol. 376, pp. 1–10

[5]. LaValle, Steven M. Rapidly-exploring random trees: A new tool for planning, 1998

[6]. LaValle, Steven M, Juffner Jr. James J Randomized Kinodynamic Planning, The International Journal of Robotics Research (IJRR). 20 (5): 378–400, 2001

[7]. Ranganathan, Ananth; Koenig, Sven. PDRRTs: "Integrating Graph-Based and Cell-Based Planning". In Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS), pages 2799–2808, 2004.

[8]. Karaman, Sertac; Frazzoli, Emilio (3 May 2010). "Incremental Sampling-based Algorithms for Optimal Motion Planning".

[9]. Karaman, Sertac; Frazzoli, Emilio (5 May 2011). "Sampling-based Algorithms for Optimal Motion Planning

[10]. Islam, Fahad; Nasir, Jauwairia; Malik, Usman; Ayaz, Yasar; Hasan, Osman; "RRT*-Smart: Rapid convergence implementation of RRT* towards optimal solution", in Proceedings of IEEE International Conference on Mechatronics and Automation (ICMA), pages 1651–1656, Chengdu, China, August 2012.

[11]. Brunner, M.; Bruggemann, B.; Schulz, D. "Hierarchical rough terrain motion planning using an optimal sampling-based method," in Int. Conf. on Robotics and Automation (ICRA), Karlsruhe, Germany, 2013.

[12]. Adiyatov, Olzhas; Varol, Huseyin Atakan. "Rapidly-exploring random tree-based memory efficient motion planning". In Mechatronics and Automation (ICMA), 2013 IEEE International Conference on, pages 354–359, 2013.

[13]. Gammell, Jonathan D., Siddhartha S. Srinivasa, and Timothy D. Barfoot. "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic." arXiv preprint arXiv:1404.2334 (2014).

[14]. L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Transactions on Robotics and Automation, 12(4):566–580, 1996.

[15]. O. khatib, Real time obstacle avoidance for manipulators and mobile robots. International Journal of Robotics Research, 5(1): 90-98, Spring 1986.

[16]. W.S.Newman and N.Hogan. High speed robot control and obstacle avoidance using dynamic potential function. Precceedings of 1987 IEEE International Conference on Robotics and Automation, Rayleigh, 1987.

[17]. Park M. G., LEE M. C. (2003). A new technique to escape local minimum in artificial potential field based path planning. KSME International Journal. Vol 17, n 12, pp 1876–1885.

[18]. Ge S. S., Cui Y. J. (2000). New Potential Functions for Mobile Robot Path Planning. IEEE Transactions on Robotics and Automation. Vol 16, no 5, pp 615–620.

[19]. Walker, Jearl (2011). Principles of Physics (9th ed.). Hoboken, N.J. : Wiley. ISBN 0-470-56158-0.

[20]. Connolly, Christopher I., and Roderic A. Grupen. "The applications of harmonic functions to robotics." Journal of Robotic Systems 10.7 (1993): 931-946, 1993

[21]. Potential Field base Sampling Heuristic for Optimal Path Planning, arXiv:1704.00264v1 [cs.RO], 2017

[22]. Improving the Efficiency of Rapidly-exploring Random Trees Using a Potential Function Planner. 44th IEEE Conference on Decision and Control and the European Control Conference 2005.

[23]. Stentz, Anthony. "The focussed D^* algorithm for real-time replanning." IJCAI. Vol. 95. 1995.

[24]. Likhachev, Maxim, et al. "Anytime Dynamic A*: An Anytime, Replanning Algorithm." ICAPS. 2005.

[25]. Ernest Julius Wilczynski; Herbert Ellsworth Slaught (1914). "Theorem 1 and Theorem 2". Plane trigonometry and applications. Allyn and Bacon. p. 85.

[26]. Law, Henry (1853). "Corollary 5 of Proposition XLVII (Pythagoras's Theorem)". The Elements of Euclid: with many additional propositions, and

explanatory notes, to which is prefixed an introductory essay on logic. John Weale. p. 49.

[27]. Hart, P. E.; Nilsson, N. J.; Raphael, B. (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". IEEE Transactions on Systems Science and Cybernetics SSC4. 4 (2): 100–107.

[28]. Pohl, Ira (1971), "Bi-directional Search", in Meltzer, Bernard; Michie, Donald, Machine Intelligence, 6, Edinburgh University Press, pp. 127–140.

[29]. Maneewongvatana and Mount 1999

BIOGRAPHICAL STATEMENT

Sandeep Chahal started his master's in Computer Science at University of Texas at Arlington in 2016 specializing in Intelligent Systems. His research interests are Machine Learning and Robotics. He is continuing with his doctoral program after graduation.