AN OVERVIEW AND IDEAS ON AUTONOMOUS ROBOT PATH PLANNING ALGORITHMS

D. Fanucchi*, J. Claassens†, M.K. Banda‡, and Simukai Utete§

Abstract

Robot Path Planning refers to the task of obtaining a continuous path through the free configuration space, $\mathcal{C}_{free}$, of some robot from an initial to a goal configuration. This Report reviews the progress in this field, with particular focus on the Rapidly Exploring Random Tree (RRT) and the Artificial Potential Fields approaches. Some theoretical results regarding RRTs are proved, motivating a modification in the RRT approach to include local biasing and a new scheme for vertex selection.

1 Introduction

In recent years *Path Planning* has grown into an enormous field, incorporating *Control Theory*, *Robotics*, *Artificial Intelligence* and to an extent *Algorithm Analysis and Design* [1]. In all its forms, *Path Planning* essentially involves defining a sequence of configurations[1] of some system beginning at an initial

*School of Computational and Applied Mathematics, University of the Witwatersrand, Private Bag 3, Wits 2050, *DarioF2@gmail.com*

†CSIR Mobile Intelligent Autonomous Systems, P.O. Box 395, Pretoria 0001, *jclaassens@csir.co.za*

‡School of Computational and Applied Mathematics, University of the Witwatersrand, Private Bag 3, Wits 2050, *Mapundi.Banda@wits.ac.za*

§CSIR Mobile Intelligent Autonomous Systems, P.O. Box 395, Pretoria 0001, *sutete@csir.co.za*

[1]A sequence of configurations may also be referred to as a "trajectory".

state and ending at a target state whilst satisfying certain constraints. In the context of robotics, an object with several degrees of freedom is given and the idea is to find a sequence of actions that will enable the object to move from one place to another in a complex environment. Problems requiring planning in this sense are vast, ranging from deciding on the best route through a maze to solving the famous *Alpha 1.0* entanglement problem[2] [1].

The first challenge in any of these problems is finding a feasible path. Often this is very difficult due to the intrinsic dimensionality of the problem. Another consideration, often addressed in *Control Theory*, is optimality.

This Report reviews two of the currently most successful methods for finding feasible paths in high dimensional spaces, Artificial Potential Fields and Rapidly Exploring Random Trees. It also introduces a few variants of the algorithms based on the two above and considers some simple computational path optimization techniques.

Section 2 introduces the main ideas in *robot path planning*; in particular Section 2.4 constitutes a background literature review in which various existing path planning algorithms are introduced. In Section 3 the two methods that form the focus of this paper are reviewed. The Artificial Potential Fields approach is discussed in Section 3.1, and several variants are described. Section 3.2 outlines Rapidly Exploring Random Trees (RRT) and sketches a few of their basic properties. In Section 4 the limiting case of 'RRT in a Large Disc', which was proposed by LaValle and Kuffner in [2], is considered in a theoretical setting, and some results are derived. Sections 5.1 to 5.2 describe some simple variants on the RRT and the Artificial Potential Fields methods, mostly based on the theoretical analysis in Section 4. Section 5.3 outlines a simple optimization procedure once feasible paths have been obtained and Section 6 provides test results for some of the algorithms presented.

2 Preliminaries

2.1 The Space of a Robot

There are two spaces usually associated with a robot - the *workspace* and the *configuration space*.

[2]This problem, which involves taking apart two wires twisted in a particular fashion, is seen as a benchmark test for planning algorithms.

The *workspace*, $\mathcal{W}$, also referred to as the *world* in [1], is the actual physical space in which the robot physically exists. If a point on the robot is fixed, the possible positions of that point (assuming all obstacles are removed) describe positions in the workspace [3]. For instance, a flat robot moving in a plane has a two-dimensional (2D) workspace that can be best parametrized with Cartesian coordinates. Likewise a flat robot moving on the surface of a sphere also has a 2D workspace, best characterized with spherical polar coordinates. A spacecraft moving in an asteroid field has a three-dimensional (3D) workspace.

Not all of the workspace can be traversed, due to the presence of obstacles. Each obstacle is also considered as a collection of points in the workspace. Let $\mathcal{W}_{obs}$ be the union of all the obstacles in the workspace, and let $\mathcal{W}_{free} = \mathcal{W} \backslash \mathcal{W}_{obs}$, which can be referred to as the free part of the workspace.

The workspace does not capture all the information required to control the robot. To capture this information the robot's *configuration* (position and orientation of all components) as well must be considered. For instance the *configuration* of a robotic arm can be completely described by the joint angles. Any rigid 3D object in 3D space has a six-dimensional (6D) configuration given by its 3D position and its orientation in terms of pitch, yaw and roll. The *Configuration Space*, $\mathcal{C}$, of the robot is defined to be the space of all configurations of the robot. In most planning problems the configuration space turns out to be a manifold. In the case of the rigid 2D body in 2D space, for instance, the configuration space is given by position ($\mathbb{R}^2$) and orientation (each angle can be associated with a point on the unit circle, $\mathcal{S}^1$), and $\mathcal{C} = \mathbb{R}^2 \times \mathcal{S}^1$, which is a smooth 3D manifold. Finally, $\mathcal{C}_{free} \subseteq \mathcal{C}$ is defined to be the set of those configurations for which the robot is entirely contained within $\mathcal{W}_{free}$.

In applications attention is restricted to some bounded space (it obviously would take infinite time to traverse an unbounded space). Thus $\mathcal{W}$ and $\mathcal{C}$ are bounded, and thus $\mathcal{C}$ is a bounded manifold.

If the robot is constrained so that it is rigid and can only translate (without rotating) in space, then the position of any (consistently chosen) fixed point on the robot fully determines the entire configuration of the robot. Hence in this case (disregarding obstacles) $\mathcal{W} = \mathcal{C}$. By appropriately re-scaling the obstacles[3] the problem can be reduced to planning the path of a single point through the new map. Thus the case $\mathcal{W} = \mathcal{C}$ is referred to as *Point Planning*.

In general, however, $\mathcal{C}$ is far more complicated than $\mathcal{W}$. A robot arm, for

[3]The mathematical morphology operation of *dilation* to the obstacles with the robot as a structuring element is applied.

instance, has several joint angles and a robot manipulating complex objects (like ropes) in the environment must deal with a very high dimensional configuration space. Thus in general the planning problem is not a *simple* problem to solve, and the solution is seldom visible at the outset. Planning algorithms, therefore, address both situations that humans find trivial and situations that humans find challenging.

2.2 Kinematics

Let $\mathcal{A}$ be a robot with a configuration space $\mathcal{C}$ and a workspace $\mathcal{W}$. Clearly each configuration in $\mathcal{C}$ maps $\mathcal{A}$ to some subset of the workspace. Thus if a point $x \in \mathcal{A}$ is fixed, any configuration in $\mathcal{C}$ sends this point to some point in $\mathcal{W}$, namely the position of x when the robot is in the given configuration. Hence the following map is described:

$$X : \mathcal{A} \times \mathcal{C} \to \mathcal{W},$$

which is referred to in the literature [3] as the *forward kinematic map*.

The *forward kinematic map* fully describes the relationship between the configuration space and the workspace.

2.3 Path Planning: Formally

Given an initial configuration $q_{init} \in \mathcal{C}_{free}$ and a set of goal configurations $Q_{goal} \subset \mathcal{C}_{free}$, path planning involves determining a continuous path $f : [0, 1] \to \mathcal{C}_{free}$ subject to certain constraints.

Often the robot must move according to some dynamics, and its motion is a function of the controls applied. This can be written as

$$\dot{q} = f(q, u)$$

where $q \in \mathcal{C}_{free}$ and u is the control chosen to act on the robot. This is often referred to as the *state transition equation* [1].

In the simplest case, the sequence of positions of the robot are essential. This is equivalent to dynamic cases where the robot can be controlled completely and is referred to as *Holonomic Control*. In this instance, the set of controls is defined as $\mathcal{U} = \{u : ||u|| \leq c\}$ with c some constant and f is simply given by $\dot{q} = u$. Hence all velocities up to a maximum bound are attainable at

any point. This ensures that all physical paths are possible. Often for Holonomic control the velocities are not important. In this case the state transition equation are not even used, but instead plan a path in space directly. This paper will be primarily concerned with Holonomic Planning.

Nonholonomic planning refers to instances where the available controls are not enough to fully characterize the motion of the robot. For instance driving a car is a Nonholonomic model because while the car has 3 degrees of freedom (it can rotate and translate in the plane), 2 degrees of freedom (specifying the angle of the wheels and the forward thrust) are controllable when driving the car.

2.4 Overview of Methods

In this Report two methods, the Potential Fields and Rapidly-Exploring Random Tree approaches, will be discussed in detail in Section 3. Here the context in which these methods appear is presented. Further to that, the most common methods in the literature will also be discussed.

We first consider methods specialized towards 2D point planning: planning a path for a point particle in a 2D workspace. As mentioned earlier, any path planning problem with a non-rotating rigid body in 2D can be reduced to a point planning problem. Finding a feasible path in a 2D region with obstacles is a very well studied problem, and several methods have been developed. The simplest methods, which are very easy to implement in real-time for robots navigating a floor with obstacles, are the so-called *bug algorithms* [1]. All these essentially work on the principle of starting at x_{init} and moving in a straight line towards x_{goal}. Whenever an obstacle is encountered the robot chooses a direction and moves around the obstacle (keeping it at a constant distance) until it returns to the original line on which it was traveling, on the other side of the object. This continues until the goal is reached. It can be proven that, with polygonal obstacles, this method will always reach the goal.

An advantage of the bug algorithms is that they are simple to implement in a real robot with no maps of the environment except the location of the target. They are also guaranteed to produce a path if one exists in a 2D point planning situation.

The main disadvantage of the bug algorithms is that they cannot readily be extended to higher dimensions or to more complex configuration spaces. This limits their applicability to the rather trivial case of 2D point planning. Another (less serious) disadvantage of the bug algorithms is that the paths

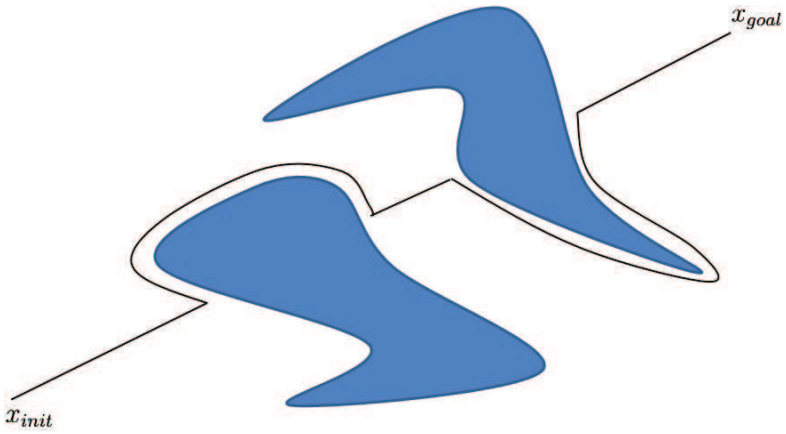

Figure 1: An example of the *Bug2* variant of the bug algorithm.

they return are not optimal (although there are improvements that make the paths shorter).

If the configuration space is of relatively low dimension (≤ 3 in most cases [4]) then *Dynamic Programming* is a feasible method for path planning. A suitable variation of Dynamic Programming for the path planning problem is presented in [4]. Given a set of local controls $\mathcal{U}$, the idea is to obtain a discretization of the space and then to iterate through this grid repeatedly updating the optimal *cost-to-go*, $l(x_k, u_k)$, and best *next control* at each point, $L_k^*(x_{k+1})$. This is done in the classical *Dynamic Programming* sense [4]:

$$L_k^*(x_k) = min_{u_k}\{l(x_k, u_k) + L_k^*(x_{k+1})\}.$$

Dynamic Programming has the advantage of computing *optimal paths* in quite a general sense (optimizing a functional). The disadvantage is that it becomes very expensive as the dimension of the problem increases.

Relatively new approaches to the path-planning problem are the so-called *topological methods*. These methods are derived from analytical results on various classes of configuration spaces that give information about the complexity of the path planning problem on these spaces and algorithms for finding smooth paths. The construction of smooth paths in an arbitrary configuration space is a nontrivial task. Indeed, simple methods available in Euclidean space are not available in, for instance, the *rotation group*. In [5] *Topological Planning methods* for building such smooth paths via interpolation are introduced.

Further [6] considers how very high dimensions can affect normally simple path finding problems, and assigns a value - the *topological complexity* - to the real projective spaces, which determines the difficulty of the planning problem in these spaces.

Most of this work cannot (currently) deal directly with obstacles and discontinuities and it is therefore mostly insufficient to build planners entirely based on these ideas at present, except in cases where $\mathcal{C}_{free}$ is a well-behaved and well-studied topological space. On the other hand these methods provide a good way of dealing with dynamics. In some instances it may be viable to combine topological methods with more general planners such as RRT or Probabilistic Road Maps discussed below. They may be particularly useful for the local connect steps that are sometimes required in these algorithms.

The *Probabilistic Road Maps* (PRM) method for path planning is a highly popular and very general method for finding paths in $\mathcal{C}$. The basic strategy is to generate many random configurations in $\mathcal{C}_{free}$ and try to connect nearby configurations (using some local planner). The distance between configurations is determined by some suitable metric. (These are not, in general, easy to find. See Section 3.2 for a discussion in the context of RRTs). The configurations and the connections between them form a graph in $\mathcal{C}$, which can be searched using some graph-searching algorithm[4].

The main objection to PRM is that it is not always easy to connect t-wo random nearby states. Indeed in some instances (such as those involving high dimensional kinodynamic planning) this can be almost as difficult as the original planning problem [4].

In [7] the use of *Adaptive Random Walks* for single-shot path planning was proposed. The idea is to start at a point and repeatedly generate a neighbour based on an increment given with zero mean and a covariance that is adapted as the walk proceeds. The conditions for convergence are provided in [7]. An interesting property of this method is that it adapts to the topology of the environment - if the walk is moving through a narrow corridor, the covariance matrix will eventually narrow out to choose points in the corridor with more frequency than those outside. This makes it more efficient than a blind random walk. An advantage of this method over more structured methods like PRM or RRT is that it only requires constant time computation on each step (as opposed to linear or logarithmic time). This makes it feasible even though it needs to generate many more points than RRT, for instance, to find a path.

[4]The common choice here being the A^* algorithm.

It would be interesting to combine this method with some of the major planners in some way. In particular, it is considered as a local bias routine for RRT in Section 5.1

3 The Potential Fields and RRT Approaches

This section introduces the *Potential Fields* and *RRT* approaches to path planning. Each is introduced and reviewed in some detail, and some broad suggestions are made towards possible extensions.

3.1 The Potential Field Approach

Formally, the potential field method involves coming up with some artificial potential function V on the configuration space $\mathcal{C}$ such that V has a minimum at the goal configuration (or on the set of goal configurations)[5]. This potential function generates an artificial force, $F = -\nabla V$, that acts on the configuration of the robot at any point. The robot is moved as though it was *really* being acted upon by this force. Because the goal is the minimum of the artificial potential, the resulting force causes the robot to be attracted to the goal (like a marble rolling down hill towards the bottom). Ideally, the goal would be the only minimum of V, and this is easy to achieve in cases where there are no obstacles. For instance, one might set V to be a paraboloid, a cone or indeed a hyperboloid function (as considered more recently in [8]) centered at the goal. In these cases one has a lot of control over what kinds of paths are developed, and V can be tweaked in order to come up with different shapes and classes of paths.

With the introduction of obstacles several issues arise. In order to prevent the robot from traveling through the obstacles, the obstacles must be assigned repulsive forces in the potential function. This is usually accomplished by setting V to be large on the obstacles. The problem is that this usually adds local minima to the potential function, since it must have continuous derivatives. These local minima are *traps* for the robot. Several solutions have been proposed for this 'local minima' problem, and two of them will be considered shortly.

Note that mostly the use of numerical methods will be necessary in this method because ∇V needs to be computed to move the robot. Sometimes, for

[5]Usually V is zero at the goal.

specialist applications, it is possible to come up with an analytic expression for V beforehand and hence avoid numerical differentiation and the associated discretization of the space, but in most applications this is unreasonable (indeed V often depends on the layout of the space, including the obstacles and hence cannot be computed beforehand).

Since this method often demands the discretization of the space, one is concerened about the large number of dimensions in the configuration space. However, the fact that the robot is actually embedded in a lower dimensional workspace is advantageous. An approach that first appeared in [3] is to discretize the workspace and use this discretization in the computation of a $\mathcal{C}$-space potential function. The $\mathcal{C}$-space does not then need explicit discretization. This method is described below.

3.1.1 Numerical Potential Fields

The algorithm considered in [3] consists of four steps:

1. Discretize the Workspace, $\mathcal{W}$.
2. Compute $\mathcal{W}$ Potentials.
3. Decide step sizes in $\mathcal{C}$-space.
4. Traverse the $\mathcal{C}$-space.

The first point involves the selection of a discretization parameter or step size, δ. In [3] several values of δ are chosen, creating a pyramid of discretized bitmaps of the $\mathcal{W}$-space.

For a given discretization, the $\mathcal{W}$ potentials are computed. This can be done in many ways. The first method considered in [3] is to compute the $\mathcal{W}$-potential of a point by executing a breadth first search (BFS) through $\mathcal{W}_{free}$, started at the goal. The potential of a node is set when it is visited to be one more than the potential of its parent in the BFS tree. This method was implemented in MATLAB for a simple environment, and the resulting potential function can be visualized in Figure 2.

In [3] another algorithm that keeps objects a good distance away from obstacles is also considered. This method first builds the potential function on the *skeleton* (as in Mathematical Morphology) of the $\mathcal{W}$-space. The potential function is then computed in the remainder of the space.

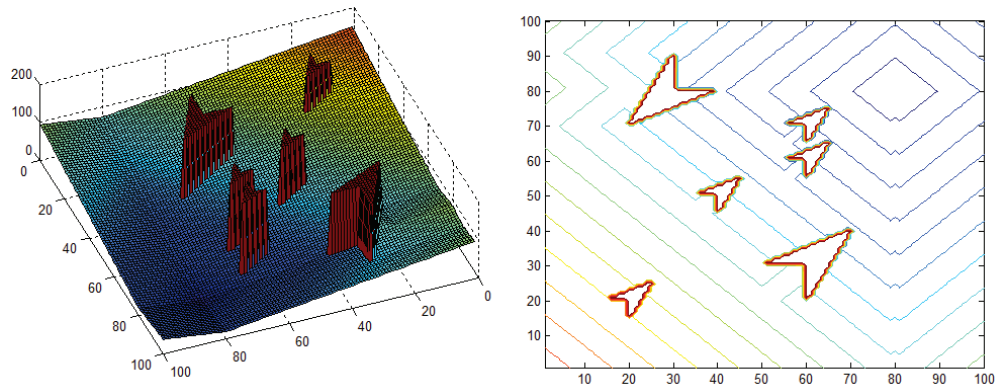

Figure 2: An illustration of the BFS construction of a discrete potential field in a simple space. The field is illustrated (left) and the original map is endowed with field contours (right).

The main contribution of [3] was the computation of a $\mathcal{C}$-potential when given a $\mathcal{W}$-potential. The advantages of this approach are clear - the $\mathcal{C}$-space does not need to be explicitly discretized at any time. Instead the value of the $\mathcal{C}$-potential at a point is computed by using the values of the $\mathcal{W}$-potentials. This is done by making use of the forward kinematic map, f. Some points on the robot are preset. These are generally chosen as extreme points or other points that will capture the shape of the robot. These points are called *control points*, $\{c_1, c_2, \ldots, c_k\} \subset \mathcal{A}$. Now in the configuration q_{init}, the initial positions of all the control points are obtained as $\{f(c_1, q_{init}), f(c_2, q_{init}), \ldots, f(c_k, q_{init})\}$. The positions of the control points at q_{goal} can be found analogously.

Thus one has a system of planning problems in the $\mathcal{W}$-space. For each control point there is an initial position and a goal position. In particular, the goal position of each control point can be used to build a potential on the $\mathcal{W}$-space. For control point c_i, V_i denotes its associated $\mathcal{W}$-potential.

A potential function $\bar{V}$ on $\mathcal{C}$ is then given by

$$\bar{V}(q) = \mathcal{G}(V_1(f(c_1, q)), V_2(f(c_2, q)), ..., V_k(f(c_k, q)))$$

where $\mathcal{G}$ is referred to as the *arbitration function*. It turns out that the choice of $\mathcal{G}$ is crucial because mostly it introduces spurious minima.

Using V as defined above it is not necessary to discretize the $\mathcal{C}$-space explicitly. However, as the planner will need to compute ∇V numerically, a mesh

granularity in $\mathcal{C}$-space needs to be specified. This is the 'increment distance', Δ, used for computing ∇V. In [3], this value is computed for each corresponding value of δ, creating a $\mathcal{C}$-space pyramid. The problem is then attempted first at the lowest granularity and the granularity is increased until a solution is found.

When choosing the granularity of the $\mathcal{C} - space$ two considerations are important: the mesh must not be too fine relative to the $\mathcal{W}$ mesh since then the computation of $\bar{V}$ will be erroneous. On the other hand, the mesh must not be too coarse relative to the $\mathcal{W}$ mesh since that would mean unnecessary work has been expended on the $\mathcal{W}$ mesh and not all of the information will be captured in the discretization on $\mathcal{C}$. In [3] the criterion for selecting the $\mathcal{C}$-space step size, Δ, given the $\mathcal{W}$-space step size, δ, is to impose the condition that no point in $\mathcal{A}$ moves by more than $\epsilon\delta$ when the configuration changes by Δ. Here ϵ is a small constant (in [3] $\epsilon = 2$ is suggested).

This can be enforced as follows: If we take $\bar{q}_i - q_i = \Delta_i$ then, assuming that x are the variables in workspace, by Taylor expansion one obtains

$$\begin{aligned}\bar{x}_j &= x_j(p, q_1, q_2, \ldots, q_i + \Delta_i, \ldots, q_n);\\ &= x_j(p, q_1, q_2, \ldots, q_i, \ldots, q_n) + \Delta_i \frac{\partial x_j}{\partial q_i} + O(\Delta_i^2);\\ &\approx x_j(p, q) + \Delta_i \frac{\partial x_j}{\partial q_i};\end{aligned}$$

where $q = (q_1, q_2, \ldots, q_n) \in \mathcal{C}$. In this case the convention of Einstein's summation is applied.

Hence

$$\Delta_i \frac{\partial x_j}{\partial q_i} + O(\Delta_i^2) = \bar{x}_j - x_j(p, q) \leq \epsilon\delta, \quad \forall q \in \mathcal{C}, \quad \forall p \in \mathcal{A}, \quad \forall j \in \{1, \ldots, d\}$$

where d is the dimension of $\mathcal{W}$.

Because this inequality must be satisfied over all $q \in \mathcal{C}, p \in \mathcal{A}, j \in \{1, \ldots, d\}$, the following holds:

$$\Delta_i \leq \inf_{\{q \in \mathcal{C},\, p \in \mathcal{A},\, j \in \{1,\ldots,d\}\}} \left(\frac{\epsilon\delta}{\frac{\partial x_j}{\partial q_i}} \right) = \frac{\epsilon\delta}{\sup_{\{q \in \mathcal{C},\, p \in \mathcal{A},\, j \in \{1,\ldots,d\}\}} \left(\frac{\partial x_j}{\partial q_i} \right)}.$$

It can be observed that the last step follows from the result

$$\inf\left(\frac{1}{f}\right) = \frac{1}{\sup f}$$

and the fact that $\epsilon\delta$ does not depend on any of the variables in question.

3.1.2 Probabilistic Escape

Regardless of the choice of the arbitration function, there will always be situations where the $\mathcal{C}$-space potential is riddled with local minima. The algorithm can check for a local minimum by testing the *numerical* gradient against a tolerance. The solution first proposed in [3] for the problem of local minima was that of *probabilistic escape*. This has turned out to be a powerful method, leading to the class of algorithms known as *Randomized Potential Fields* planners. Once it has been determined that the algorithm is in a local minimum, random escape directions are attempted (in the most general sense a random control is chosen and actuated for a random length of time). After each random attempt to escape, the algorithm again moves down the potential field. If it arrives back at the same minimum it tries to escape again but with a *larger* displacement than before until it has escaped. Alternatively it can make several random steps from the minimum (constituting a random walk). This method can still cause the system to become stuck in a local minimum, but given enough time, it will always escape the minimum eventually. There have been many proposals as to which random schemes to use for the random escape. In [9] a proposal was made to use RRTs for this task and the use of Adaptive Random Walks was proposed in [7]. To the best of the authors' knowledge, neither has yet been tried in this context. Several schemes have been developed in the optimization literature for escaping local minima, and many of them could be applied to the problem of escaping potential minima. It might also be of interest to apply clustering or simplex based methods to this situation.

3.1.3 Harmonic Functions

A major problem with the potential fields method is the existence of many local minima. To avoid local minima entirely one can impose conditions on V. In particular, an approach that has gained popularity is to eliminate minima altogether.

A harmonic function on an open region Ω is a function, $\phi(x_1, ..., x_n)$, satisfying *Laplace's Equation*:

$$\sum_{i=1}^{n} \frac{\partial^2 \phi}{\partial x_i^2} = 0.$$

It is a classical result that if Ω is multiply connected, then ϕ will attain its maximum and minimum values *exclusively* on the boundary, $\partial\Omega$ [10].

This eliminates local minima within the region Ω. Note that this means the goal point will have to be removed from Ω in order to make it a minimum of V. The usual procedure is to add a point potential well at the goal such that $V(x) \to -\infty$ as x approaches the goal.

An example of a harmonic function which might be used for the goal is [11]

$$g(x) = -\log ||x - x_{goal}||.$$

Similarly, one can define a repulsive potential by the negative of the above function. Since Laplace's equation is linear, any linear combination of the above potentials will also be harmonic. This means harmonic functions can be built by superposition of the equations of the above form, building an attraction at the goal and repulsion around the obstacles.

However, it is argued in [11] that there are always situations in which superposition style arguments lead to collisions with obstacles, because they only become infinite at individual points.

A method put forward in [11] to avoid these problems is simply to solve Laplace's equation numerically in the given region. From the Taylor expansion with step size h, $u_{i,j}$ is a discretization (in two dimensions) of ϕ as shown below:

$$h^2 \frac{\partial^2 \phi(x_i, y_j)}{\partial x^2} \approx u_{i-1,j} + u_{i,j-1} + u_{i+1,j} + u_{i,j+1} - 4u_{i,j}.$$

The resulting linear systems can be large (particularly in higher dimensional space). Thus an iterative solver for linear systems such as the Gauss-Seidel iterations can be applied [11]. This was implemented in MATLAB for some 2D environments. Figure 3 illustrates the potential field generated after a small number of iterations.

The nature of this discretization is that ultimately many flat regions form in space (due to numerical round-off errors). In [11] a way to alleviate this by bit-shifting whenever such flat regions are observed was specified.

The establishment of a grid in high dimensions can be an expensive operation. However, the philosophy in [3] can be applied in this instance to build

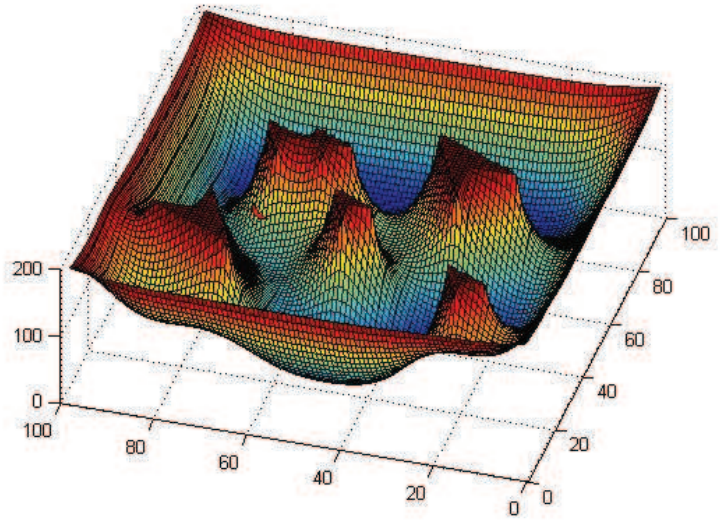

Figure 3: A simple discretized computation of an harmonic function.

the potential function in the $\mathcal{W}$-space. Control points $\{c_1, c_2, \ldots, c_k\} \subset \mathcal{A}$ are chosen together with goals $\{g_1, g_2, \ldots, g_k\} \subset \mathcal{W}$. The Harmonic Potentials for each control point are built, which would individually drive them to their respective goals.

An arbitration function that causes the resulting potential function on the $\mathcal{C}$-space to be harmonic needs to be chosen. In terms of the arbitration function, the new potential $\bar{V}$ can be considered as a change of variables:

$$\bar{V}(q) = \mathcal{G}(V_1(x_1^1(q), x_2^1(q), \ldots, x_m^1(q)), \ldots, V_k(x_1^k(q), x_2^k(q), \ldots, x_m^k(q))).$$

Thus

$$\frac{\partial \bar{V}}{\partial q_s} = \sum_{i,j} \frac{\partial \mathcal{G}}{\partial V_i} \frac{\partial V_i}{\partial x_j^i} \frac{\partial x_j^i}{\partial q_s}.$$

Hence

$$\frac{\partial^2 \bar{V}}{\partial q_s^2} = \sum_{i,j} \frac{\partial}{\partial q_s} \left(\frac{\partial \mathcal{G}}{\partial V_i} \frac{\partial V_i}{\partial x_j^i} \frac{\partial x_j^i}{\partial q_s} \right)$$
$$= \sum_{i,j} \left(\frac{\partial^2 \mathcal{G}}{\partial V_i^2} \frac{\partial V_i}{\partial x_j^i} \frac{\partial x_j^i}{\partial q_s} + \frac{\partial \mathcal{G}}{\partial V_i} \frac{\partial^2 V_i}{\partial (x_j^i)^2} \frac{\partial x_j^i}{\partial q_s} + \frac{\partial \mathcal{G}}{\partial V_i} \frac{\partial V_i}{\partial x_j^i} \frac{\partial^2 x_j^i}{\partial {q_s}^2} \right)$$

yielding

$$\nabla^2 \bar{V} = \sum_{i,j,s} \left(\frac{\partial^2 \mathcal{G}}{\partial V_i^2} \frac{\partial V_i}{\partial x_j^i} \frac{\partial x_j^i}{\partial q_s} + \frac{\partial \mathcal{G}}{\partial V_i} \frac{\partial^2 V_i}{\partial (x_j^i)^2} \frac{\partial x_j^i}{\partial q_s} + \frac{\partial \mathcal{G}}{\partial V_i} \frac{\partial V_i}{\partial x_j^i} \frac{\partial^2 x_j^i}{\partial {q_s}^2} \right).$$

This gives us a form for $\mathcal{G}$ that must be investigated.

As an experiment the Harmonic Function approach in the 2D workspace of a bar in a simple environment was implemented. The arbitration functions suggested in [3] (supremum minus infimum) to create a potential function on $\mathcal{C}_{free}$ were applied. Provided that enough iterations of the numerical scheme for Laplace's Equation were performed, the bar never gets stuck in local minima in any of the experiments.

3.1.4 Other variants

Many of the Potential Field methods in the literature are inspired by physical situations. Some make use of the superposition of harmonic functions from physics while others use Simulated Annealing type strategies to avoid getting trapped in local minima in arbitrary potential functions. A particularly interesting example of a potential field method for single body robots is to be found in [12]. This introduces a model which simulates steady state heat transfer with variable conductivity to find *optimal paths* by minimizing thermal resistance. A powerful consequence of this method is that it allows a separate search of the translation and rotation components of the object, significantly reducing the problem dimension.

3.2 Rapidly Exploring Random Trees

Since many path planning problems have been shown to be *PSPACE-hard* [1], it is not surprising that a moderately large collection of probabilistic techniques to solve path planning problems exist. The most prevalent examples

of such techniques are the Probabilistic Roadmap Methods (Briefly discussed in Section 2.4), Randomized Potential Fields (discussed in Section 3.1.2) and Rapidly Exploring Random Trees (RRTs). As the name implies, an RRT planner solves the planning problem by building a tree (a connected, acyclic graph) in the free configuration space of a robot. In the standard RRT, this tree is rooted at q_{init}, the initial configuration, and is grown until some node falls within $Q_{goal} \subset \mathcal{C}$, the goal region.

The key feature of RRT is the manner in which new nodes are added to the tree at each step [9]. This is performed as follows: a random point is generated in $\mathcal{C}_{free}$, and the node closest to this random point in the existing RRT is selected. Then out of the possible controls available a feasible one is chosen (usually also one that minimizes distance to the random point) and the node is extended by integrating the state transition equation for a small fixed time δ (but never so long as to go beyond the chosen random point). Thus a new node is added to the tree, and the node from which it originated is considered its parent. The chosen control is also stored and associated with the edge between the parent and the new node.

Clearly once some node falls within the goal region, one can backtrack up the tree to the root, generating the path to the goal in reverse. The controls together specify a sequence of controls that will drive the system to a state in the goal region. Figure 4 presents an example of RRTs applied to *Point Planning* with straight line control.

RRTs enjoy the following properties:

1. An RRT is a minimal connected structure in the search space [9]. This means that under reasonable assumptions the branches of an RRT will never cross (simply because they were selected to grow outwards from the nearest neighbour).

2. The space does not need to be discretized when applying RRT. This is a major advantage as there is no need for an expensive precomputation step.

3. The RRT algorithm is very simple to implement (indeed, of the methods reviewed here it is undoubtedly the simplest to implement in MATLAB).

4. A single RRT can be grown from q_{init} to search for many different goal configurations.

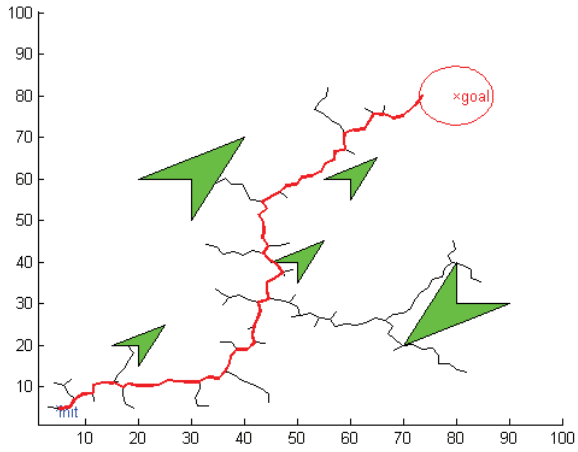

Figure 4: RRT applied to a point planning problem in a 2D space.

5. RRT is probabilistically complete [3]. This means that given enough time the RRT algorithm will *always* find the goal.

6. An RRT is *biased* towards free space. This is the characteristic that sets RRTs apart from other methods and makes them very well suited to high dimensional path planning problems. A mathematical realisation of this fact is provided in Subsection 4.

RRT methods cannot, however, be viewed as a *panacea* to the path planning problem. While they are a powerful method for quickly finding feasible paths in high dimensional spaces, they are not particularly suited, for instance, to the obtaining of optimal paths (although they have been considered for this - see [13]). Moreover, RRT methods generate non-smooth paths (although some classical methods of path smoothing can be employed on the path after it has been generated). If the planner has to be reused to drive several different starting points to the same goal point, then one would have to grow new trees from each different starting point. One way of avoiding this is by growing a single tree with reverse dynamics from the goal point until it reaches the given starting points.

Another drawback of RRT is the cost of adding a vertex to the tree. A naive nearest neighbour query takes $O(n)$ time where n is the number of vertices in

the tree. This situation can be improved however (see [14]) in order to make approximate nearest neighbour queries near $O(\log n)$ time.

3.2.1 The Voronoi Bias

Given a set, $\mathcal{S}$, of points in space, one can build a structure called the *Voronoi diagram* associated with $\mathcal{S}$. The *Voronoi cell* of some point $s \in \mathcal{S}$ is defined as the set of points in space that are closer to s than to *any* other element of $\mathcal{S}$. The union of Voronoi cells for all the elements of $\mathcal{S}$ partitions space into the so called *Voronoi diagram* of $\mathcal{S}$. These diagrams are useful in constructing meshes for studying various problems or for understanding the distribution of points in space. In [9] Voronoi diagrams were used to illustrate a desirable property of RRTs, known as *Voronoi Bias*.

Voronoi Bias refers to the inherent *bias* of the RRT algorithm towards exploring free space. To understand the meaning of the phrase consider an RRT growing in a bounded manifold $\mathcal{C}_{free}$ with a metric, ρ. The nodes, $\{r_1, \ldots, r_n\}$ in the RRT are points in $\mathcal{C}_{free}$. Each of these points generates a *Voronoi Cell*, $\mathcal{V}(r_i) = \{w \in \mathcal{C}_{free} | \rho(r_i, w) \leq \rho(r_j, w),\ \forall j \neq i\}$. The cells partition $\mathcal{C}_{free}$ into the Voronoi Diagram $\{\mathcal{V}(r_i)\}_{i=1}^{n}$. Each cell has a Lebesgue measure, $\mu(\mathcal{V}(r_i))$. Now given a random point, $x \in \mathcal{C}_{free}$, the probability that $x \in \mathcal{V}(r_i)$ is given by

$$\mathbb{P}[x \in \mathcal{V}(r_i)] = \frac{\mu(\mathcal{V}(r_i))}{\mu(\mathcal{C}_{free})}.$$

Clearly, x has a higher probability of being in one of the larger Voronoi cells. This in turn would cause the point in r generating that cell to be extended, thus splitting the cell. Hence in a sense the RRT algorithm splits Voronoi cells as quickly as possible.

Figure 5 shows a simulation of an RRT in MATLAB and its associated Voronoi Diagram. Notice that there are no unusually large cells in the diagram. This illustrates the result that the distribution of vertices in the RRT approaches a uniform distribution.

3.2.2 Other Extensions

There are many extensions to the basic RRT algorithm in the literature, some of these are briefly outlined below.

In [15] a de-randomized version of RRT is derived by selecting k random points at each step instead of just one. Efficient nearest neighbour queries in

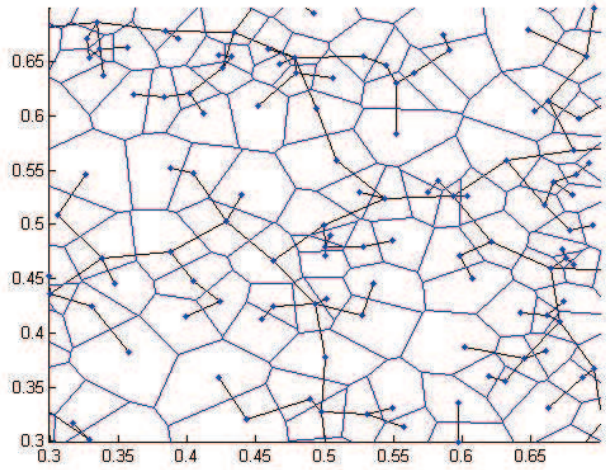

Figure 5: The Voronoi Regions associated with the nodes of an RRT in the plane.

the typical manifolds found as configuration spaces in path planning problems are developed in [14], enabling a logarithmic time nearest neighbour step in the RRT algorithm and hence faster convergence. A technique for biasing the random distribution for finding nodes in the RRT that produces more optimal paths is considered in [13]. In [1] a method (bidirectional RRT) in which two trees are grown is described, one from the source and the other from the target, with one tree growing towards a random point on a given iteration and the other growing towards the first tree. The roles of the trees are interchanged at each step. Experiments have shown that this planner produces better results for target query planning than standard RRT [1]. In [16] a more greedy algorithm is developed that does not limit the step length like in the standard RRT. This method turns out to be successful for Holonomic planning.

4 Theoretical Results on RRT in a large disc

In [2] the limiting case of Holonomic RRT path planning with $\mathcal{C} = \mathcal{C}_{free}$ a large n dimensional ball was considered. In particular, it was of interest to determine the behaviour of the RRT as the radius of the ball tends to infinity.

It was observed experimentally in [2] that in this limiting case, the RRT tends to grow outwards in $n+1$ branches, and that these branches tend to touch the vertices of a regular $(n+1)$-simplex. What follows constitutes a theoretical framework in which to consider this behaviour.

4.1 The Two-Dimensional Case

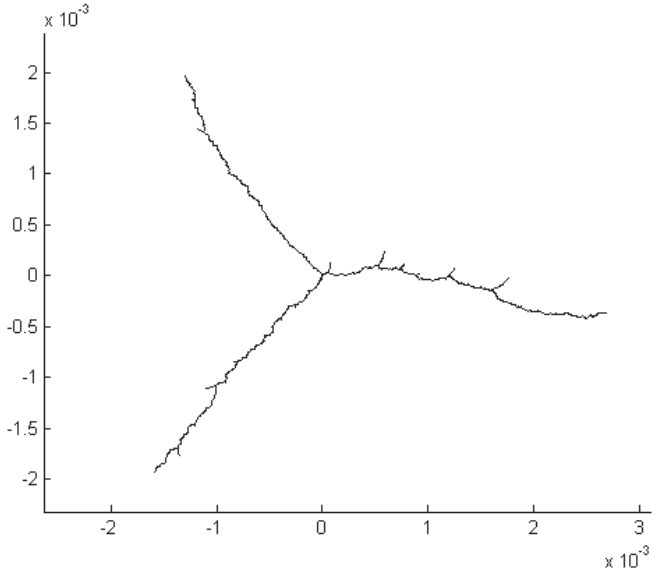

Figure 6: An RRT in a large disc: $\delta = 0.00001, \lambda = 1000$.

This section is dedicated to the 2D case of the above problem, where the control set $\mathcal{U}(x)$ is just the set of straight lines of length δ falling entirely in $\mathcal{C}_{free}$ and emanating from x. A model will be derived that captures the behaviour of the RRT in the limiting case as the radius of the disc approaches infinity.

Mathematical clarifications for the experimental results obtained in [2] will be sought. In particular, the 2D RRT in a very large disc will be shown to tend to grow in three main 'spokes' outward, and that after a large number of iterations, the three extreme points can be connected to form an equilateral triangle.

Definition 1. *Let $\mathcal{R}_\lambda(m)$ be an RRT of m vertices grown in a disc of radius λ. Let r be the root node of the RRT and $D = \{x \in \mathbb{R}^2 \colon ||x - r|| < \lambda\}$ also referred as the domain of the RRT.*

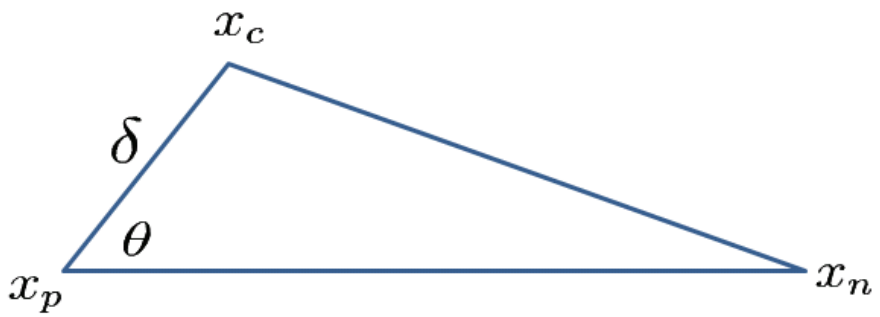

Figure 7: Adding a new vertex, where the parent is x_p and the child x_c.

Lemma 1. *Given vertices $x_p, x_c \in \mathcal{R}_\lambda(m)$ (not necessarily adjacent), and given an x_n from the uniform distribution. Let θ be the angle formed between x_c and x_n at x_p (See Figure 7). Then in the limit as $\lambda \to \infty$ if $\theta < \frac{\pi}{2}$, the probability that the RRT algorithm will not extend x_p towards x_n approaches 1.*

Proof. The cosine rule is applied to the triangle formed by the displacements (See Figure 7) to obtain:

$$||x_c - x_n||^2 = ||x_p - x_n||^2 + \delta^2 - 2\delta||x_p - x_n|| \cos(\theta) \ . \tag{1}$$

From this the difference of the squares of the distances can be computed after first rearranging the terms. The sign of this value determines which one is smaller, and hence which one the algorithm would prefer:

$$\begin{aligned} \text{sign}(||x_c - x_n||^2 - ||x_p - x_n||^2) &= \text{sign}(\delta(\delta - 2||x_p - x_n|| \cos(\theta))); \\ &= \text{sign}(\delta - 2||x_p - x_n|| \cos(\theta)); \\ &= \text{sign}(\delta - 2N \cos(\theta),) \\ &=: s(\theta) \end{aligned}$$

where N gives the distance from x_p to the new point. Notice that if $\cos(\theta) < 0$ then $s(\theta) = 1$.

Now consider the case where $\cos(\theta) > 0$. Let $0 < K < \lambda - ||x_p - r||$, then the circle of radius K and centre x_p is contained in D. The Lebesgue measure of this circle is just its area: πK^2, and the probability that a point from the uniform distribution on D falls in this circle is thus given by $\dfrac{\pi K^2}{\pi\lambda^2} = \dfrac{K^2}{\lambda^2}$. Now choose $K = \dfrac{\delta}{2\cos(\theta)}$. Then $K > 0$ since $\cos(\theta) > 0$, and for sufficiently large λ, $K < \lambda - ||x_p - r||$. Hence, $N < K$ with probability $\frac{\delta^2}{4\lambda^2\cos^2(\theta)}$. This approaches zero as $\lambda \to \infty$. Clearly, then, as $\lambda \to \infty$, $\mathbb{P}[N \geq K] \to 1$. Hence

$$\lim_{\lambda\to\infty} s(\theta) = \begin{cases} 1 & \text{if } \cos(\theta) < 0; \\ -1 & \text{if } \cos(\theta) > 0. \end{cases} \tag{2}$$

At all times, when $|\theta| > \frac{\pi}{2}$, the algorithm will not extend x_c. Further in the limiting case as $\lambda \to \infty$, when $|\theta| < \frac{\pi}{2}$, x_p will not be extended.

□

Note that setting $\lambda = \infty$ does not make sense in practice. The proof above provides a means, however, of determining the probability that the algorithm will differ from the limiting case. Increasing λ will decrease this probability.

As an example of how Lemma 1 may be applied, consider the case of a tree with two nodes where x_p is the root node and x_c is the only other node in the tree. Then the previous lemma gives the conditions for the behaviour of the RRT on the next step as:

1. Extend x_p towards x_n if $\theta > \frac{\pi}{2}$;
2. Extend x_c towards x_n if $\theta < \frac{\pi}{2}$;

with probability approaching 1 as $\lambda \to \infty$.

Interestingly, for any given finite λ there will be an interval of values for θ for which this decision mechanism breaks down. Given $0 < \gamma < 1$, there exists ϵ such that the probability of failing is greater than γ if $\theta \in (\frac{\pi}{2} - \epsilon, \frac{\pi}{2})$. Increasing λ will decrease ϵ. It is clear from the proof of Lemma 1 that a good choice of λ should satisfy $2\lambda\cos(\frac{\pi}{2} - \epsilon) \gg \delta$.

Lemma 2. *In the limit as $\lambda \to \infty$, no vertex in $\mathcal{R}_\lambda(m)$ has degree greater than 3.*

Proof. Consider a vertex, c, of the RRT. Assume it has four neighbours, $\{v_1, v_2, v_3, v_4\}$ and that they are taken in anti-clockwise order, let the angles between them be $\theta_1, \theta_2, \theta_3$ and θ_4. Clearly $\sum_{i=1}^{4} \theta_i = 2\pi$. Now x_c must have been extended towards each one of the neighbours in turn, and in particular for each neighbour added, x_c must have been preferred by the algorithm over all the other neighbours. Thus the angle between any two of the neighbours of x_c must be greater than $\frac{\pi}{2}$ by the previous Lemma. Hence $\sum_{i=1}^{4} \theta_i > 4\frac{\pi}{2} = 2\pi$, contradiction.

Therefore, each vertex can have degree no greater than 3. □

An obvious consequence of the above lemma is that in the limit as $\lambda \to \infty$, no vertex in $\mathcal{R}_\lambda(m)$ except the root can have more than 2 children. Experiments have shown that for modestly large values of λ, this result still holds approximately.

It follows from Lemma 2 that if a node has degree three it will not be involved in any further nearest neighbour tests in the limiting case. This gives rise to the following definition.

Definition 2. *We call a node $v \in \mathcal{R}_\lambda(m)$ an **Active Node** if its Voronoi region is unbounded. Otherwise we call it an **Inactive Node**.*

In the limiting case, the probability of selecting an *Inactive Node* for extension goes to zero as $\lambda \to \infty$ since its Voronoi region remains of a fixed area, whilst the area of D approaches ∞. Hence the name *Inactive Node*.

This and Lemma 2 imply that in the limiting case any node of degree 3 in the RRT has a finite Voronoi region.

Note that the RRT algorithm can run until a node that has degree 3 has been constructed. It is a simple matter to verify that this will eventually occur. For what remains, assuming there is some vertex of degree 3 in the RRT and, without loss of generality, assuming this to be the root vertex, r (if it is some other vertex, the tree can be rotated so that it is the root).

Definition 3. *Let $x \in D$. The **angle** of x is defined to be the angle taken between the positive x-axis rooted at r and the line connecting r and x. This is denoted by* $\arg(x)$.

In the limiting case, when x is uniformly selected, $\arg(x)$ actually gives us the angle to x from any point in the RRT, as illustrated in the following Lemma.

Lemma 3. *Let* $\arg(C; A)$ *denote the angle between the positive* x*-axis rooted at* A *and the line connecting* A *and* C*. Then for* $C \in unif(D)$ *and* A, B *any points in* D, $\arg(C; A) \to \arg(C; B)$ *as* $\lambda \to \infty$.

Proof. The sine rule on $\triangle ABC$ (See Figure 8) will be applied.

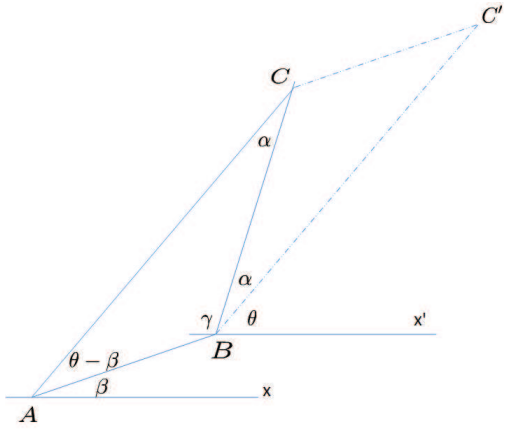

Figure 8: The angle to C from A and B.

Construct point x' to complete the parallelogram $Axx'B$. Let θ be the angle from the $x-axis$ at A to x and let β be the angle at A from the $x-axis$ to B. Let α be the angle $\angle CBC'$, then by parallel lines $\angle ACB = \alpha$.

Now applying the sine rule on $\triangle ABC$:

$$\frac{\sin(\alpha)}{AB} = \frac{\sin(\theta - \beta)}{BC}.$$

Also

$$\sin(\alpha) = \sin(\theta - \beta)\frac{AB}{BC} < \frac{AB}{BC}.$$

Now it can be shown by a similar argument to that used in the proof of Lemma 1 that as $\lambda \to \infty$, and for any $K > 0$, $\mathbb{P}[BC > K] \to 1$. Hence as $\lambda \to \infty$, and for any $\epsilon > 0$, $\mathbb{P}[\frac{AB}{BC} < \epsilon] \to 1$. And so with full probability, $\lim_{\lambda\to\infty} \sin(\alpha) = 0$, and since sin is continuous, $sin(0) = 0$ and α is acute, then $\alpha \to 0$ as $\lambda \to \infty$. Hence $\arg(C; A) \to \arg(C; B)$ as $\lambda \to \infty$. □

The next objective is to determine the criterion for finding the nearest neighbour in the existing tree to a point $x \in unif(D)$ as $\lambda \to \infty$. It turns out that this result is only dependent on the angle of x.

Theorem 1. Selection Criterion
Given $x \in unif(D)$, and vertices $\{v_1, v_2, \ldots, v_n\}$ in the RRT, with the root node at the origin. In the limiting case as $\lambda \to \infty$, the nearest neighbour of x is that vertex v_i maximizing $v_j \cdot \frac{x}{||x||}, j = 1, 2, \ldots, n$.

Proof. Let $x = (N\cos(\theta), N\sin(\theta))$. and $v_j = (a_j\cos(\alpha_j), a_j\sin(\alpha_j))$ for each j. Then

$$x - v_j = (N\cos(\theta) - a_j\cos(\alpha_j), N\sin(\theta) - a_j\sin(\alpha_j)).$$

Hence

$$||x - v_j||^2 = N^2 + a_j(a_j - 2N\cos(\theta - \alpha_j))$$

and

$$||x - v_i||^2 - ||x - v_j||^2 = a_i^2 - a_j^2 + 2N(a_j\cos(\theta - \alpha_j) - a_i\cos(\theta - \alpha_i)).$$

Now by a similar argument to that used in the proof of Lemma 1, it can be shown that in the limiting case $\lambda \to \infty$, the last term in the above equation dominates the result. Thus in the limiting case, v_i will be closer to x than v_j only when

$$a_j\cos(\theta - \alpha_j) - a_i\cos(\theta - \alpha_i) < 0,$$

i.e. only when

$$a_j\cos(\theta - \alpha_j) < a_i\cos(\theta - \alpha_i).$$

So in the limiting case, v_i will be the closest point to x in the set $\{v_1, ..., v_n\}$ if and only if $a_i\cos(\theta - \alpha_i)$ is maximized.

Now notice that this quantity can be reduced as follows:

$$\begin{aligned} a_i\cos(\theta - \alpha_i) &= a_i\cos(\theta)\cos(\alpha_i) + a_i\sin(\theta)\sin(\alpha_i); \\ &= v_i \cdot (\cos(\theta), \sin(\theta)); \\ &= v_i \cdot \frac{x}{||x||}. \end{aligned}$$

□

This theorem stipulates how to code simulations of the limiting case directly (as opposed to just making λ large). Several simulations were run, and the results were similar to those obtained when running simulations with

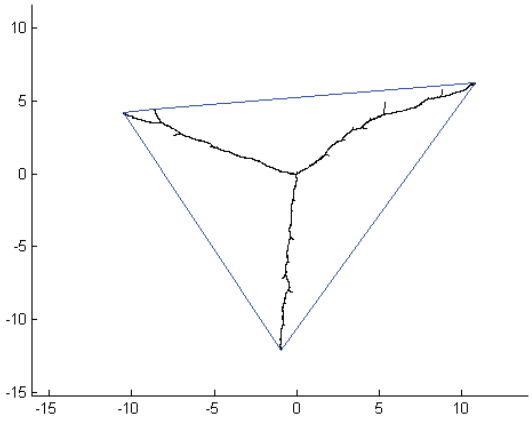

Figure 9: An RRT with infinite disc simulation using Theorem 1 and its convex hull.

the original RRT algorithm in a disc with $\delta \ll \lambda$. See Figures 6 and 9 for comparison.

Theorem 1 shows that the choice of nearest neighbour in the limiting case depends on the angle of x but not its magnitude. Indeed in the computation $\frac{x}{||x||}$ was used, which belongs to the unit circle, $\mathcal{S}^1$. Thus the decision process can be thought of as taking as input a point in $\hat{x} \in \mathcal{S}^1$ and returning the node v in the tree maximizing $\hat{x} \cdot v$. This leads to a construct which will be termed the *Connect Region Diagram*.

Definition 4. *Let $\mathcal{R}$ be the RRT being constructed by the method of Theorem 1. For each node $v \in \mathcal{R}$ let $\mathcal{C}(v)$ be the set of $\hat{x} \in \mathcal{S}^1$ that would be classified nearest to v by the method of Theorem 1. The class $\{C(v)|v \in \mathcal{R}\}$ forms a partition of $\mathcal{S}^1$ which is called the* ***Connect Region Diagram****.*

Notice that this is similar to a Voronoi Diagram, except that $\mathcal{S}^1$ is partitioned based on elements not in $\mathcal{S}^1$. Indeed in this case $v \notin C(v)$, in general. Also take note that if v is an inactive node, then $\mathcal{C}(v) = \emptyset$. In fact it is easy to verify that $\mathcal{C}(v) = \emptyset$ precisely when v is an inactive node.

Lemma 4. *The locus of the projections of v onto all the points in $\mathcal{S}^1$ is a circle passing through 0 and v, which is termed the* projection circle *of v. The projection circle of v is denoted by $P_c(v)$.*

Proof. Clearly the points 0 and v must be on this locus, corresponding to angles equal to that of v and perpendicular to it. Further, for any other point, the projection x_2 completes a right angled triangle $\triangle ovx_2$, with right angle at x_2. Hence these are angles in a circle with ov as diameter, and centre $\frac{v}{2}$. Another way of seeing this is by using congruent triangles as in Figure 10 to show that the distance to the projection from $\frac{v}{2}$ is always equal to $||\frac{v}{2}||$. So the locus of these projections, $P_c(v) = \{\text{proj}_u(v) | u \in S^1\}$, is a circle centered at $\frac{v}{2}$ with radius $||\frac{v}{2}||$.

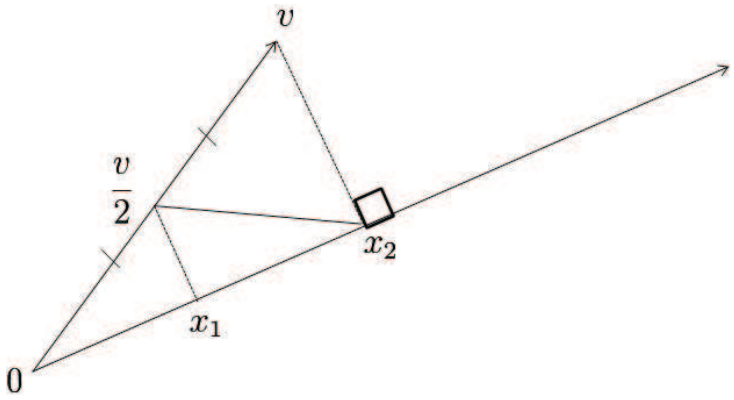

Figure 10: The length of the line from $\frac{v}{2}$ to x_2 is equal to $||\frac{v}{2}||$.

□

Lemma 5. *For each $v \in \mathcal{R}$, $\mathcal{C}(v)$ is connected.*

Proof. (Sketch)

Clearly for any element, $\hat{x}$ of $\mathcal{S}^1$, the quantity $\hat{x} \cdot v$ is the length of the line segment starting at the origin, passing through $\hat{x}$ and ending on $P_c(v)$. Clearly v that maximizes this quantity for a given x is sought. Recall that o is the root node of the RRT, and it satisfies $\hat{x} \cdot o = 0$, $\forall\, \hat{x} \in \mathcal{S}^1$. Thus for any $v \neq 0$, v can only be considered for maximizing $\hat{x} \cdot v$ if this quantity is positive - i.e. if the line passing $\hat{x}$ intersects $P_c(v)$ on the same side of the origin that it intersects $\hat{x}$.

Now consider the figure formed by the union of the projection circles of all the points in the RRT, taken together with their interiors. It is clear

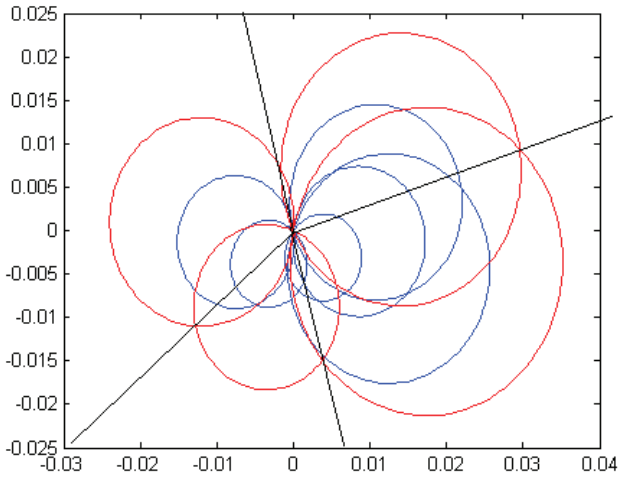

Figure 11: Boundary circles correspond to active nodes.

from construction that those circles with points on the boundary of this union constitute the 'active nodes' (presented in red in Figure 11)

No new projection circle can intersect the boundary of the Figure more than twice. Indeed, let Γ_0 be any circle that intersects the Figure at least two times. Since any two circles intersect at most twice and all the projection circles already intersect at the origin, Γ_0 must intersect at least two other projection circles, say Γ_1 and Γ_2. Let the unique point of intersection (other than the origin) with Γ_1 be A and likewise let the unique point of intersection with Γ_2 be B. Then since 0 is contained within all circles, we must have that the path on Γ_0 from 0 to A is contained in Γ_1, and likewise the path from 0 to B is contained in Γ_2 (See Figure 12). Hence the arc on Γ_0 moving through A, 0 and B is completely contained in the figure. Likewise if at some point on the opposite arc, Γ_0 were to enter the figure, it would intersect with some circle Γ_3, and would not be able to escape that circle until returning to the origin - a contradiction (because it must re-enter the figure at the other two circles). Hence the outer arc is completely outside of the figure.

This shows that the set of points on the projection circle of v that is also on the boundary of the figure is connected. It is easy to find a homeomorphism (via straight lines through the origin) mapping this set to $\mathcal{C}(v)$. Hence since connectedness is a topological property, $\mathcal{C}(v)$ is connected.

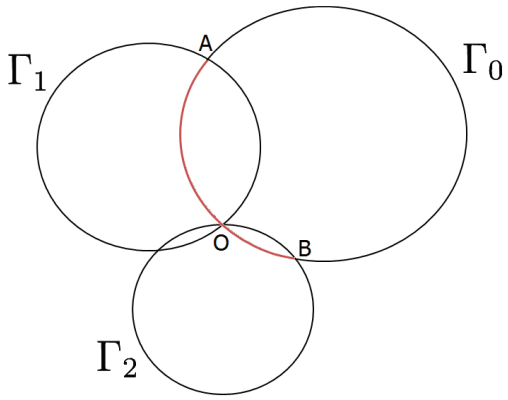

Figure 12: Γ_0, Γ_1 and Γ_2 .

□

Thus the Connect Region Diagram can be treated as a *pie chart* giving the points on $\mathcal{S}^1$ nearest to each vertex in the RRT. For each active node, v, since $\mathcal{C}(v)$ is connected it has a central angle. Denote this angle at the n^{th} step by $\Theta_n(v)$. Then the probability of choosing v on this step is $\frac{\Theta_n(v)}{2\pi}$.

An immediate observation is that if v becomes inactive at step n, then $\Theta_m(v) = 0, \ \forall m \geq n$. Another relatively simple result is that Θ_m is non-increasing for all m.

From the above argument it is clear that with a given active node, one can associate with it a projection circle. Further each active projection circle either intersects no other circles on the boundary of the figure, or two such circles. The case with no intersections is degenerate, hence it is neglected.

Definition 5. ***Active neighbours*** *of an active node, v, are defined to be the two active nodes w_1 and w_2 corresponding to the two* neighboring *circles when the projection circle diagrams are drawn.*

After the algorithm has run for long enough, $||v|| \gg \delta$. This means that if v is extended by δ it is changed only very slightly relative to its original position, and the only connect regions influenced are those of v, w_1 and w_2. All of them are either left unchanged or decreased, whilst the region of the new node will be slightly larger than what v was before. This can be visualized as v exerting a *force* on its neighboring circles proportional to the size of $\mathcal{C}(v)$.

An outline of how a proof of the main result might be obtained is now provided. If for some node v, its active neighbours both have larger connect regions than itself, then their connect regions will tend to grow, decreasing the connect region of v still further. Thus in some sense the number of active nodes must tend towards being minimal as the distance from the root gets larger. But there cannot be less than three such active nodes, because then the origin could not be completely covered, and there would always be a chance for a new active node to grow there.

Further, given that there are three active nodes, the case in which they have similar radii and are equally spaced apart is in some sense an equilibrium state. In this state each active node attempts to grow into the active regions of its neighbours with the same probability, and each growth step is roughly the same size. This means that the radii and angles will be kept similar. If the radii are vastly different at some step, but the connect regions have equal sizes, then the system is not in equilibrium because even though each active node has an equal chance of being extended, the resulting changes in the Connect Region Diagram would be different. This is because a small circle causes more change in connect regions when altered by a constant δ than does a large circle, so the small circle will be inclined to grow into the connect region of its neighbours. This in turn will cause its connect region to be increased, causing it to be selected more often, and hence for its radius to increase. Conversely, a large circle affects a smaller change in connect region diagram, so one would expect the region to decrease over time until the radii are equal. In the case of equal radii, geometric arguments should suffice to show that the angles must also approach equality.

The last part of this article sketches a possible proof. The first step towards a formal derivation will be to find precise description of how the connect regions change when some node is extended by δ in a given direction. A technique that might be successful here is to model the system in the complex plane.

4.2 Higher Dimensions

Results in 2D are obtained by first associating a random point in a large disc with its angle alone. This is analogous to associating a random point in n dimensions with a point on S^{n-1}. Theorem 1 has an analogous form in this case: for a random point $x \in S^{n-1}$, choose the vertex v_i that maximizes $v_i \cdot x$. The behaviour in n dimensions as $\lambda \to \infty$ can thus be easily simulated. The definition of an active node as a node with an unbounded Voronoi region

extends naturally to any metric space.

The Connect Region Diagram on $\mathbb{R}^n$ becomes a partition of S^{n-1} and $\mathcal{C}(v) = x \in S^{n-1}\ s.t.\ x \cdot v > x \cdot w,\ \forall w \in \mathcal{R}\backslash\{v\}$. An analysis of this structure could possibly shed light on the analysis of higher dimensional cases.

5 Suggested Extensions to RRT

5.1 Locally Biased RRT

As illustrated by the RRT-in-a-disc model above, the standard RRT search tree tends to grow outwards towards the boundary of the disc, biased towards free space, without exploring the nearby space much. This behaviour is desirable on the whole, but there are some circumstances where it would be inefficient:

- Consider an example where a point robot is situated in the centre of a very large plane. The robot has to navigate to a goal location relatively close by. The standard RRT algorithm would behave like the limiting case and search outwards in a long, sparse structure. As such it would have a poor chance of finding the goal.

- Consider as a second example the situation depicted in Figure 13. As ϵ is made smaller, the standard RRT is less and less inclined to traverse the corridor. With some local bias the chances of traversing the corridor are increased, and once the tree has entered the corridor, the local bias is likely to pull it through to the other side (particularly if using adaptive random walks).

Thus there is some justification in biasing the distribution function in the region to favour local points slightly over more distant points. (It is important to keep this bias slight, as a large bias would cancel the Voronoi bias of the RRT, which is one of its strongest features). This leads to a Locally Biased RRT approach. There are several options as to how to implement local bias in the RRT distribution:

- Choose points based on a normal distribution with centre at the root node and with a very large standard deviation (on the order of the size of the whole region). This solution is not ideal, however, as the bias is statically associated with the root node, even when the remainder of the tree has changed.

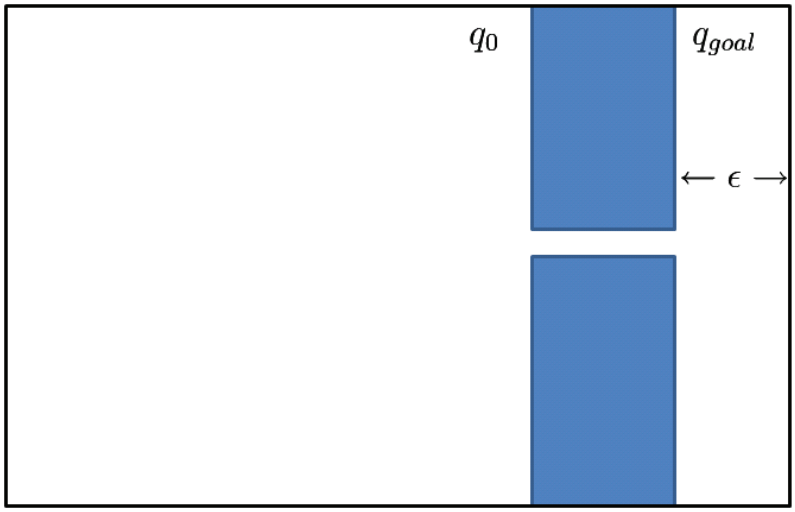

Figure 13: A situation which causes bad performance in the standard RRT.

- Use a weighted sum of Gaussian distributions centred at each node of the tree. This seems feasible in terms of introducing a local bias that is not static, but it would be impractical from an implementation point of view as the cost would increase with the size of the tree.
- Choose a uniform distribution with probability $1 - \mu$ and a local distribution with probability μ where μ is a fixed number representing the probability of performing local search at a given step. The expected number of steps between performing local search is $\frac{1}{\mu}$. The local distribution is a Gaussian distribution centred at a randomly chosen vertex of the RRT. This can be combined with goal-biased RRT by including a separate, small probability $\nu \ll 1$, with $\mu + \nu < 1$ and let the distribution be a Gaussian distribution centred at x_{goal} with probability ν. Let normal(C_{free}, v_{rnd}) denote a normal distribution on C_{free} centred at a random vertex of the RRT with prescribed variance. The choice of distribution is then a function of the random variable X:

$$dist_X = \begin{cases} \text{normal}(C_{free}, v_{rnd}), & \text{if } X \leq \mu; \\ \text{normal}(C_{free}, x_{goal}), & \text{if } \mu < X \leq \mu + \nu; \\ \text{unif}(\mathcal{C}_{free}), & \text{otherwise.} \end{cases}$$

This biasing scheme can be easily applied to Bi-RRT. In [17] and [18]

a kind of local bias for RRT is introduced. The former by emulating a local 'flood fill' behaviour and the latter by restricting the distribution to the union of the dynamically visible regions surrounding each point.

- Apply a random walk planner [7] with an adaptive covariance matrix. The covariance changes so that, for instance, while walking down a thin horizontal corridor the chance of moving right or left increases, while the chance of moving up or down decreases. This method could be incorporated into an RRT, by keeping a covariance matrix at each (active) node and letting the local search be a single step of the Adaptive Random Walk algorithm proposed in [7].

5.2 Modified Nearest Neighbour Search for Holonomic RRT

The nearest neighbour search is a central part of the RRT algorithm. If performed naively, it uses $O(k)$ work on each step, where k is the size of the RRT. Hence, when building an RRT of n vertices, the Nearest Neighbour algorithm contributes $O(n^2)$ amount of work, which can be very large. This situation cannot be improved much if the exact nearest neighbour of each point must be found. There are approximate nearest neighbour schemes [14] that can do the work in around $O(\log(k))$ per step, and hence $O(n \log(n))$ overall. However, if some nodes could be removed from consideration on certain steps, then this should benefit all methods.

The motivation for this algorithm comes primarily from the notion of *active node*, which arose when dealing with the limiting case. It is easy to keep track of active nodes in the tree, and when the neighbour for selection is sufficiently far away from the remainder of the tree (which is especially true while the tree is growing in a large space) all other nodes can be eliminated from consideration.

At each stage in the RRT algorithm, the following information can be made cheaply available:

1. The centroid of the RRT. This can be tracked in linear time;
2. The Euclidean diameter of the RRT or an upper bound thereof;
3. An upper bound on the Euclidean radius of a circle containing the Voronoi region of each inactive node.

For instance, given the centroid of the first n vertices, $\hat{x}_n$, and a new vertex r_{n+1}, the new centroid can be written immediately as:

$$\hat{x}_{n+1} = \frac{n\hat{x}_n + r_{n+1}}{n+1}.$$

The Euclidean diameter can be estimated by the sum of the distances of the two furthest vertices from the root (which can be kept track of as the algorithm progresses).

For vertices with three neighbours, an upper bound on the Voronoi region is given by the area of the Voronoi triangle formed between those three neighbours and the vertex. This information can be stored with the vertex. If the vertex gets a new neighbour it can be updated.

A list of vertices of degree less than 3 is also maintained (candidates for active vertices).

The nearest neighbour procedure can thus be subjected to the following improvements:

Let x be the random point generated.

1. If x is further from the root than the Euclidean diameter, only compare x to the active nodes.

2. If x is further from the centroid than the Euclidean diameter, only compare x to the active nodes.

An interesting modification to the algorithm would be to choose the active node with the highest estimated Voronoi area in this case and extend it regardless of whether it is actually closest to x. In this case, then, the selection is in constant time. This latter case also looks a bit like a local biasing. It would make interesting further research to investigate whether the method provides improvements over standard RRT.

5.3 A new method for improving path optimality

Because RRT is a fast and randomized algorithm, there is perhaps some sense in running it several times to generate paths with different topologies through a space. If performed judiciously, these repeated runs can be utilized to produce optimal looking paths.

Define the cost of a polygonal path as the sum of some functional over that path (much as in the case of Dynamic Programming). Simulated Annealing can

then be used to generate optimal paths as follows: The standard Metropolis algorithm is used, with the neighbour function either permuting slightly one of the elements in the path (with high probability) or planning an entirely new path using a method like RRT (with low probability). After applying either of these operations, a path smoothing algorithm is applied, which eliminates unnecessary jumps in the path.

The Simulated Annealing Approach was tested in 2D discretized point planning environments using a randomized planner akin to a naive Probabilistic Road Maps Method. Figure 14 illustrates some of the different paths considered by the Simulated Annealing method. Notice how the topologies are different in some cases.

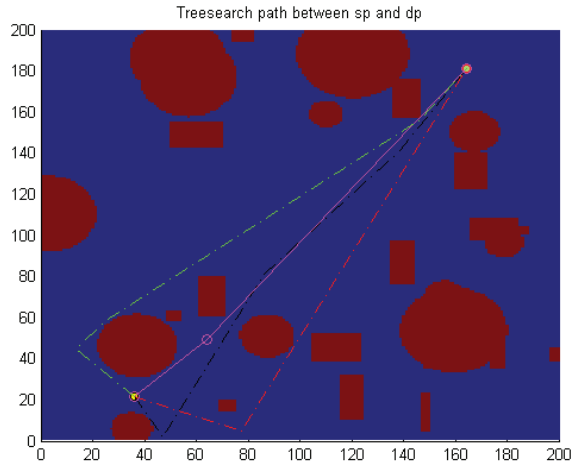

Figure 14: Different Paths available for selection by the Simulated Annealing Method.

6 Further Experiments

Consider the problem of navigating a rectangle through a field of obstacles. The problem is Holonomic, since we can completely control the rectangle's configuration at any step. The workspace in this instance is two dimensional - some bounded region in $\mathbb{R}^2$. Since the rectangle is free to translate to any

position in the plane and rotate by any angle, its configuration space is a 3D bounded subspace of $\mathbb{R}^2 \times \mathcal{S}^1$. See Figure 15.

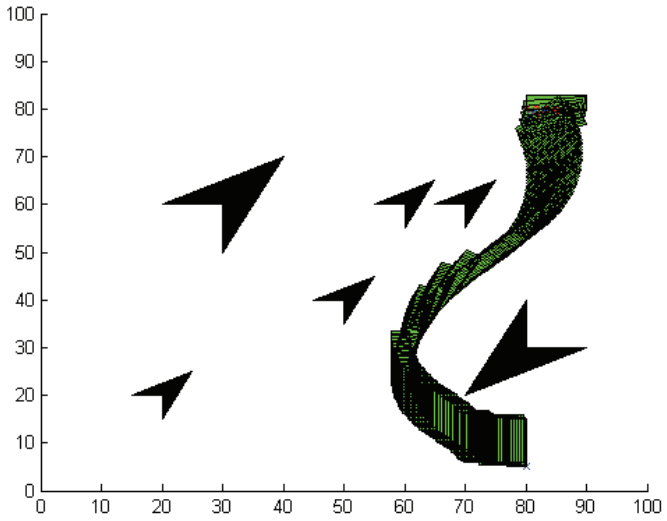

Figure 15: An example of a path planned for a 2D rectangle.

The following methods were employed to find a path for the rectangle.

1. Basic RRT;

2. RRT with local and goal bias;

3. The Numerical Potential Field method from [3];

4. A Numerical Potential Field method finding a harmonic function in the $\mathcal{W}$-space and extends it by the method of [3].

The forward kinematic function for the rectangle was computed, and KKT optimization was used to find the necessary suprema in the method from [3] to find the $\mathcal{C}$-space granularity, Δ. The code was all written in MATLAB.

6.1 Results and discussion

The Numerical Potential Field algorithm which was harmonic in the workspace worked particularly well for this problem. On all initial orientations attempted, the algorithm returned a path that was both smooth and avoided obstacles.

The potential field planner following [3] could find the goal from some locations and orientations, but was extremely poor at doing this from others. In particular, if the rectangle was started in a half-rotated state it had trouble.

Both the standard RRT and the RRT with local bias found paths for the robot. In terms of the number of iterations required the standard RRT outperformed the locally biased RRT in a normal to slightly cluttered grid, but the locally biased RRT seemed to flourish when the grid was either empty or full.

It might be interesting to test these algorithms in more complex scenarios by using, for instance, the *Motion Strategy Library* [19].

7 Conclusion

The first part of this paper provided a review of robot path planning and some of the techniques available in the literature, with specific emphasis on Artificial Potential Fields and RRT. A careful consideration of the former, particularly the methods proposed in [3] and [11] led to the idea of creating $\mathcal{W}$-space harmonic potential functions, and the conditions under which these could be extended to harmonic functions on the $\mathcal{C}$-space. Some theoretical results on the performance of RRT in the limiting, Holonomic case were also considered. An algorithm for simulating the limiting case effectively was derived and a basic framework was created in which further reasoning should be possible.

The theoretical analysis and literary review led naturally into the description of three methods. The first, motivated in part by the preceding theoretical discussion, is the inclusion of a slight local bias into the RRT. The use of the adaptive random walks of [7] as a local step was considered but not implemented, and therefore is a candidate for future work. The second, motivated by some heuristic arguments, was an adaptation of the *nearest neighbour* selection scheme in RRT, leading to a Greedy RRT algorithm. Finally an optimization technique based on Simulated Annealing and relying on the ability of planners like RRT to find feasible paths quickly was discussed. It remains

to be investigated whether one of RRT or Adaptive Random Walk (or indeed some optimization heuristic) might be suitable as escapes in Potential Fields Methods, or whether the modified nearest neighbour method proposed for R-RT algorithms in this paper is feasible in practice. The harmonic workspace potential method has proven itself a fierce contender in the case of the toy example of moving a rectangle through an obstacle field, but it would need to be tested in more complex environments.

Acknowledgement:

We are very grateful to the following group members for their contributions and dedication at the initial stages of the Mathematics in Industry Study Group - South Africa (2009): Abdullahi R. Adem, Charis Harley, Masiala Mavungu, Innocent Rusagara, Linda Zwane, Tshifhango Ndaza, Kagiso Rapetswa, Musa Shabalala.

References

[1] LaValle, S.M. Planning algorithms, Cambridge University Press, 2006.

[2] Lavalle, S.M. and Kuffner, J.J. Rapidly-exploring random trees: Progress and prospects. In: Algorithmic and Computational Robotics: New Directions, 2000, pp. 293–308, .

[3] Barraquand, J., Langlois, B. and Latombe, J.C. Numerical potential field techniques for robot path planning. *IEEE Trans. on Systems, Man, and Cybernetics*, **22**, (1992), 224–241.

[4] Lavalle, S.M. From dynamic programming to RRTs: Algorithmic design of feasible trajectories. In: A. Bicchi, H.I. Christensen, and D. Prattichizzo, editors, Control Problems in Robotics, Springer-Verlag, 2002, pp. 19–37.

[5] Noakez, L. and Popiel, T. Geometry for robot path planning. *Robotica*, **25**, (2007), 691–701.

[6] Farber, M., Tabachinikov, S. and Yuzvinsky, S. Topological robotics: Motion planning in projective spaces, Int. Math. Research Notices, **34**, (2003), 1853 – 1870.

[7] Carpin, S. and Pillonetto, G. Motion planning using adaptive random walks. In: *IEEE Trans on Robotics*, (2005), pp. 3809–3814.

[8] Badawy, A. and McInnes, C.R. Robot motion planning using hyperboloid functions. In: Proc. of the 2007 World Congress on Engineering, 2007.

[9] Lavalle, S.M. Rapidly-exploring random trees: A new tool for path planning, TR 98-11, Computer Science Dept., Iowa State University, 1998.

[10] Saff, E.B. and Snider, A.D. Fundamentals of complex analysis with applications to engineering and science, Prentice Hall, Upper Saddle River, New Jersey, 2003.

[11] Connolly, C.I., Burns, J.B. and Weiss, R. Path planning using Laplace's equation. In: Proc of IEEE Int. Conf. on Robot. and Autom., 1990, pp 2102–2106.

[12] Wang, Y. and Chirikjian, G.S. A new potential field method for robot path planning. In: Proc. IEEE Int. Conf. on Robot. and Autom., 2000, pp. 977–982.

[13] Urmson, C. and Simmons, R. Approaches for heuristically biasing RRT growth. In: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, **2**, (2003), 1178 – 1183.

[14] Atramentov, A. and Lavalle, S.M. Efficient nearest neighbor searching for motion planning. In: Proc. IEEE Int. Conf. on Robot. and Autom., (2002), 632 – 637.

[15] Lindemann, S.R. and LaValle, S.M. Steps toward derandomizing RRTs, Robot Motion and Control, Lecture Notes in Control and Information Sciences, **335**, (2006) 287 – 300.

[16] Kuffner, J. and Lavalle, S. RRT-connect: An efficient approach to single-query path planning. In: Proc. IEEE Int. Conf. on Robot. and Autom. (ICRA'2000), April, 2000, 995-1001.

[17] Dept, M.K. and Kalisiak, M. RRT-blossom: RRT with a local flood-fill behavior. In: Proc. IEEE Int. Conf. on Robot. and Autom. (2006), 1237 – 1242.

[18] Yershova, A. Jaillet, L., Simon, T. and LaValle, S.M. Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain, In: *Proc. IEEE Int. Conf. on Robot. and Autom.*, (2005), 3867–3872.

[19] Motion Strategy Library. http://msl.cs.uiuc.edu/msl/.