Planning and Exploring Under Uncertainty

Elizabeth Murphy
Somerville College



Supervisor:
Dr Paul Newman

Robotics Research Group
Department of Engineering Science
University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity Term 2010

Elizabeth M. Murphy
Somerville College

Doctor of Philosophy
Trinity Term 2010

Planning and Exploring Under Uncertainty

Abstract

Scalable autonomy requires a robot to be able to recognize and contend with the uncertainty in its knowledge of the world stemming from its noisy sensors and actuators. The regions it chooses to explore, and the paths it takes to get there, must take this uncertainty into account. In this thesis we outline probabilistic approaches to represent that world; to construct plans over it; and to determine which part of it to explore next.

We present a new technique to create probabilistic cost maps from overhead imagery, taking into account the uncertainty in terrain classification and allowing for spatial variation in terrain cost. A probabilistic cost function combines the output of a multi-class classifier and a spatial probabilistic regressor to produce a probability density function over terrain for each grid cell in the map. The resultant cost map facilitates the discovery of not only the shortest path between points on the map, but also a distribution of likely paths between the points.

These cost maps are used in a path planning technique which allows the user to trade-off the risk of returning a suboptimal path for substantial increases in search speed. We precompute a probability distribution which precisely approximates the true distance between any grid cell in the map and goal cell. This distribution underpins a number of A* search heuristics we present, which can characterize and bound the risk we are prepared to take in gaining search efficiency while sacrificing optimal path length. Empirically, we report efficiency increases in excess of 70% over standard heuristic search methods.

Finally, we present a global approach to the problem of robotic exploration, utilizing a hybrid of a topological data structure and an underlying metric mapping process. A 'Gap Navigation Tree' is used to motivate global target selection and occluded regions of the environment (*'gaps'*) are tracked probabilistically using the metric map. In pursuing these gaps we are provided with goals to feed to the path planning process en route to a complete exploration of the environment.

The combination of these three techniques represents a framework to facilitate robust exploration in a-priori unknown environments.

Acknowledgements

First and foremost I would like to thank my supervisor, Paul Newman, for his tireless support, input and good humour. I benefited greatly from being allowed the freedom to explore interesting things while always having his wise words to set me back on track when things went pear-shaped. Thank you also to the members of the Mobile Robotics Group for their help and insightful discussions throughout the course of my research.

My research was funded initially by the Rhodes Trust, and latterly by the Fellowship Fund branch of the Australian Federation for University Women (QLD). I am ever grateful to both bodies for their support.

Thanks also to my family, who despite being half a world away offered me continual support and encouragement. Finally thanks to my mum and to Peter Hall, two people I've really missed over the last 4 years. Without their unfailing belief and encouragement I never would have chased this dream in the first place.

Contents

1 Introduction **1**

1.1 Motivation . . . 1
1.2 Outline of the Problem . . . 2
1.3 Assumptions . . . 3
1.4 Contributions . . . 4
1.5 Structure of the Thesis . . . 5
1.6 Publications . . . 7

2 Planning Algorithms - Comparison, Construction and Analysis **8**

2.1 Overview of the Path Planning Problem . . . 9
2.1.1 Combinatorial Planners . . . 12
2.1.2 Sampling-based planners . . . 14
2.1.2.1 Probabilistic Roadmap Planners . . . 14
2.1.2.2 Rapidly Exploring Random Trees . . . 16
2.1.3 Grid-based costmap planners . . . 18
2.1.3.1 Assumptive Techniques . . . 20
2.1.3.2 Partially Observable Planning . . . 23
2.1.4 Summary of the Planning Literature . . . 27
2.2 Cost Maps . . . 28
2.2.1 Summary . . . 30
2.3 A* Search . . . 30
2.4 D* Search . . . 37
2.5 Field D* Search . . . 52
2.6 Chapter Summary . . . 59

3 Systems to Support Autonomous Exploration **60**

3.1 Path Planning Systems . . . 61
3.1.1 The D* library . . . 62

3.1.2 Planning Processes . . . 65
3.2 The Costmap library and Costmap Processes . . . 66
3.2.1 Occupancy Grids . . . 66
3.2.2 Costmaps . . . 71
3.2.3 Costmap Processes . . . 72
3.3 Navigation . . . 73
3.3.1 SLAM . . . 74
3.3.2 Single Vehicle SLAM Formulations . . . 76
3.4 Chapter Summary . . . 81

4 Exploration 83
4.1 Motivation . . . 84
4.2 Background to the Exploration Problem . . . 85
4.2.1 Exploring by bumping into things - Early Work . . . 85
4.2.2 Exploring by making maps . . . 88
4.2.2.1 Grid Based Exploration . . . 89
4.2.2.2 Feature Based Exploration . . . 93
4.2.2.3 Topological Exploration . . . 94
4.3 The Gap Navigation Tree . . . 96
4.4 A Probabilistic Gap Sensor . . . 103
4.5 Gap Detection . . . 104
4.6 Gap Tracking Metric . . . 106
4.7 Data Association . . . 111
4.8 Results . . . 112
4.8.1 Performance of the Gap Motion Model . . . 112
4.8.2 Coverage exploration with the GNT . . . 116
4.9 Chapter Summary . . . 127

5 Probabilistic Cost Maps - Enabling Planning with Uncertainty 129
5.1 Motivation . . . 129
5.2 A Probabilistic Planning Framework . . . 132
5.3 Required Tools . . . 135
5.3.1 Why Gaussian Processes? . . . 135
5.3.1.1 Why Use Gaussian Processes for Terrain Classification? 135
5.3.1.2 Why Use Gaussian Processes for Terrain Costing? . . . 138
5.4 Gaussian Process Modeling . . . 139
5.4.1 Gaussian Process Regression . . . 143

5.4.2 Gaussian Process Classification . . . 145
5.4.3 Training the GP . . . 149
5.4.4 Computational Complexity and Sparse Methods . . . 150
5.5 Results . . . 152
5.5.1 Responding to Variable Costs Within One Terrain Type . . . 153
5.5.1.1 Synthetic Dataset I . . . 153
5.5.1.2 The Tempe Dataset . . . 158
5.5.2 Generating a Distribution of Paths . . . 163
5.5.2.1 Synthetic Data Set II . . . 163
5.5.2.2 Tempe Data Set . . . 167
5.5.2.3 Lisbon Data Set . . . 169
5.5.2.4 Whyalla Data Set . . . 172
5.5.2.5 Classifier Performance . . . 175
5.6 Chapter Summary . . . 175

6 Risky Heuristics - Efficient Path Planning on Probabilistic Costmaps 178
6.1 Motivation . . . 179
6.2 The Heuristic Tradeoff . . . 182
6.3 Precomputing a Probabilistic Heuristic . . . 185
6.4 Proof of Tradeoff Risks . . . 188
6.5 Results . . . 193
6.5.1 Obtaining an approximation to the true heuristic . . . 193
6.5.2 R^*_δ Search Results with Initial Heuristic Approximation . . . 196
6.5.2.1 Suboptimal Termination Risk Functional . . . 197
6.5.2.2 Expected Risk Functional . . . 199
6.5.3 Learning the Scaling Parameters . . . 202
6.5.4 Performance of the ST1 Heuristic . . . 205
6.5.5 Ten Costmap Dataset . . . 211
6.6 Chapter Summary . . . 212

7 Conclusions 215
7.1 Summary . . . 215
7.2 Discussion and Conclusions . . . 218
7.3 Future Work . . . 219
7.3.1 Probabilistic Cost Maps . . . 220
7.3.2 Risky Heuristics . . . 221
7.3.3 Exploration . . . 223

7.4 Concluding Remark 223

A Optimized D* **224**

B Hardware Systems **226**

C Random Costmaps **229**

Bibliography **233**

Chapter 1

Introduction

1.1 Motivation

Increasingly, robots are being used to perform useful tasks in areas such as undersea mapping, planetary exploration, mining, and search and rescue operations, where for economic or safety reasons it is not possible to send humans. Often very little is known about the geometry or topology of the environment in which the robot is being deployed and generally access to a reliable means of localization such as Global Positioning Systems (GPS) is not possible. Hence the environment is largely unknown. However, before we can even think about using robots to perform useful work in these situations, we must first arm them with the capability of being able to explore and map their environment, of being able to move from point A to point B in that map, and with the knowledge of what to do when they encounter previously unseen obstacles. This thesis is primarily concerned with algorithms which enable a robot to efficiently and autonomously plan paths in unknown environments.

In this thesis the closely-coupled problems of both robotic exploration (deciding where to go next) and path planning (deciding how to get from A to B) are examined.

These are both fundamental problems in robotics. Path planning has been a core area of research for several decades and the field is now at the point where robots can successfully navigate from A to B on Mars. However, 'probabilistic planning' - wherein

the planner takes into full account the uncertainty of the robot's sensors and actuators and the subsequent potential for error in both its own location and its identification of the type of terrain surrounding it - is a field still in its infancy. In contrast, approaches to exploration tend to depend on the application (cleaning, reconnaissance, planetary exploration) and the number of robots in use and a convergent approach to this problem is yet to arise.

This thesis aims to show how a single robot operating in an unknown environment could go about exploring that environment using an intuitive, human-like exploration strategy supported by a path planning framework that is robust to the uncertainty in the robot's knowledge of the environment. The framework should allow the robot to adapt to any environment, be it indoor, outdoor, or on another planet.

1.2 Outline of the Problem

Concretely this thesis addresses two separate but closely coupled problems. The first is, given an unknown environment, how do we go about exploring it in order to come up with a reasonably complete map of it? How do we decide what is interesting? How do we know when exploration is complete? How do we break the task up such that we feed the robot's on-board path planner a series of goals $A \rightarrow B \rightarrow C \rightarrow \ldots \rightarrow Z$ so that these subgoals lead us to fully explore the environment?

Here we present an extension to an existing exploration approach that uses the intuitive approach of looking for 'gaps' in the robot's field of view and explores by 'chasing' these gaps - much like a human would explore an unknown environment by looking for doorways, the end of walls or hedges or gaps between trees which we recognize as obscuring some part of the environment. To do this, we must have some way of identifying gaps and then tracking how their appearance changes as the robot moves.

Once we have an exploration strategy we are still faced with getting the robot from its current location to an interim goal (e.g. a gap corresponding to a corner) on the way to fully exploring the environment. Which brings us to the second problem, how do we go about incorporating the uncertainty in the robot's knowledge of the environment into planning paths from point A to point B? This is a difficult problem as it involves a speed/accuracy trade off - accounting for uncertainty requires evaluating all possible outcomes which is costly. On the other hand, discarding unlikely but possible outcomes means that the solution so discerned is liable to be incorrect.

The dominant algorithms for path planning in field robotics currently are graph based algorithms which base their solution paths on the most likely representation of the environment. These path planning algorithms require a simplified *costmap* of the robot's surrounding environment to operate on. Traditionally the way to approach constructing these costmaps is to divide the environment up into a grid of cells and to classify each part of the environment corresponding to one grid cell as being one (and only one) type of terrain. You then associate a single scalar cost proportional to the cost of traversing that terrain type to the cell. Each cell is then associated with a node in a graph and the graph is searched by the D* algorithm to find a solution path.

We would like to build a system that retains the speed of the graph based planners but incorporates knowledge of the uncertainty into the planning task. This will involve learning a probability distribution rather than a scalar cost for each grid cell and incorporating that into the grid based planner.

1.3 Assumptions

In approaching this problem, a number of assumptions were made about the nature of the problem and the robot's capabilities. The robot will be equipped with sensors

which may include laser scanners and cameras which allow it to sense and map the environment as it moves through it. The problem we seek to solve is to explore and navigate outdoors; and although we present techniques in Chapter 4 based on indoor environments we foresee eventual adaptation to outdoor environments. We assume the availability of a localization capability; through either a Global Positioning System (GPS) or a Simultaneous Localization and Mapping (SLAM) system to enable true robot autonomy capable of functioning in areas (such as on the surface of Mars, or underwater) where GPS does not function.

1.4 Contributions

The main contributions of this thesis are as follows:

- an adaptation of the *Gap Navigation Tree* algorithm for use on a real robot in exploring and mapping its environment. An implementation of a robust gap sensor is provided using a laser scanner and metric mapping system.
- a framework for generating probabilistic costmaps using aerial imagery and machine learning techniques,
- a means of speeding up path planning using the above-mentioned cost maps using risk-based precomputed heuristics,
- implementations of the existing A*, D* and Field D* algorithms compatible with the Mission Oriented Operating System (MOOS) robot operating system used by multiple research institutions across the globe (Newman, 2010).

1.5 Structure of the Thesis

Our goal is to equip a robot with the ability to operate autonomously in an outdoor environment. To be able to do this requires three key competencies:

1. An exploration strategy (What is interesting enough to be B?),
2. A path planning policy (How do I get from A to B?),
3. A navigation competency (This is A, where am I in the map?).

These three competencies are strongly interlinked. It is impossible to contemplate exploring in a meaningful way without knowing where you are and without a reliable method of navigating towards a goal in unknown territory. It seems natural then that a thesis addressing the problem of autonomous exploration requires some treatment of all three topics. In this work we have chosen to focus on techniques to accomplish exploration and path planning whilst building on established competencies in navigation.

We begin in Chapter 2 by providing background to the planning task and an overview of the existing approaches to tacking this problem. We also introduce costmaps, critical to the grid-based planning problems that are the focus of later chapters. We round off this chapter with a discussion of the dominant approaches to field robot path planning (A*, D* and Field D*) and describe how they work. It is these algorithms that we have chosen to extend and augment in later chapters.

In Chapter 3 we move on to describe the implementation of these algorithms on the robots operated by the Oxford University Mobile Robotics Group (MRG) in conjunction with the MOOS operating system and the suite of systems developed internally in MRG for robot operation and control. We also provide an overview of the navigation systems utilized in the thesis.

With a grounding in path planning and navigation provided, in Chapter 4 we shift the focus to the exploration problem. We begin with a brief background treatment

of the problem, discussing existing exploration techniques; many of which build on specific planning or navigation competencies we have discussed in Chapters 2 and 3. The heart of this chapter details the adaptation of an existing topological navigational algorithm, the Gap Navigation Tree, for use on a real robot in exploring and mapping its environment. The original algorithm (Tovar et al., 2007) worked by identifying 'gaps' corresponding to occluded portions of the environment, and constructed a topological map of the environment by relating how the various gaps split and merged in relation to each other. In reality, these 'gaps' are difficult to identify using available sensors and this chapter details work on implementing a robust gap sensor using a laser scanner and metric mapping system to enable the algorithm to run on a real robot.

Having presented a method of *deciding B*, in Chapters 5 and 6 we return to the problem of *getting from A to B*. Here we present an approach to probabilistic path planning. We focus on augmenting the D* family of algorithms to handle uncertainty. Chapter 5 presents an alternative method of constructing costmaps, using overhead images or aerial imagery. The technique produces 'probabilistic' costmaps which take into account the fact that we do not always know what type of terrain is actually represented in the cell; or indeed how much that type of terrain should cost; and which allow the cell's cost to be a probability distribution over the terrain type.

In Chapter 6 it is shown that the probabilistic nature of these costmaps can be used to 'speed up' path planning by providing guaranteed bounds on the sub optimality of paths produced when we trade the accuracy of the result for the speed in finding a potential path. The D* algorithm requires a heuristic; the heuristic is used as the graph is expanded to guess how far a particular node under examination is away from the goal. The more accurate this heuristic is, the less time we spend searching. We have combined recent techniques which precompute these heuristics with our probabilistic costmaps so that we can now trade off computation time against

a known risk that the path returned will be less than optimal.

1.6 Publications

The work in Chapter 4 was published in the proceedings of the International Conference on Robotics and Automation (ICRA) in 2008 (Murphy and Newman, 2008) with subsequent presentation at the workshop for Topology and Minimalism in Robotics at the Robotics: Science and Systems (RSS) Conference in 2008. The framework described in Chapter 5 was published in the proceedings of ICRA in 2010 (Murphy and Newman, 2010). Work done as described in Chapter 3 contributed to the publication in (Newman et al., 2009). Work done in Chapter 6 has been accepted for publication at ICRA 2011 (Murphy and Newman, 2011).

Chapter 2

Planning Algorithms - Comparison, Construction and Analysis

We begin our treatise on autonomous robotic exploration in this chapter with an exposition on the critical path planning competency. At the commencement of this thesis, we identified three key competencies we would require a robot to have in order to be able to act autonomously - path planning, navigation and exploration. Of those three it was path planning which eventually demanded the bulk of our attention.

The reason for this is uncomplicated. Ultimately we want to endow the robot with the capability to choose its own goals - the capability to explore. But there is little benefit in equipping the robot with this ability if it cannot first plot a course from its present location to its chosen goal.

In this chapter we provide background to the path planning problem. It is a mature field that has absorbed disparate techniques from robotics, control theory, artificial intelligence and algorithms and is applied to problems as wide ranging as protein folding, drug design, computer graphics, manufacturing and aerospace applications. Here we review only those path planning algorithms relevant to our desired application - a single robot operating in a partially known environment under exploration. Typically these planners require a simplified representation of the world - a cost map - over which to plan. We provide background on typical approaches to

constructing these cost maps.

We conclude the chapter with a detailed examination of the path planning algorithms A*, D* and Field D* which underpin this thesis. We provide an overview of the operation of each of the algorithms and contrast their characteristics and suitable applications. Short examples of how these algorithms work will be provided, and will be illustrated using the output of tools built during the thesis. In Chapter 3 we move to discuss the implementation of these three algorithms as part of the MRG software suite.

2.1 Overview of the Path Planning Problem

Path planning is the task of finding a trajectory through a system model that leads from an initial configuration to a goal state; deciding how to get from A to B. For more than 10 years path planning for field robotics has been dominated by path planning on costmaps and the use of associated grid (graph) based techniques such as the D* algorithm. Path planning for field robots has stood separate from 'traditional motion planning' - the name we give to techniques generally applied to automation, for a number of reasons:

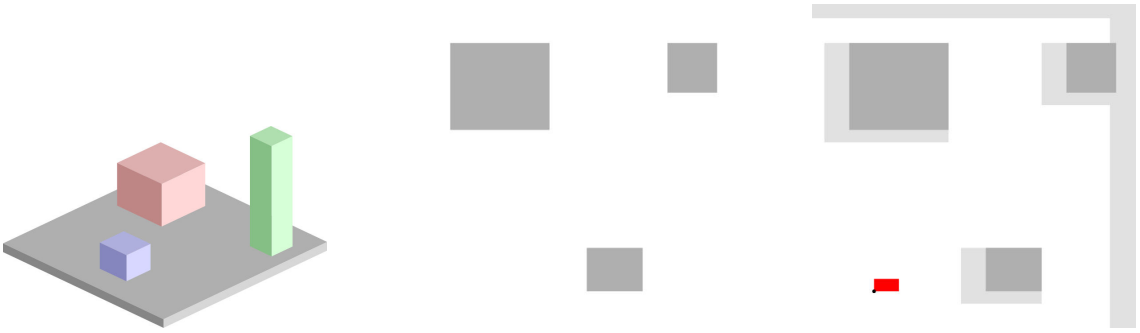
1. Traditional motion planning techniques tend to cater for binary worlds, while in the field we have to deal with a continuum of terrain costs.

2. Traditional motion planning assumes complete knowledge of the world, but this is frequently not the case when operating in the field where much of the terrain is unknown or uncertain.

3. Traditional motion planning tends to focus on high dimensional spaces (such as those required for robotic arms) and the problem is one of finding feasible paths. Mobile robots on the other hand present low dimensional path plan-

ning problems, they require near-optimal solutions and require near real-time planning.

However, recent advances in the field have seen motion planning techniques increasingly applied to path planning on costmaps and we include a brief review of the literature on these techniques here. Path planning within robotics is typically



(a) The Workspace with three obstacles

(b) The Configuration Space of a point robot. The dark grey regions shown here represent obstacles in the workspace.

(c) The Configuration Space of a rectangular robot that can translate and rotate. The additional light grey regions which were not present in the Workspace in (b) denote areas where the robot would touch an object or risk leaving the workspace.

Figure 2.1: The Workspace versus the Configuration Space. The workspace is the same for both robots, but the areas to which each robot can safely move are dictated by the robot's shape.

approached using the configuration state space introduced by Lozano-Perez (1983). The two or three dimensional Euclidean space in which the robot moves is referred to as the *workspace*; its *configuration space* has the same dimension as the number of degrees of freedom of the robot system. Figure 2.1 illustrates this principle. A Configuration Space representation (Figure 2.2) allows the robot to be represented as a single point in the configuration space, C, in which each co-ordinate of the object represents a degree of freedom in the position and/or orientation of the object. The space is partitioned into the space of free configurations C_{free} which the robot may reach without violating its mechanical constraints or colliding with obstacles in the environment, and its complement, the space of obstacle configurations C_{obs}. The aim

of robotic path planning is thus to compute a continuous function $f : [0, 1] \rightarrow C_{free}$ connecting an initial world state $x_{start} = f(0)$ to a goal state $x_{goal} = f(1)$. While the Configuration Space allows every possible configuration of the robot system to be uniquely represented, representations of invalid configurations and obstacles which are clearly defined in the workspace are often difficult to obtain.

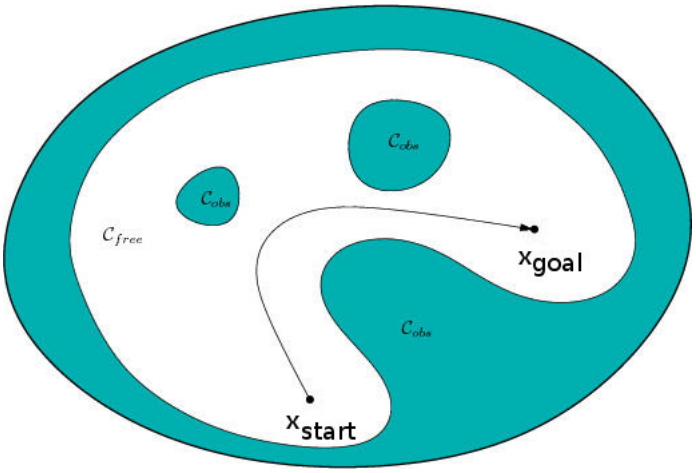
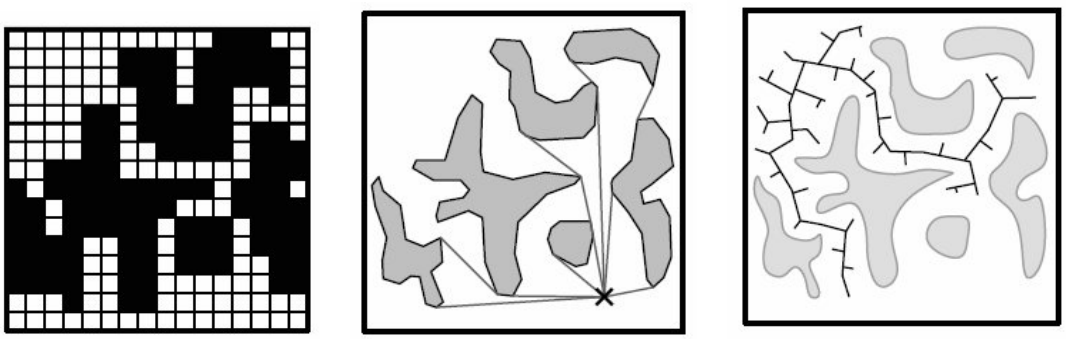


Figure 2.2: The generalized robotic path planning problem shown in a 2D configuration space. We wish to plan a path from q_I to q_G through the free space represented by C_{free}. Obstacles are represented by C_{obs}. Figure from LaValle (2006).

As a result, two distinct philosophies have grown to address the motion planning problem. A simple taxonomy of the literature divides approaches to path planning amongst Combinatorial, Sampling, and Grid-Based techniques (represented in Figure 2.3).



(a) Grid-Based (b) Combinatorial (c) Sampling

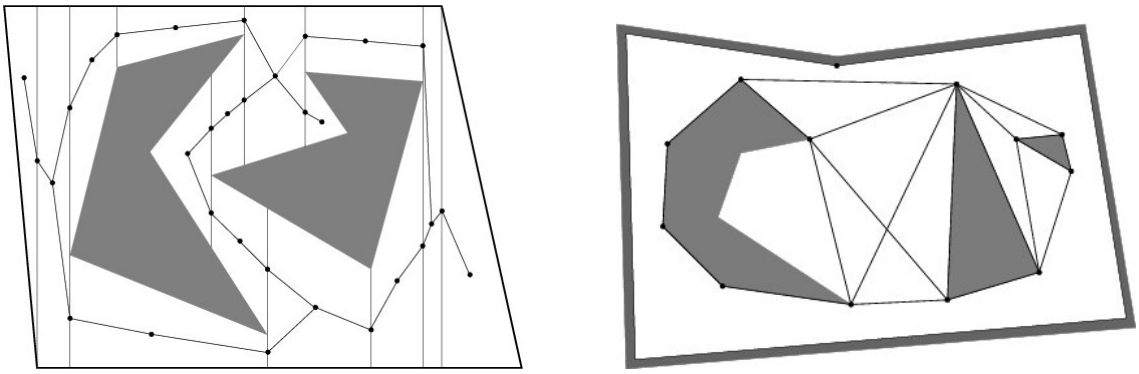
Figure 2.3: Three approaches to path planning, figure taken from Wooden (2006).

2.1.1 Combinatorial Planners

Combinatorial planners represent the oldest branch of planning, and in contrast to the two other planning techniques they are *exact* algorithms. They do not simplify the environment in any way, and use the boundary of polygonal obstacles to plan in the configuration space. This lack of approximation means that the algorithm will either find a solution or report correctly that a solution does not exist. This is in contrast to sampling and grid based methods, where approximate representations may lead to the existence of a path being masked.

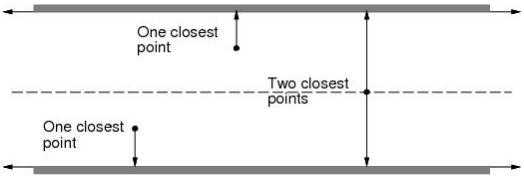
Combinatorial Motion Planners work by constructing a *roadmap* in the configuration space. Roadmaps preserve the connectivity of the regions of the original graph and can be used to solve queries (x_start, x_goal) by connecting the start and goal nodes to the roadmap and then performing a graph search over the roadmap. In combinatorial planning, roadmaps take one of three forms. Cell decomposition roadmaps break the environment up into a number of distinct cells. One example of this type is the vertical cell decomposition (Chazelle, 1987) which may be obtained by a left-to-right ordering of the vertices of all polygonal obstacles in the workspace and then doing a plane sweep from left to right, adding a vertical line to the graph whenever a vertex is encountered. A single *sample point* (e.g. the centroid of the cell) is chosen to represent each cell and the roadmap is drawn by connecting the sample point to those of neighbouring cells. A roadmap and decomposition resulting from this technique is shown in Figure 2.4(a).

Maximum Clearance roadmaps (Canny and Lin, 1993; O'Dunlaing and Yap, 1982) are useful in some mobile robot applications as they minimize the risk of collisions. These roadmaps are constructed by drawing maximum clearance lines between every vertex-vertex and edge-edge pair in the graph as well as parabolas (which represent the shortest distance between a point and a line) between every vertex-edge pair. The maximum clearance roadmap is obtained from the intersection of these curves.



(a) A cell decomposition roadmap

(b) A shortest path roadmap



(c) A maximum clearance roadmap

Figure 2.4: Combinatorial Motion Planners. Images courtesy (LaValle, 2006)

Shortest Path Roadmaps (also called Reduced Visibility Graphs) (Nilsson, 1969; Latombe, 1991) by contrast tend to graze the corners of the obstacles in the map. A reflex vertex is defined as a polygon vertex whose interior angle (that lies in the free space) is greater than 180°. Edges in the shortest path roadmap are constructed in one of two ways. Firstly, all reflex vertices that form edges of the polygonal obstacles and the associated edges are added to the roadmap. Secondly, if a (bitangent) line can be drawn between two reflex vertices without cutting through an obstacle then it too is added to the roadmap. The resulting roadmap can be used to find paths between any given vertices q_a and q_b by connecting the vertex to the roadmap and running Dijkstra's search algorithm (Dijkstra, 1959), where each edge in the roadmap is given a cost proportional to its length.

While many combinatorial motion planners have been designed and implemented successfully, they have recently been superseded as a result of the rise of fast, approximate solutions offered by sampling based planners (in high dimensional spaces) and grid based planners (for field robotics). Combinatorial planners also do not extrapo-

late well to more than 2 dimensions, and their basis in computational geometry means that they are not suitable for use when the environment is only partially known or uncertain. They also bear little relevance to path planning on mobile robots as often the configuration space is unknown and/or likely to change.

2.1.2 Sampling-based planners

Sampling-based techniques also hail from the motion planning domain and are typically applied to high-dimensional problems in the fields of manufacturing, computational biology or robotic arm manipulation, where computing the configuration space is infeasible (Barraquand and Latombe, 1991). The sampling approach arose in the mid-1990s with techniques such as Randomized Path Planning (Barraquand and Latombe, 1991), Ariadne's Clew (Mazer et al., 1998), Probabilistic Roadmap Planners (Kavraki et al., 1996), and Rapidly Exploring Random Trees (LaValle, 1998); and uses sampling of the configuration space in conjunction with a collision detector to build an approximation of the space of obstacle configurations, C_{obs}. Completeness is sacrificed in favour of probabilistic completeness, where if a solution exists the probability of finding it converges to 1 as the computation time tends to infinity. Here we discuss two important and successful approaches which implement this approach to path planning, Probabilistic Roadmap Planners and Rapidly Exploring Random Trees.

2.1.2.1 Probabilistic Roadmap Planners

Probabilistic Roadmap Planners (PRMs) (Kavraki et al., 1996) work in two stages to construct a path from start to goal. In the initial *learning stage* the algorithm samples the configuration space according to some sampling measure π and builds an undirected graph $G = (V, E)$ of samples in free space C_{free}, which constitutes the Roadmap. Edges in the graph are built by a local planner, which attempts to

connect pairs of useful samples usually with a simple, fast strategy such as a straight line interpolation. The local planner is allowed to fail in difficult cases, but if it can successfully connect two samples and the path is collision free, the edge E is added to the graph. In the second *Query stage* the graph is used to solve specific Roadmap planning instances whereby it is attempted to connect start and end nodes to the existing graph in the same manner as nodes are added to the roadmap in the learning stage. Once these nodes are connected to the graph path planning is reduced to a graph search. The learning stage of the PRM technique takes significantly longer than the query stage, meaning this approach is better suited to solving multiple instances of the path planning problem in the same workspace. In its standard form, the PRM is unsuited to dynamic environments as when objects move the roadmap is invalidated.

In Hsu et al. (2006) it is explained that the probabilistic nature of this technique stems from the fact that the planner could conceivably maintain a representation (H, η), where H is the set of all hypotheses over the shape of the free space F and η is a probability measure that assigns to each hypothesis in H the probability of it being correct. An optimal sampling measure $\hat{\pi}$ would be the one that locates samples in the environment such that the number of remaining iterations to connect the start node to the goal node was minimized. Conceivably, $\hat{\pi}$ could be inferred from (H, η) but actually maintaining this representation over continuous space would be impossible. Instead, most implementations of PRMs approximate $\hat{\pi}$ using heuristics which use knowledge of the shape of the free space to increase the sampling density in difficult regions.

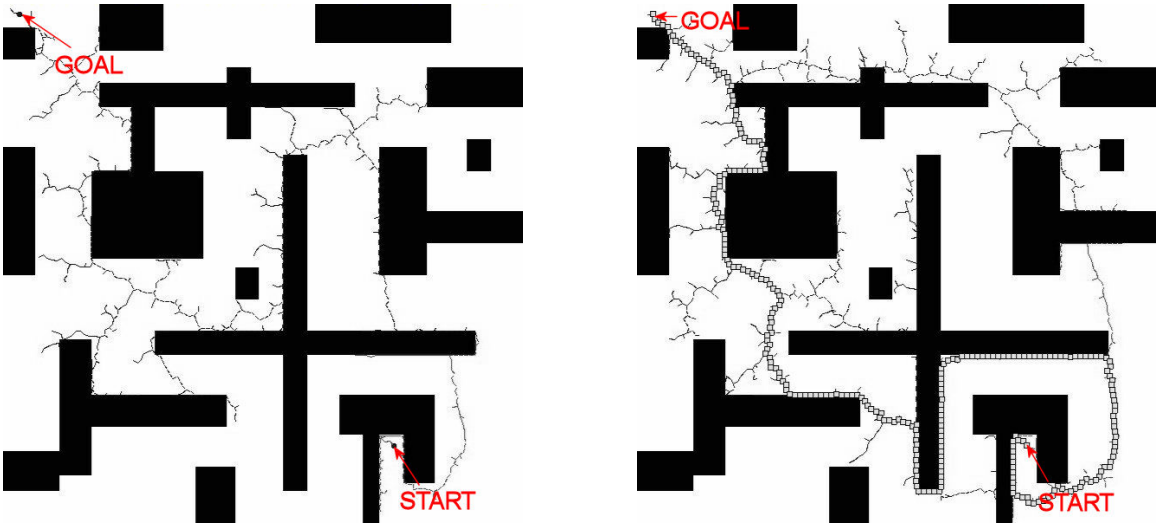
Missiuro and Roy (2006) describe an extension to the PRM algorithm to provide robustness against uncertain maps for a simple 2-DOF mobile robot. A feature-based map models the uncertainty of obstacles in the free space and the sampling strategy is adapted to account for the map uncertainty by accepting or rejecting samples according to a stochastic threshold. A cost of collision is computed probabilistically

and associated with each edge in the graph and a minimum cost planner is used to compute valid paths. Burns and Brock (2007) use a function which combines utility and cost, weighted by the probability that the path is successful, to select paths through an unknown environment that minimize uncertainty and to direct further sensing if required. Despite adding robustness against uncertainty, these algorithms still operate in a world where terrain costs are binary, and hence in present form are unsuited planning paths in outdoor environments where we have a continuity of terrain cost.

2.1.2.2 Rapidly Exploring Random Trees

Unlike PRMs which first sample the entire configuration space before connecting a graph, Rapidly Exploring Random Trees (RRTs) (LaValle, 1998; Lavalle and Kuffner, 2000) grow a search outward from a single point. A tree of reachable configurations is maintained from this start node. During the *expansion phase* a random configuration from the configuration space q_{target} is selected. The closest node (typically measured by Euclidean distance) on the tree, $q_{nearest}$, is calculated and it is attempted to grow the tree some distance d towards the target by placing a new node q_{node} in the direction of q_{target} from $q_{nearest}$. A collision detection function is used to check if the new node satisfies global constraints, and if extending towards q_{target} requires passing through an obstacle, no extension occurs. This process is repeated until the tree reaches a threshold distance within reach of the goal location.

The RRT-Connect algorithm (Kuffner and LaValle, 2000) incrementally builds two random trees rooted at the start and goal configurations. The trees grow towards each other as the last branch added to one tree becomes q_target for the next expansion step of the other tree. A number of works build on this to apply RRTs in a field robotics context. In (Ettlin and Bleuler, 2006a,b) the tree is extended by measuring the 'obstacleness' of candidate nodes and comparing this against a traversability



(a) RRT Construction (b) Path finding with RRT

Figure 2.5: The construction of an RRT through an environment with obstacles. The path is highlighted in 2.5(b). Note that Planning is done in the C_{space} but for clarity the tree construction is shown in R^2. Note also how the tree construction grows close to obstacles in the environment due to the unsophisticated sampling strategy employed.

threshold that biases the search towards easily navigable terrain. This threshold is revised upwards until a path can be found. Urmson and Simmons (2003) introduced a quality measure to incorporate the cost of the path into the RRT algorithm with the goal of applying RRTs to navigation in difficult outdoor terrain. Similarly Ferguson and Stentz (2006a) combines RRTs with Anytime algorithms in a non-uniform cost environment to produce a system that generates a succession of RRTs in response to an initial query. Each tree reuses the results of the previous search to produce increasingly better (cheaper) paths.

The downside of these randomized (and sampling) methods is that they still do not guarantee any properties of the resultant path beyond that they do not intersect any obstacle, and/or that they are the product of a strategy to bias path construction towards low cost regions. Variants of this family of algorithm tend to produce complex, jagged paths with many unnecessary segments that still require some form of post processing before they can be fed to the robot's motion control module. For this reason, planning techniques based on grid maps still dominate the mobile robot

path planning literature.

2.1.3 Grid-based costmap planners

In grid-based path planning a regular grid is imposed on the environment over which we wish to plan. Some cost, proportional to the traversability of that particular part of the environment, is attributed to each cell in the map. The goal is to find the minimum cost path from the robot's current location to the goal location, both of which are located at the corners of cells. Typical approaches to solving this kind of single pair shortest path problem then replace the grid with a graph. The nodes of the graph are placed either in the center or at the corners of the grid cells. Commonly, each node is connected to eight neighbours which places graph edges only between nodes from adjacent cells. This method is often favoured as the number of edges in the graph is linear in the number of grid cells; so planning is fast, although the resulting paths restrict movements to increments of 45 degrees. Naive approaches which do not restrict the robot's possible headings require connections between all node pairs, meaning that graphs have edges which are quadratic in the number of grid cells and are significantly slower to plan over.

When planning outdoors we are often confronted with the situation in which there is some uncertainty as to what terrain type a particular cell represents. Figure 2.6 shows the type of distribution over terrain that might arise for a number of reasons - we do not have perfect information about the cell, the classifier we use is uncertain, or the resolution of the cell is such that it represents a mixture of terrain types.

There are two approaches to dealing with a distribution rather than a fixed (scalar) value for cell cost. The first, *Assumptive Planning* (Nourbakhsh and Genesereth, 1996) requires making a principled simplifying assumption about the terrain represented, ideally an assumption that will preserve the robot's ability to reach the goal even if the assumption is wrong. Assumptive Planning techniques make planning

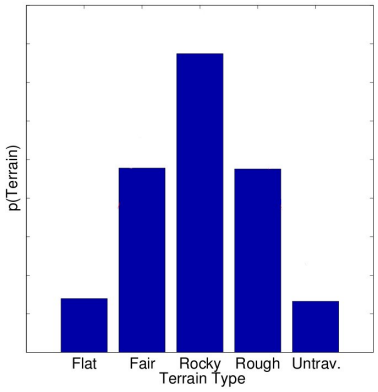


Figure 2.6: An example of the form a probability distribution over terrain type might take for a particular grid cell.

tractable by representing the terrain distribution with an acceptable single case; such that a cell which is most probably *scrubland* but also exhibits some probability that it might be *grassland* is chosen to be represented as *scrubland*. The alternative option *Partially Observable Planning* generates every possible terrain type and its associated probability for each cell. We call it 'Partially Observable' as the terrain distribution cannot be directly observed, but we have access to measurements which give some clue as to the true terrain distribution. By exhaustively expanding the possible terrain distributions for each cell, it is possible to compute the true expected cost. Computing these costs is exponential in the number of cells in the environment and although Partially Observable techniques have long been considered intractable, simplifications have meant that they are gaining some foothold in the field robotics fraternity (Du et al., 2010; Hsiao et al., 2007; Kurniawati and Lee, 2008; Pineau et al., 2003; Smith and Simmons, 2005). Although in this thesis we will ultimately examine probabilistic planning from within an assumptive planning framework, a survey of the current state of the field would not be complete without some discussion of the leading approach to Partially Observable Planning - Partially Observable Markov Decision Processes (POMDPs), and this will be done in Section 2.1.3.2.

2.1.3.1 Assumptive Techniques

Path planning problems in grid-based form can be solved with traditional graph search methods. Dijkstra's algorithm (Dijkstra, 1959) solves the single source shortest path problem for a graph with non-negative edge costs, finding the lowest cost path between the source node and every other node in the graph. It begins the search at the start node and expands all neighbours of the start node, calculating their distance from the start. Once this operation is complete, it then examines the neighbours of the unvisited node with the smallest distance from the initial node. This process is repeated until all nodes in the graph have been visited. Dijkstra's run time is hence slow as it requires visiting every node in the graph.

The A* search (Hart et al., 1968) cuts down the portion of the graph that must be explored through the use of a *heuristic* that focuses the search towards the goal. It expands nodes according to a priority given by a distance-plus-cost heuristic function $f(s) = g(s) + h(s)$ where $g(s)$ is the 'start distance' of the node as calculated in Dijkstra's search and $h(s)$ is a heuristic estimate of the distance of the node s under consideration to the goal node. Although it is impossible to know the exact cost to the goal in advance, good heuristics can be constructed using the euclidean distance or a Manhattan distance to the goal. These often provide reasonable under-estimates of the goal distance - the path length can only increase once obstacles are factored in. If $h(s)$ never overestimates the cost to the goal then the heuristic is *admissible* and A* is guaranteed to return an optimal solution to the path search. Under this condition A* will consider fewer nodes than any other admissible search algorithm using that same heuristic (Dechter and Pearl, 1985). A* is discussed in depth in Section 2.3.

A number of techniques tailored to robotics applications build on A* to reuse prior knowledge gained in previous search iterations. They incrementally repair paths should obstacles move or costs change triggering a change in the underlying graph; and find solutions much faster than would be possible by solving the search task from

scratch. In Lifelong Planning A* (Koenig et al., 2004) the first search is the same as A*. However LPA* maintains a second estimate of the start distance in addition to g(s). This one step look ahead (the $rhs(s)$-value) is the minimum of the $g(s)$-values of all the cell's neighbours plus the cost of moving from the neighbour to the cell under consideration. If $g(s)$ is the same as $rhs(s)$ the cell is 'locally consistent', but it is the 'locally inconsistent' cells that provide a starting point for replanning after a change in the graph. This leads to an algorithm that updates only those $g(s)$-values for cells that are relevant for computing a shortest path, rather than all cells in the case of replanning A* from scratch. It produces good results if the problem domain changes only slightly and the changes are located close to the goal.

Whereas an A* search plans from the start to the goal, D* (Stentz, 1993, 1995) plans backwards from the goal node. The name D* stems from Dynamic A*, as the algorithm behaves similarly to A* except when path costs change. D* lite (Koenig, 2002) implements the same behaviour as Stentz's algorithm but is algorithmically different. Simpler to implement (Koenig and Likhachev, 2002b), it forms the basis for most fielded D* systems (Wooden, 2006). An in-depth discussion of the operation of D* is provided in Section 2.4.

These early incremental replanners restrict transitions from each node to a four or eight-connected grid of neighbouring cells, so that heading changes are restricted to increments of $\frac{\pi}{4}$ radians or $\frac{\pi}{8}$ radians. Two main algorithms have been developed to perform any-angle planning on a cell decomposition of an environment. Theta* (Nash et al., 2007) and its incremental variant (Nash et al., 2009) do this by searching for line-of-sight shortcuts when updating the g-value and parent of previously unexpanded vertices during their initial expansion. Field D* (Ferguson and Stentz, 2005b, 2006b) interpolates along edges during vertex expansion and has been successfully deployed on the Mars Rover vehicles (Carsten et al., 2009). It is discussed in depth in Section 2.5.

A number of approaches have arisen to speed up the run-time of grid-based algorithms. They attack the problem from one of two ways. First is to reduce the number of nodes in the grid. Hierarchical Pathfinding A* (Botea et al., 2004) preprocesses the grid to build higher-level graphs (possibly on multiple levels) and plans paths recursively down through the levels. Alternatively quadtrees model large constant-cost regions of the environment as single cells and can be used to represent the environment using a non-uniform resolution grid (Samet, 1982), albeit one which suffers from suboptimal solutions due to the restriction that paths must transition through the centres or corners of cells. Framed quadtrees (Chen et al., 1995) add a region of cells of the highest resolution around each cell to produce better results at the cost of greater computational complexity. In Ferguson and Stentz (2006c), the edge interpolation method of Field D* is applied to nodes with varying numbers of neighbouring edges of variable lengths.

The second method is to influence the heuristic employed by A* and its variants to ensure fewer nodes outside the optimal path are considered in the search. In the ALT (so named because of its use of A^*, Landmark and the Triangle inequality) algorithm (Goldberg and Harrelson, 2005; Goldberg, 2007) a small set of vertices are selected as landmarks and the distances to the landmarks are computed from every other node in the graph. Then, when it comes time to compute the heuristic value to guide the A^* search, we use the triangle inequality to produce a tight lower bound on the cost to goal. Consider the landmark L in Figure 2.7, if $d(\cdot)$ defines the distance *to* L, then by the triangle inequality:

$$d(u) - d(v) \leq dist(u, v). \tag{2.1}$$

Otherwise if $d(\cdot)$ defines the distance *from* L, we have

$$d(v) - d(u) \leq dist(u, v). \tag{2.2}$$

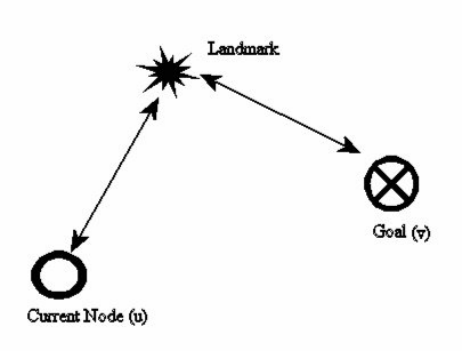


Figure 2.7: A*, Landmark and Triangle (ALT) in action

The maximal lower bound over all landmarks is used to select the heuristic which guides the A^* search.

The LPI algorithm (Grant and Mould, 2008) also uses landmarks to precompute heuristics. It differs from ALT in that it does not compute an exact path - paths found using LPI are restricted to follow shortest paths stored between landmarks. HTAP (Mould and Horsch, 2004) works by precomputing a hierarchy of abstracted graphs, at each level of the hierarchy a path is found between start and goal and this is used to constrain subsequent higher-resolution searches. The paths produced by these two techniques cannot be guaranteed to be globally optimal.

2.1.3.2 Partially Observable Planning

Partially Observable Markov Decision Processes (POMDPs) (Smallwood and Sondik, 1973; Kaelbling et al., 1998) provide a principled framework for dealing with uncertainty. They take into account that the robot's sensors and actuators are imperfect and noisy, meaning that observations of the robot's current state and the effect of actions to change that state are stochastic in nature. If the robot has imperfect knowledge of its state, then in choosing the next action to perform it must take into account all possible states that are consistent with its observations. POMDPs model the robot state as a *belief*, a probability distribution over all robot states. Where classical planners produce a sequence of actions, POMDPs produce a universal *policy*

for action selection.

Formally, a POMDP is a tuple $(S, A, O, T, Z, R, \gamma)$ where S is a set of states, A a set of actions and O a set of observations. A conditional probability function $T(s, a, s') = p(s'|s, a)$ models the transition from state $s \in S$ to a new state s' after taking some action $a \in A$. The Markov property dictates that s' only depends on the previous state s and the action a. The observation model $Z(s, a, o) = p(o|s, a)$ is the probability that the robot sees observation o in state s after taking action a.

The reward function $R(s, a)$ is used to describe the desired behaviour of the robot. It is a real-valued function proportional to the utility of taking action a while in state s. At each time step, the robot takes some action and receives a reward. Problems that have a goal state can be represented in this framework by assigning the goal state a high reward and all other states zero. Alternatively, the reward function can penalize certain conditions along the way and/or represent multiple goal regions. The individual rewards r_τ are multiplied by a discount factor γ^τ which dictates how much the robot should devalue a reward one time step into the future. For an infinite-horizon POMDP the discount factor $\gamma \in [0, 1)$ makes the problem tractable by ensuring the total reward is finite.

The goal of the POMDP solution is to compute a policy $\pi(b) \rightarrow a$ for selecting actions based on the current belief. Of greatest interest is the *optimal policy* that maximizes the expected total sum of rewards $E[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)]$ over the *horizon* - the number of time steps we wish to evaluate the problem over.

To reiterate, the key assumption of a POMDP is that the robot's state is only partially observable. So in order to choose an optimal policy the robot requires some form of memory of its history of its actions and observations. A sufficient statistic that accomplishes this is the *belief state* b:

$$b_t = P(s_t|b_0, a_0, o_1, ..., o_{t-1}, a_{t-1}, o_t) \tag{2.3}$$

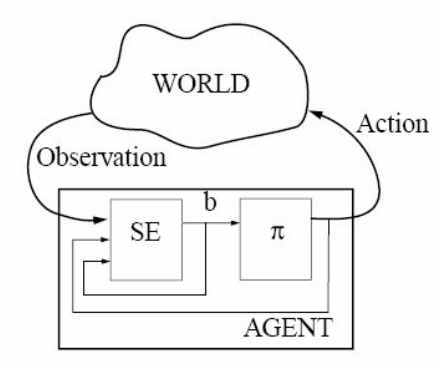


Figure 2.8: POMDP control loop, observing, estimating the state (denoted by box SE), using this to evaluate policy π and choose an appropriate action.

which represents the probability distribution over states S as an $|S|$-dimensional vector whose elements $b_t(s_i)$ specify the robot's belief that it is in state s_i.

The belief is updated via Bayes rule at each time step to incorporate the latest action and observation:

$$b_t(s') = P(s'|a_t, o_t, b_{t-1}) = \frac{Z(s', a_t, o_t) \sum_s T(s, a_t, s') b_{t-1}(s)}{P(o_t|a_t, b_{t-1})}. \tag{2.4}$$

We refer to equation 2.4 as the state estimator $SE(b, a, o)$. In creating the belief state we have replaced the unobservable state with an observable *information state* and the POMDP is now a special kind of Markov Decision Process (MDP). The optimal policy we are searching for is the solution of a continuous-space 'belief MDP' (Kaelbling et al., 1998), and the transition and reward functions can be transformed to yield Bellman's equation for belief MDPs (Bellman, 1957):

$$V^*(b) = max_a[\sum_s b(s)R(s, a) + \gamma \sum_o P(o|a, b)V^*(SE(b, a, o))]. \tag{2.5}$$

An optimal policy greedily maximizes equation 2.5.

This optimal value function of equation 2.5 can be approximated by a piecewise-linear convex function

$$V(b) = max_{\alpha \in \Gamma}(\alpha \cdot b) \tag{2.6}$$

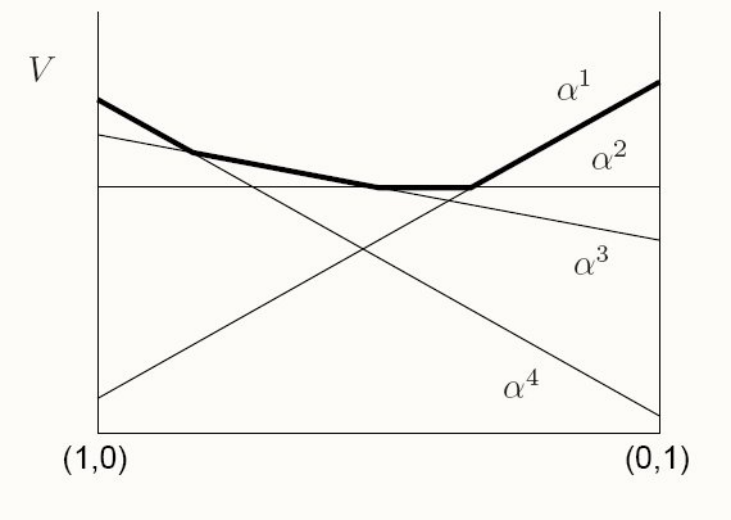


Figure 2.9: A sample value function over the belief space of a POMDP. The vertical axis is the value, while the horizontal axis is the belief state. The POMDP value function is the upper surface of a finite number of linear segments $\{\alpha^1 \ldots \alpha^4\}$, each of which propagates the history of action choices over the horizon so far.

where Γ is a finite set of vectors, each vector associated with a particular action. After n consecutive iterations the solution is represented by a set of α-vectors $V_n = \alpha_0, \alpha_1, ...\alpha_n$ each describing the value function over a bounded region of the belief $V_n(b) = max_{\alpha \in V_n} \sum_{s \in S} \alpha(s)b(s)$. The problem with this *exact update* is that at each new planning horizon we increase the size of this solution vector set exponentially in the number of observations. Couple this with a belief space which has to have the same dimensionality as the robot's state space, and finding exact solutions for POMDPs very quickly becomes intractable for more than a few states (robot motions) of the typical robotics path planning problem; this involves the robot travelling in the order of tens of metres where each grid cell (state) is of the order of 30cm resolution.

Practical POMDP applications require approximations to the solution. Of course, not all belief states are necessarily relevant, and as Figure 2.9 shows, not all of the vectors which propagate the history of the action choices are relevant to the solution either. Point based POMDP algorithms reduce the dimensionality of the belief space by sampling it probabilistically and then computing approximate solutions concentrating only on those parts of the belief space that have a high probability of being

encountered. Algorithms such as PBVI (Pineau et al., 2003), SARSOP (Kurniawati and Lee, 2008) and HSV12 (Smith and Simmons, 2005) can handle problems with up to 100 000 states, and solve problems with hundreds of states in the order of seconds (Du et al., 2010).

There are a number of instances in which point based algorithms have been used in real world applications of POMDPs. In Hsiao et al. (2007), methods of model minimization were used to create an abstract model of the underlying state space and to apply POMDPs to two-fingered grasping in two dimensions (x and z) with successful results in a 408 dimensional state-space. In Pineau et al. (2006), POMDPs are applied to a nursing-assistant robot which must systematically explore an indoor environment and deliver reminders (time to take medication, time for an appointment) to elderly patients. The state has two features, the person and robot location, represented through a discretization of the environment. PBVI is used - with 3000 belief points over the 626 dimensional state space planning takes around 11 hours (this was not an online solution!) and the robot managed to find the person more that 90% of the time. With only 443 belief points the planning time drops to less than 1.5 hours, but the robot fails to find the person in 60% of trials. In Du et al. (2010) the MiGS sampling approximation allows a Quadrocopter to navigate in an indoor environment and reach the goal 88% of the time. Here the environment is represented by 5 levels with 18 x 14 grid cells on each level and planning takes approximately 15 minutes.

2.1.4 Summary of the Planning Literature

Of the techniques we have covered here, few in their current state are suitable for use in our desired application, which involves being able to plan in arbitrary worlds where traversability is represented by a continuum of terrain costs. Combinatorial algorithms are exact and thus too slow; sampling based algorithms have not yet

evolved to the point where they can be used outdoors; and while promising, POMDP methods remain computationally intractable for long term planning and traversal. In the absence of a truly viable alternative, we have chosen to use the A* and D* family of algorithms as the basis for the path planning competency implemented in this thesis. In the next two sections which round out this chapter, we first provide some background to the Cost Maps which this grid-based family requires to operate; before describing their operation in detail beginning in Section 2.3.

2.2 Cost Maps

Reliable navigation and traversal using assumptive techniques is heavily reliant on the ability to sense and classify the surrounding terrain. Typically this information is aggregated in a cost map that is then fed to a classical grid-based planner. Terrain classification has been addressed by many researchers using a variety of sensors (laser range finders, vision sensors, ultrasonic range finders and tactile sensors) to detect features such as colour, range, image texture and vibration. Here we review the state of the art in terrain classification and cost map construction.

When operating a mobile robot indoors the typical cost map representation is a 2D Occupancy Grid (Moravec and Elfes, 1985) (see also Section 3.2.1). Occupancy grids represent the probability of each cell in the map of being occupied by an obstacle, and are often sufficient for planning in indoor environments as the floor is considered uniformally traversable and obstacles represent regions of binary cost. Costmaps for planning algorithms such as D* can be constructed by simply applying a threshold on obstacle belief and assigning one of two costs (i.e. 1 for free space, infinity for obstacle) to the thresholded occupancy grid map. There are alternatives to occupancy grids for outdoor environments which incorporate height information, Moravec (1996) extends occupancy grids to 3D using voxels, while other (2.5D) approaches store

height information in a 2D grid (Bares et al., 1989).

However, when operating outdoors we are often worried about more than bumping into obstacles. Some areas of terrain (e.g. roads) are infinitely preferable to drive over than others (such as water). Various combinations of sensors and classification techniques have been applied to the terrain classification problem. In Thrun et al. (2006), where a 142-mile autonomous traversal was completed along a desert road, lasers are used for short and medium range obstacle avoidance and each 2D location in a surface grid is classified as occupied, drivable or unknown using a probabilistic test on the height distribution of neighbouring points. Beyond the laser range, colour imagery from a camera is used to detect drivable surfaces by mapping the drivable surface detected by the laser into the image and using these points as training data for the concept of a drivable surface in a generative mixture of Gaussians classifier.

In Ferguson and Likhachev (2008), the vehicle that won the DARPA Urban Challenge (DARPA, 2007) fused 2 types of 2D cost maps together to facilitate planning on congested urban roads and in car parks. A *perception map* translates height information from laser range finders into a cost ranging from "*FREE*" to "*LETHAL*". A *constrained map* is used to express the relative desirability of certain parts of the environment in preference to others. Built from a-priori information, it encodes knowledge such as 'staying within the road lane is preferable to travelling in the oncoming traffic lane', and 'that driving off road is dangerous'. It can also incorporate aerial imagery so that features such as trees and curbs are known to the planner before they are detected by on board sensors. The two cost maps are combined in the *combined map* by taking the maximum of corresponding cell values in each of the maps.

The benefit of constructing global plans from overhead data gathered from aerial sources such as satellites and planes has been noted in (Sofman et al., 2006a). Aerial imagery is used to classify terrain using a neural network. Each terrain type is assigned a cost by a human operator. Elevation data and 3D Point Cloud data from

aerial LiDAR [1] flyovers are used to do vehicle mobility analysis, a set of functions then map parameters such as roll, pitch and ground clearance to a traversal cost for the given vehicle type. The terrain classification and mobility analysis data are then summed to produce a final traversal cost.

In Silver et al. (2008), *imitation learning* is used to automate the building of a cost function from overhead data which may be used with different planners. An expert provides examples which serve as a constraint on cost functions and the system learns mappings from raw expert data to costs that reproduce similar behaviour.

In Gerkey and Konolige. (2008); Konolige et al. (2006), stereo vision is used to classify terrain into obstacles; ground plane free space; sight line free space; and path free space for the LAGR robot, capable of outdoor runs of 200 metres. AdaBoost (Freund and Schapire, 1995) is used to learn color models for path and non-path pixels offline, and the model is then turned into an RGB lookup table for fast classification. Online terrain classification is then performed using a table lookup.

2.2.1 Summary

We will return to the topic of costmaps again in both Chapter 3.2.2 and Chapter 5 of this thesis. We included this brief review of the topic here so that the reader has a clear picture of what is constituted by the term 'Costmaps' in the context of the A* search family. We now conclude this introductory chapter with an in-depth look at the operation of the A*, D* and Field D* search algorithms.

2.3 A* Search

As alluded to in Section 2.1.3.1, A* is a graph-based planning algorithm. This means that the first step in applying it to the problem of searching over terrain is to simplify

[1] Light Detecting and Ranging - an optical remote sensing technique that measures properties of scattered light to find the range of a distant target

the search area by dividing it up into a grid. Although the grid is typically square, there is nothing stopping us from imposing a triangular, rectangular or hexagonal grid over the terrain or indeed a multi resolution grid as has been used by Chen et al. (1995), Ferguson and Stentz (2006c) and Samet (1982). Regardless of the grid configuration, we require some way of mapping the grid to a graph-based representation. The simplest way of doing this on a regular grid is to associate a *node* of the graph with the center of each grid cell. This simplified version of the problem thus restricts solution paths to traversing the centre of grid cells. *Costs* are associated with the edges of the graph. Frequently when transiting from cell A to cell B it is the average cost of terrain in cell B that is used as the edge cost.

An *explicit* A* search constructs the entire graph from the grid cells before beginning the search for a path from start to goal, and is therefore very memory-intensive. *Implicit* graph representations construct the graph on-the-fly as the search progresses. They are more useful when exploring large state-spaces (such as those in a encountered on a robotic mission) as adjacent vertices to a vertex are only generated when necessary. This frees us from having to store a large graph in memory.

As illustrated in the pseudocode of Figure 2.10, the A* algorithm employs two lists in controlling the search. The *OPEN* list is like a shopping list of nodes that are candidates for examination. Initially it contains just one node, the start node. The *CLOSED* list is initially empty but will grow to contain those nodes that have already been examined and are of no further interest to the search. The graphic of Figure 2.11 illustrates how the nodes on the open list at any one time represent the frontier of the search space, while the closed list contains those nodes that have been examined fully.

The nodes are added to the *OPEN* list with a crucial piece of information. The f-value of a node determines the order in which it is extracted from the *OPEN* list.

```
A*-Search
 1  OPEN = s_start;
 2  CLOSED = ∅;
 3  g(s_start) = 0;
 4  h(s_start) = h(s_start, s_goal);
 5  f(s_start) = h(s_start);
 6  while OPEN ≠ ∅
 7        do s_current = argmin_f{OPEN}
 8           if s_current = s_goal
 9              then return ReconstructPath(parent(s_goal));
10              else
11                    Remove s_current from OPEN
12                    CLOSED = CLOSED ∪ s_current;
13                    for s' ∈ Succ(s_current)
14                          do if s' ∈ CLOSED
15                                then continue;
16                             if s' ∉ OPEN
17                                then OPEN = OPEN ∪ s';
18                                else  if g(s_current)+cost(s_current, s') ≮ g(s')
19                                         then continue;
20                                      parent(s') = s_current;
21                                      g(s') = g(s_current)+cost(s_current, s');
22                                      h(s') = h(s', s_goal);
23                                      f(s') = g(s') + h(s');

ReconstructPath(s)
24  while s ≠ ∅
25        do Path.PushFront(s);
26           s = Parent(s);
```

Figure 2.10: A* Algorithm

It is the sum

$$f(s) = g(s) + h(s) \tag{2.7}$$

where $g(s)$ is the known cost from the starting point s_{start} to the node s currently being evaluated. Just as in a Dijkstra shortest path search, the g-values represent the best known path calculated so far by summing the known edge costs between the start and the node. In a robotics context, this would mean it is the shortest path validated by the robot over terrain it has already traversed. The h-values represent the heuristic - it is the heuristic that guides the A* search and differentiates A* from

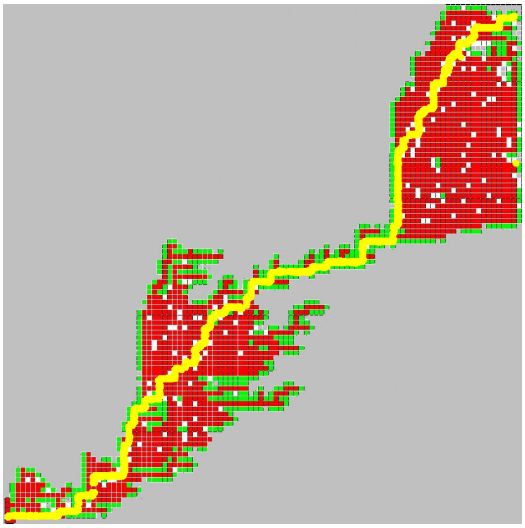


Figure 2.11: Open and Closed lists: The figure illustrates how those nodes on the open list (in green) represent the frontier of the search. The red nodes (on the interior of the search area) are on the closed list and have been fully dealt with. In this case white cells represent obstacles and the resulting shortest path is shown in yellow. The grey region in the remainder of the square constitutes cells that have not yet been considered by the search algorithm.

Dijkstra. The heuristic function $h(s)$ estimates the minimum cost from the vertex s to the goal s_{goal}.

It is the heuristic that controls how the A* search behaves.

An *admissible* heuristic *never* overestimates the cost from s to s_{goal}. It is always less than or equal to the true cost $d(s, s_{goal})$.

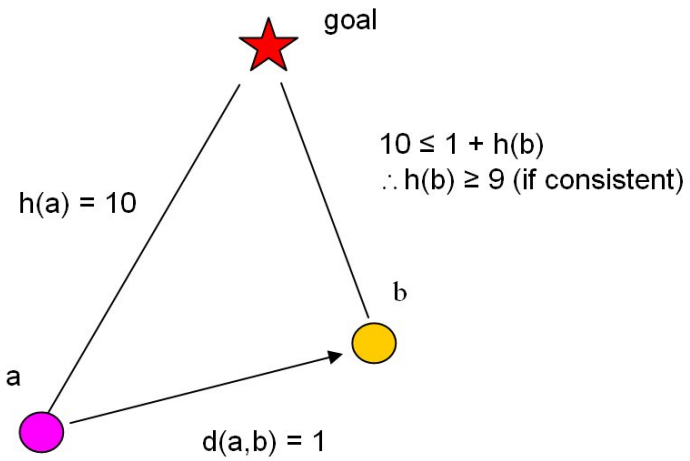


Figure 2.12: Admissibility: Note that for the heuristic to be consistent then $h(b) \leq 9$ must hold. For the heuristic to be merely admissible then $h(b) \geq 0$ is sufficient.

A *monotonic* or *consistent* heuristic satisfies the condition $h(x) \leq d(x, y) + h(y)$ - which is an expression of the triangle inequality.

Given a heuristic that is both *admissible* and *consistent*, A* is not only guaranteed to return the shortest path, it will do so by examining fewer vertices in the graph than any other search algorithm also using that heuristic.

However, as we explore in Chapter 6 the heuristic can be used to make a trade-off between speed and accuracy and this is particularly useful in robotic applications where often we are content to search for a *good* path rather than the best path available. Relaxing the requirement for consistency means that nodes are processed more than once and the search becomes less efficient. By contrast, relaxing the requirement for admissibility means that the search can run faster, but that the path returned may no longer be the shortest. Again, this speed increase is important for online settings.

The speed-accuracy trade off by varying the admissibility requirement can be characterized as follows:

- if $h(s)$ is only ever 0, then the A* search reduces to Dijkstra's algorithm which guarantees to find a shortest path. Without the benefit of the heuristic, a large number of nodes are expanded in the search,
- if $h(s)$ is always less than or equal to the true cost of moving to the goal from a given node, then A* will always find the shortest path. The lower $h(s)$ is compared with the true cost, the less efficient the search is,
- if $h(s)$ is exactly the cost of moving from n to the goal, then A* finds the shortest path and never expands nodes that are not on the shortest path. The search is extremely fast,
- if $h(s)$ is sometimes greater than the cost of moving from n to the goal, then A* will find a path but it is not guaranteed to be the shortest path. However

the search can run faster.

So as the pseudocode of Figure 2.10 shows, initially the open and closed lists are empty. The search begins by adding the start node s_{start} to the open list with its g-value set to 0, its h-value set to the heuristic estimate of the distance between the start and goal, and the f-value equal to the sum of these two terms, which in this initial case is simply $h(s_{start})$.

Note that nodes are added to the open list with a priority indicated by the f-value. Four operations are required to be performed on the list:

- *Inserting* nodes into the list (Insert),
- *Finding the highest priority node* and removing it (Top),
- *Checking* whether a node is in the list (Find), and
- *Updating* the priority of the node if it is already on the open list but its f-value through another parent node is better than the one currently stored on the open list. This could also be implemented by removing the node from the list and reinserting it with the new priority (Update).

The nature of these operations lends itself to representing the Open List as a mutable priority queue abstract data type. A mutable priority queue differs from a standard priority queue in that the priorities of entities in the queue are allowed to change after the entity has been added to the queue. How that priority queue is implemented depends on the size of the data being searched.

It is typical to denote the act of examining the nodes neighbouring a node n as *visiting* s, and the act of adding s to the Open list as *considering* s. Note that most nodes that are considered are also visited, those that have not yet been visited represent the frontier of the search shown in the green nodes which constitute the exterior of the search extent in Figure 2.11. From line 17 of the code listing of Figure

Data Structure	Insert	Delete	Top	Extract-Min	Find	Update
Unsorted Array	O(1)	O(N)	O(N)	O(N)	O(N)	O(N) (find) O(1) (update)
Sorted Arrays	O(N)	O(1)	O(1)	O(1)	O(log N)	O(log N) (find) O(N) (update)
Binary Heap	O(log N)	O(log N)	O(1)	O(log N)	O(log N)	O(log N) (find) O(log N) (adjust)
Fibonacci Heap	O(1)	O(log N)	O(1)	O(log N)	O(log N)	O(log N) (find) O(1) (decrease key)

Table 2.1: Amortized worst case run times for various data types which may be used to implement the Open List. Using heap based implementations improves the running time of planning problems considered in this thesis due to the low amortized running time of insertion, finding the minimum item and extracting the minimum item from the queue.

2.10 we note that each node considered is inserted once. In line 7 the *top* operation is performed once for each node visited. The membership check (find) is performed for every neighbour of every node visited (in line 16), so typically 8 times the number of nodes visited. The update of line 18 is performed comparatively rarely.

Table 2.1 illustrates some of the different underlying implementations of the priority queue available and their corresponding run-times. The choice of underlying data structure needs to take into consideration the size of the graph being planned over as well as the relative frequency of insertions, removals, updates and accessing the highest priority node on the tree. Heap-based implementations are best suited to planning problems.

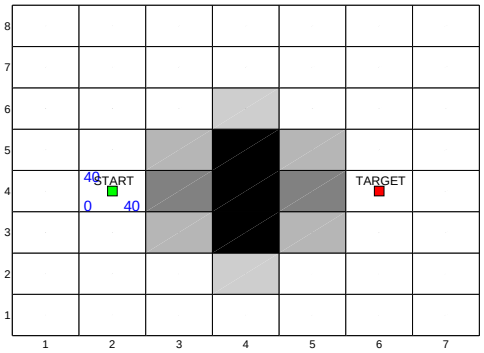
The search continues as follows while there are still nodes on the open list. First, the highest priority node is removed from the OPEN list and transferred to the CLOSED list. We call this node $s_{current}$. If this node happens to be the goal node then the search is complete and we go about reconstructing the path from the parent pointers maintained during the search. If not, we then examine all the neighbouring nodes of the node we have just pulled from the OPEN List: i.e. 8 neighbours in an 8-connected grid. If the neighbour s' is on the CLOSED list we simply skip it and forward to the next neighbour. If it is not on the OPEN list then we add it to the

open list with a g-value equal to the sum of that of its predecessor $g(s_{current})$ plus the cost of traversing from the predecessor to the neighbour $cost(s_{current}, s')$. Its priority is then the corresponding f-value which sums this g-value together with the heuristic estimate of the distance between the neighbour and the goal. In the case that the neighbour is already on the open list, we first compute this tentative new g-value and compare it with the g-value of the node already stored on the OPEN list. If it is NOT less than the g-value on the open list then we do nothing. If it is, we update the priority of the node on the OPEN List accordingly.

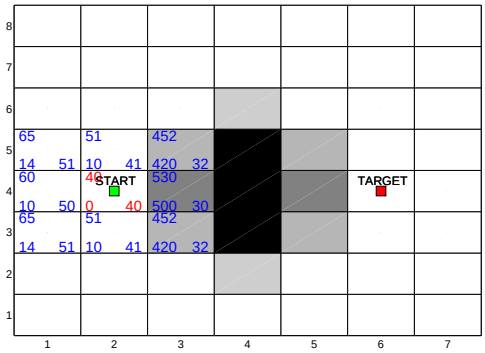
The process continues until either the OPEN list is empty (in which case a path has not been found) or the goal is removed from the OPEN List. Once we remove the goal we are able to compute the path by back propagating the parent pointers from the goal to the start node. Figure 2.13 shows the progression of an A* search on a simple environment.

2.4 D* Search

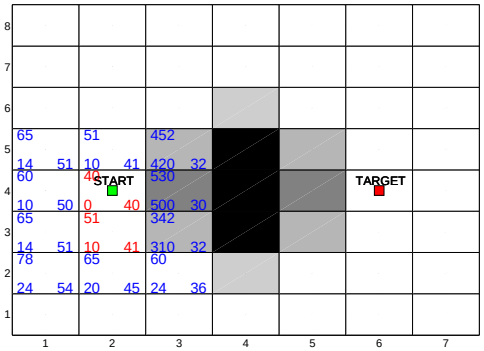
While A* works well for one-shot planning in known environments, this is seldom applicable to planning paths for mobile robots. Typically, the robot does not have full knowledge of the environment a-priori. The operating environment is often dynamic so edge costs in the graph are continually changing. The start location changes over time as the robot moves through the environment. Coupled with this the size of the terrain explored is usually large and this translates to graph representations that are large. While we could use A* to replan after changes to the environment are detected, the size of the problem means that replanning could take of the order of minutes (Koenig and Likhachev, 2002a) and may be required frequently, depending on how dynamic the environment is and how much of it is a-priori unknown to the robot.



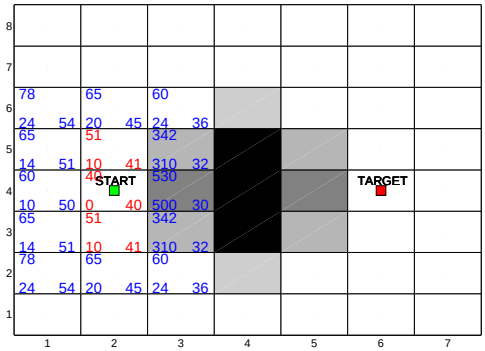
(a) First Move - The start node is added to the open list with g-value (bottom left of cell) set to zero, h-value (bottom right of cell) set to the euclidean distance metric heuristic value and the f-value (top left of cell) set to the sum of $g + h$. Note that the cost of traversing a white cell is 10, light grey is 200, darker gray 300, darkest gray 500 and black 1000.



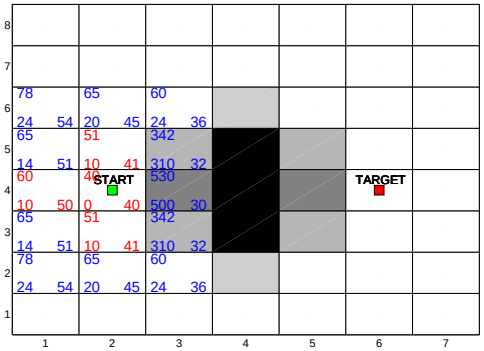
(b) Second Move - The start node is now coloured red as it has been removed from the OPEN list, expanded, and added to the closed list. Its eight neighbours have been added to the OPEN list.



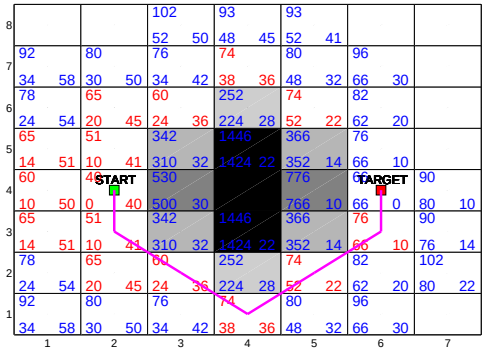
(c) Third Move - The cell directly below the start cell corresponds to the lowest f-value on the OPEN list. It is expanded and its neighbours added to the OPEN list.



(d) Fourth Move - The node directly above the start cell is removed from the OPEN list, expanded and added to the CLOSED list.



(e) Fifth Move - The search continues, removing the lowest f-value from the OPEN list.



(f) Final Move - Finally the Goal is removed from the OPEN list and the path from Start to Goal is reconstructed by backtracking through the parent nodes of the Goal node.

Figure 2.13: Walking through A*

Previous to the advent of the D* algorithm (Stentz, 1994b,a, 1993) in the mid-1990s, approaches to planning trajectories for mobile robots relied upon techniques such as planning a global path using known information and then locally circumventing newly detected obstacles by modifying the path locally (Goto and Stentz, 1987; Lumelsky and Stepanov, 1986). While such approaches find paths, they are sub-optimal in the sense that the sensor information is not incorporated into searching for a lowest cost path. Alternative approaches involve brute force replanning once discrepancies between the sensed environment and the a-priori map are detected. Although approaches such as representing the environment using quadtrees (Zelinsky, 1992) mitigate the inefficiencies of this technique somewhat, brute force replanning is grossly inefficient in typical mobile robot applications where the goal is far away, little is known of the environment to begin with and the search environment is large.

The D* algorithm generalizes A* to dynamic environments. It is what is known as an incremental heuristic search, i.e. a heuristically guided search that uses the results of previous searches to speed up subsequent ones. The development of D* springs from pivotal observations about the way a robot moves through an environment and its sensing and planning capabilities - the algorithm is hence tailored for a single robot equipped with a short range sensor exploring an environment alone. In such cases the limited range of the sensor means that changes to edge costs in the graph are only going to occur in the vicinity of the robot. This means that paths need only to be repaired and updated 'locally'. The robot also moves towards the goal in a monotonic fashion - small, unseen obstructions are most common and they only require simple path deflections. Computationally expensive backtracking is rarely required. Only the remaining portion of the path needs to be replanned and this gets progressively shorter as the robot bears down on the goal monotonically. Stentz (1995) reports that D* achieves a speed up over 300 times faster for large environments compared with brute force replanning using A* .

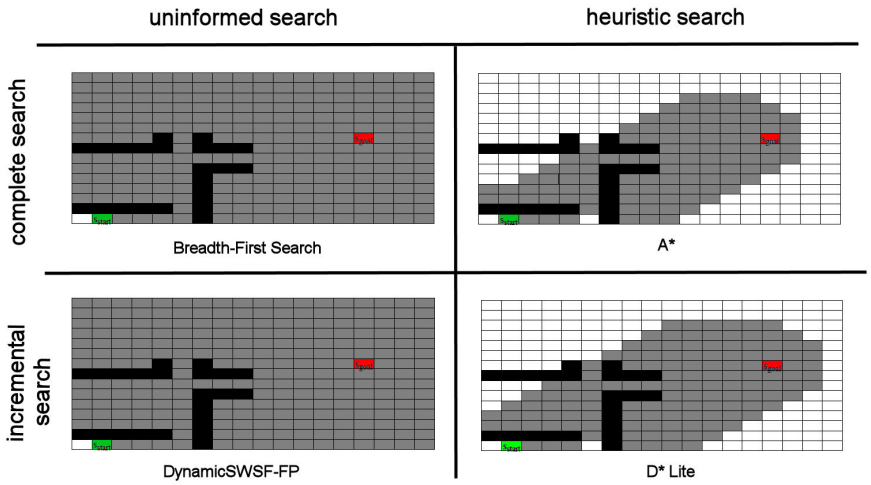
Like A*, D* operates over a discretized version of the environment and uses cost heuristics to narrow down the search space. Unlike A*, it conducts the search from the goal to the robot location (so that the equivalent g-value of A* is actually 0 at the goal in a D* search). This is done because changes occur within the sensor range of the robot close to its current location. If we were to plan from start to goal as in A*, changes close to the robot would need to be propagated towards the goal (a long way away). By planning backwards, we reduce the distance and influence over which changes need to be propagated.

Figure 2.14 illustrates the effectiveness of D* in replanning after a change in the environment is detected compared with uniform complete search, heuristic complete search (A*) and an uniform incremental search. We can see that after the change is detected the D* algorithm expands fewer nodes than the others in replanning the path.

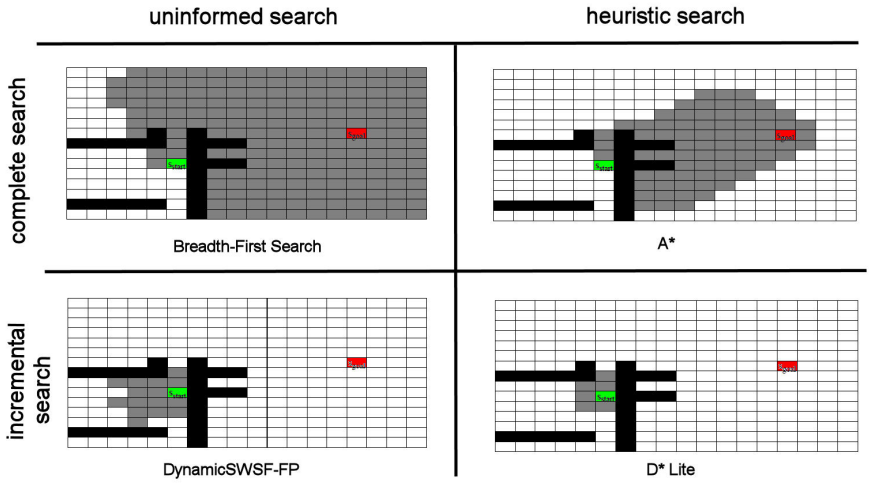
D* has undergone a series of evolutions since the first D* algorithm was first described by Stentz (1994a). The first algorithm is functionally equivalent to an optimal replanner (i.e. running A* search after each modification to the graph) but far more efficient. Provided the robot motion is accurately coupled to the grid cells represented in the algorithm, D* generates optimal trajectories. The original D* resembles A* in that the core of the algorithm is the Open list containing nodes to be evaluated. The *path cost* function $h(G, s)$, as in A*, is an heuristic estimate of the sum of the edge costs between the node under consideration, s and the goal node, G. Each state on the Open list has a *key*, $k(G, s)$ which is the minimum of $h(G, s)$ since s was placed on the open list.

In D*, the Open list is also used to propagate information about changes to edge costs between nodes in the graph. Each state carries a *tag*:

- NEW meaning it has never been placed on the open list,
- OPEN it is currently on the open list,



(a) Before the robot moves



(b) After the robot moves to the new start position

Figure 2.14: A comparison highlighting the benefits of heuristic searching combined with incremental planning - as is done in a D* search. The grey cells denote those expanded by the search. The green cell marks the start node of the search and the red cell is the goal. Figure (a) shows that heuristic searches in general expand less nodes than complete searches. In (b), the robot has moved to the cell marked by s_{start}. It finds the cell diagonally above and to the right, initially thought to be free, is now blocked. The incremental searches (DynamicSWSF-FP and D* Lite) reuse knowledge gained from the initial search in (a). The combined efficiency of heuristic search and incremental replanning makes D* the most efficient of the algorithms depicted. Figure adapted from Koenig (2002).

- CLOSED meaning it is no longer on the open list,
- RAISE if the node's key is less than the heuristic $k(G, s) < h(G, s)$,
- LOWER if the key is equal to the heuristic $k(G, s) = h(G, s)$.

The value of the node's tag on the OPEN list determines how the node is processed once it is chosen for expansion.

In the Original D* algorithm, path cost changes are propagated without regard for which expansions are relevant to the robot's current location. This is akin to running A* with $h(s) = 0$, or a backwards Dijkstra search. The Focussed D* Algorithm (Stentz, 1995) improves upon the original algorithm by focusing the propagation of the RAISE and LOWER states towards the robot's current location.

So just as A* can use its heuristic to focus the search towards the goal, Focussed D* uses a heuristic to focus the (backwards) search in the direction of the robot and hence reduce the total number of state expansions required. A *focusing heuristic* is introduced $g(s, R_i)$ which is the estimated path cost from the robot's current location R_i to the node at s. A new function $f(s, R_i) = h(G, s) + g(s, R_i)$ is also introduced. Just like the f-value in the A* algorithm, it provides an estimate of the path cost from the robot through the node s to the goal. States on the open list are sorted by a *biased* version of this cost. The sequence of states $R_0, R_1, \ldots, R_N$ denotes the nodes occupied by the robot when states were added to the Open list. The biased value attached to a node when it is added to the open list is hence calculated as $f_B(s, R_i) = f(s, R_i) + d(R_i, R_0)$, where $d(R_i, R_0)$ is the distance between the robot's initial location R_0 and its current location at R_i. To handle *ties* on the list, each node is added with a vector of values $< f_B, f, k >$ and ties are rectified in this order of priority. Ties in k-values are ordered arbitrarily.

In implementing D* on the MRG suite of robots we chose to use the D* Lite algorithm (Koenig and Likhachev, 2002a,b,d).

```
CalculateKey(s)
1  return [min(g(s),rhs(s))+h(s_start,s) +k_m; min(g(s),rhs(s))];

Initialize()
2  U = ∅
3  k_m = 0;
4  for all s ∈ S
       do rhs(s) = g(s) = ∞
5  rhs(s_goal) = 0;
6  U.Insert(s_goal,CalculateKey(s_goal));

UpdateVertex(u)
7  if (u ≠ s_goal)
       then rhs(u) = min_{s'∈Succ(u)}(c(u, s') + g(s'));
8  if (u ∈ U)
       then U. Remove(u);
9  if (g(u) ≠ rhs(u))
       then U. Insert(u, CalculateKey(u));

ComputeShortestPath()
10 while (U. TopKey() < CalculateKey(s_start) OR rhs(s_start) ≠ g(s_start))
11       do k_old = U. TopKey();
12          u = U. Pop();
13          if (k_old < CalculateKey(u))
14             then U. Insert(u, CalculateKey(u));
15             else if (g(u) > rhs(u))
16                     g(u) = rhs(u);
17                     for all s ∈ Pred(u)
                           do UpdateVertex(s);
18             else
19                     g(u) = ∞;
20                     for all s ∈ Pred(u) ∪ u
                           do UpdateVertex(s);
```

Figure 2.15: The D* Lite Algorithm, continued over page

This algorithm implements the same navigation strategy as Focussed D* (Koenig and Likhachev, 2002b) yet is algorithmically different. It builds on the Lifelong Planning A* Algorithm developed by the same authors (Koenig and Likhachev, 2002c; Koenig et al., 2004) which (when compared to the original D*) has properties that are provably similar to A* and thus offer rigorous proofs on its efficiency and a solid theoretical foundation. It requires 30% fewer lines of code to implement (Koenig and

```
MAIN()
21  s_last = s_start;
22  Initialize();
23  COMPUTESHORTESTPATH();
24  while (s_start ≠ s_goal)
        do
25          if g(s_start) = ∞)
              then there is no known path
26          s_start = argmin_{s'∈Succ(s_start)}(c(s_start, s') + g(s'));
27          Move to s_start
28          Scan graph for changed edge costs;
29          if any edge costs changed
30            then k_m = k_m + h(s_last, s_start);
31                 s_last = s_start
32                 for all directed edges (u, v) with changed edge costs
33                     do Update the edge cost c(u, v);
34                        UPDATEVERTEX(u);
35                 COMPUTESHORTESTPATH();
```

Figure 2.15: The D* Lite Algorithm, continued from previous page

Likhachev, 2002b) and as Figure 2.15 illustrates uses only one tie-breaking criterion when comparing priorities. The Original D* algorithm by comparison employs a complex series of nested-if statements making the algorithm difficult to analyse and implement. Furthermore, D* Lite is considered easier to implement and extend, as evidenced by the original creator of D* choosing to extend D* Lite rather than his own algorithm in (Ferguson and Stentz, 2005a; Mills-Tettey et al., 2007).

One of the primary differences between D* and D* Lite is the components of the keys used to sort the Open list. This means that slightly different priorities are assigned to the same states given identical problems tackled by the two algorithms.

D* Lite maintains estimates of the goal distance $g(s)$ from node s as well as the start distance $h(s)$. It introduces a third metric $rhs(s)$ which is a one-step lookahead

estimate of the goal distance based on the g-values of s's neighbours.

$$rhs(s) = \begin{cases} 0 & \text{if } s = s_{goal} \\ min_{s' \in succ(s)}(g(s') + c(s', s)) & \text{otherwise} \end{cases} \tag{2.8}$$

Intuitively, the rhs-value has the potential to be better informed than its corresponding g-value, as it draws on the values of its neighbours and thus changes are propagated to the rhs-value one step before the g-value. The use of the rhs-value in D* Lite allows the introduction of the notion of *inconsistency*. An inconsistent node is one whose g-value does not equal its rhs-value and the priority queue under D* lite contains only the inconsistent nodes. It is these nodes that potentially need to be updated and expanded again to compute the shortest path once edge costs have changed.

D* Lite dispenses with the complex notion of tagging states on the OPEN list - and refers to the OPEN list as the 'Priority Queue'. A node is either on the priority queue or it is not, and unlike in Focussed D* no record needs to be kept of where the robot was when the node was inserted on the queue. Instead, it is inserted with a 2-member vector - the key, k, calculated as

$$k =< min(g(s), rhs(s)) + h(s_{start}, s) + k_m; min(g(s), rhs(s)) > . \tag{2.9}$$

Note also the similarity between the first part of the key is like the biased f_B value from Focussed D*. It borrows the same technique of computing a lower bound on the f-value in order to avoid having to reorder the priority queue after the robot has moved. The first part of Equation 2.9 depends on the start location s_{start}, and when the robot moves from s_{start} to s' this first part of the key will decrease by at most $h(s_{start}, s')$. This quantity is the same for all nodes currently on the priority queue, so we can avoid reordering the queue by maintaining a record distance the robot has

travelled in k_m and adding this quantity to the key of each node.

Note that in lines 12 to 14 of *ComputeShortestPath* of Figure 2.15 once the top node is popped from the priority queue it uses *CalculateKey* to compute the priority it should currently have, given that the robot may have moved since the node was inserted on the priority queue. If the old priority is less than the new one, the node is not expanded and is instead reinserted into the priority queue with correct priority.

D* Lite begins the search by initializing the priority queue to an empty state, setting the distance offset k_m to 0, and setting the rhs and g-values for all nodes to infinity. The rhs-value of the goal node is then set to 0 and the goal node s_{goal} is inserted into the priority queue. In line 23 of Figure 2.15 the first call to *ComputeShortestPath* is made. It simply runs a backwards A* search to compute an initial path. Note that a call *ComputeShortestPath* can terminate when either the start vertex is consistent, or the start node has a key that is less than or equal to any remaining on the priority queue and hence offers the shortest path of those possible.

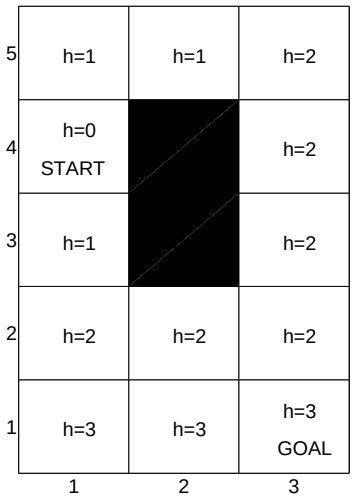
In lines 24 to 27 of *Main* the algorithm uses the results of *ComputeShortestPath* to move along the shortest path from s_{start} to s_{goal} by finding the successor to the current node s' that minimizes $c(s, s') + g(s)$.

Once the robot has moved, in lines 29 to 34 it scans the environment and graph for any edge cost discrepancies. Affected nodes then have their position in the priority queue updated according to the new value, and in line 35 a further call to *ComputeShortestPath* is made to compute the new optimal path. It iterates through the process of calculating the next move, moving, scanning and recomputing if necessary until the robot reaches the goal. Figure 2.17 shows a toy example of D* searching and replanning in operation, wherein the robot conducts an initial search, moves one step along the path, senses and detects a change in the environment and must replan.

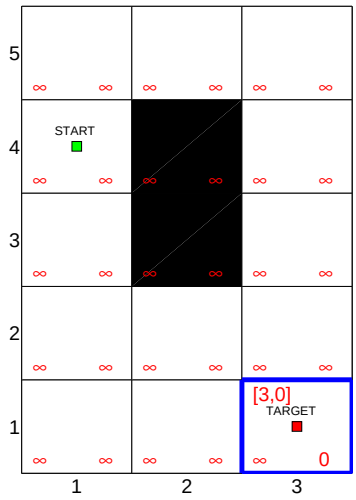
Figure 2.18 provides a large scale example of D* in operation and illustrates the magnitude of the efficiency gain that D* offers over replanning with A* in a robotic

traverse. The figure was generated using tools implemented during the course of the thesis, which will be described in Chapter 3.

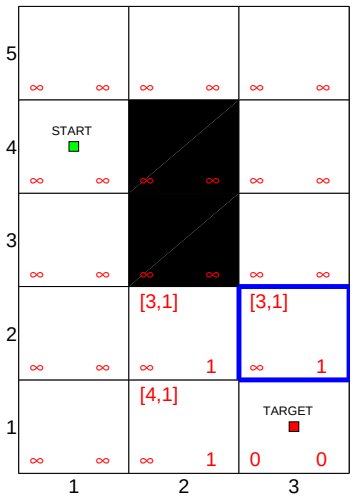
One undesirable characteristic of the paths generated in both Figure 2.17 and 2.18 is that paths are restricted to moving the robot between the centre of a grid cell to the center of one of the eight cells neighbouring it. This makes for jagged paths that can lead to gross inefficiencies in the dynamic control of a robot. In the section that follows we discuss an algorithm that remedies this problem using interpolation and thus allows for a continuous range of headings.



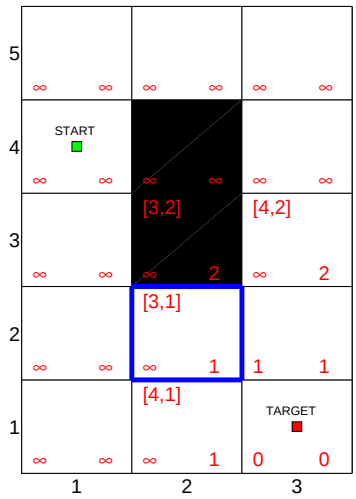
(a) Costs and Heuristic Values. The costmap is a binary grid world, clear cells have traversal cost 1 while black cells have infinite traversal cost. The heuristic estimate of each cell is shown from the start node at $\{4, 1\}$.



(b) Initial step. The goal node's key is calculated as $< 3, 0 >$. The goal node is then added to the priority queue with this key.

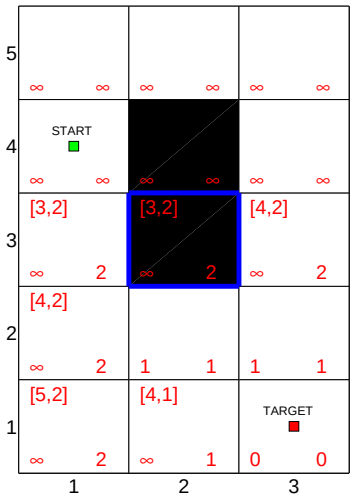


(c) Second step. The goal node, being the only node on the priority queue, is popped off the queue. The keys of its 3 neighbours are calculated and they are inserted into the priority queue.

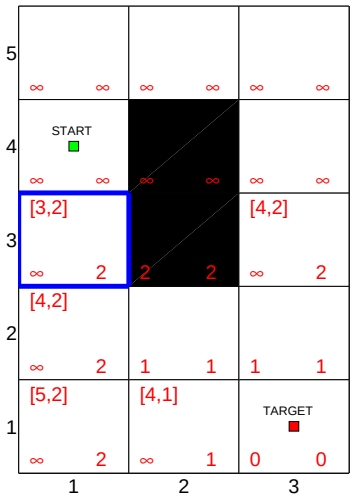


(d) Third step. Two nodes on the priority queue are tied with the lowest key value. We arbitrarily break the tie choosing to expand cell $\{2, 3\}$, and adding its neighbours to the priority queue.

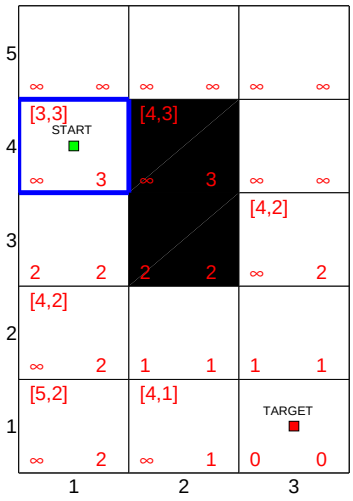
Figure 2.16: Walking through D* - First Call to Compute Shortest Path. Figure continued overleaf. Key values are shown in the top left corner of the cell, *rhs*-values in the bottom right corner and *g*-values in the bottom left corner. We have used the maximum of the absolute differences of the x and y coordinates of both cells as the heuristic. At each step we show the evolution of the *g*-values, *rhs*-values and keys of each cell as the highest priority node is removed from the queue and expanded. At any point in time the priority queue contains all the inconsistent cells for which $g(s) \neq rhs(s)$. The cell with the smallest key which will be expanded next is shown in bold.



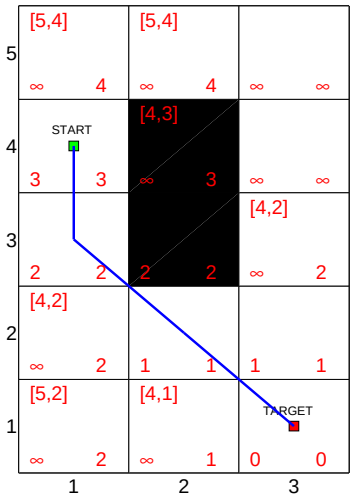
(e) Fourth Step



(f) Fifth Step

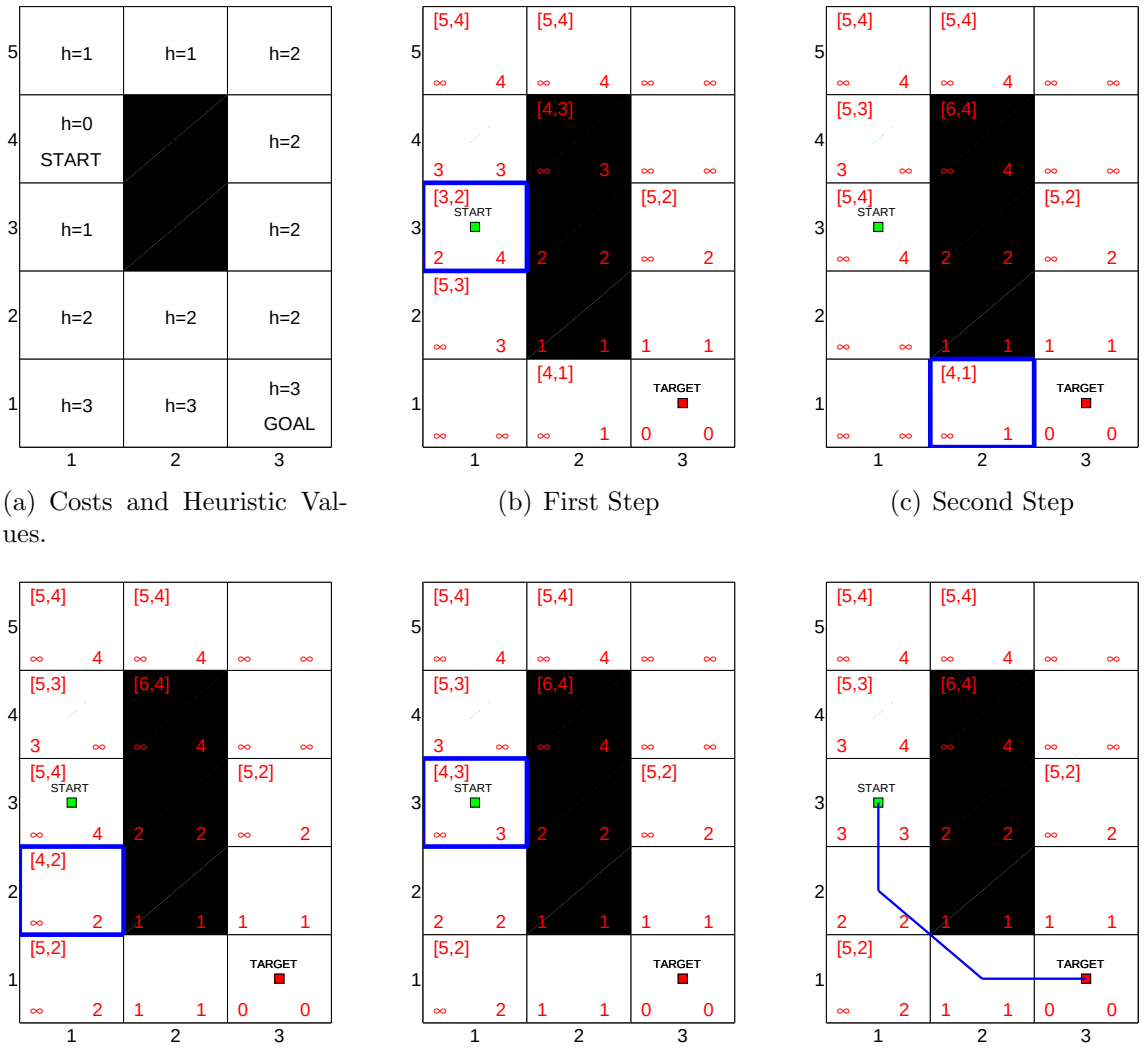


(g) Final Step



(h) Path

Figure 2.16: Walking through D* - First Call to Compute Shortest Path. Continued from previous page. We continue expanding neighbours until the start node becomes consistent in (g), at which point we are able to trace the shortest path from start to goal in (h). Figure based on example in Koenig and Likhachev (2002d).



(a) Costs and Heuristic Values.

(b) First Step

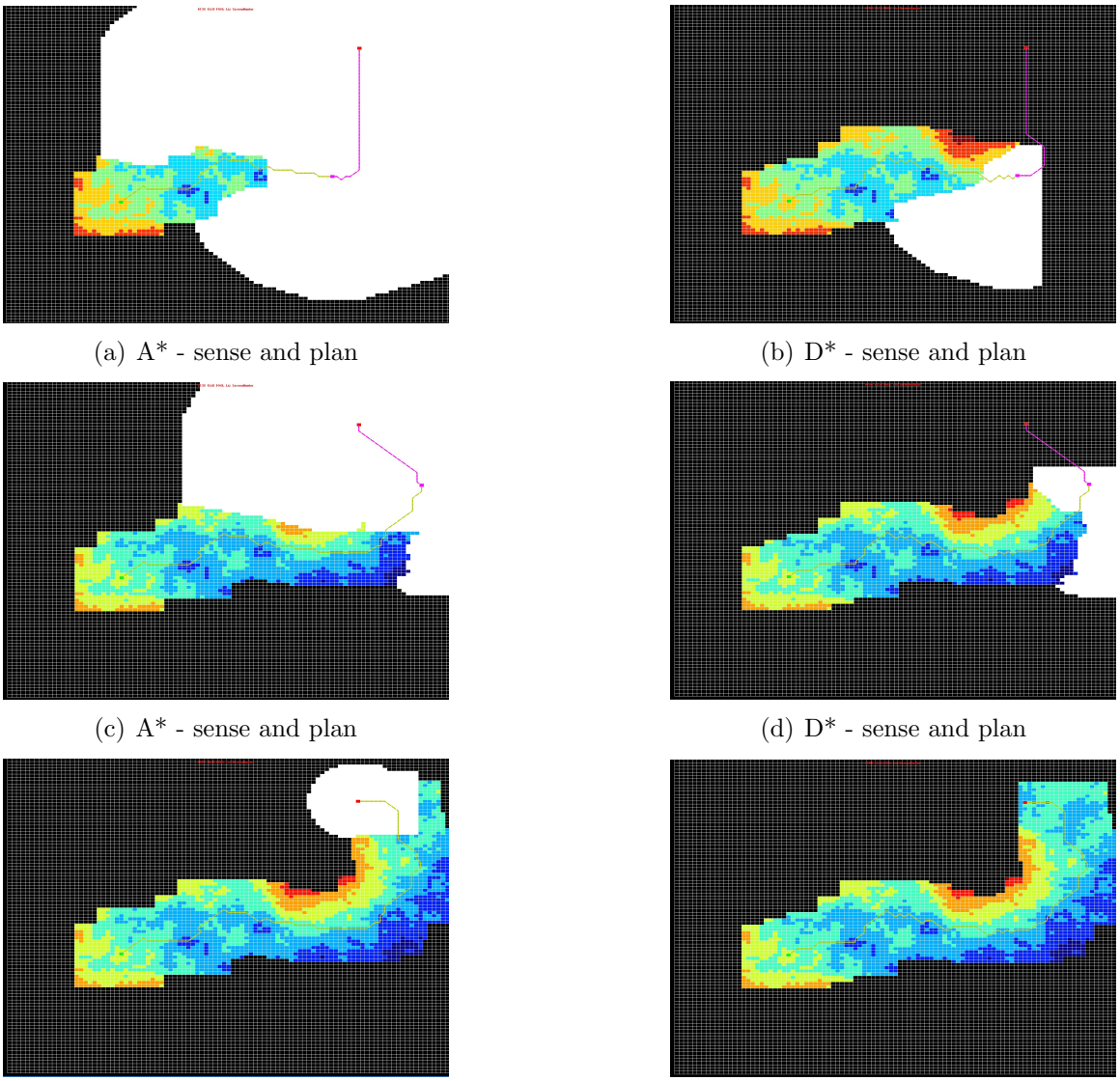
(c) Second Step

(d) Third Step

(e) Fourth Step

(f) Path

Figure 2.17: Walking through D* - Second Call to Compute Shortest Path. The robot has now traversed one grid cell towards the goal from the previous step. This new start position is reflected in the updated heuristic values in (a). Additionally, in the course of moving the robot has observed that cell (2,2) is actually occupied and the costmap has been updated accordingly. Nodes affected by the change in edge costs which result from discovering (2,2) is now blocked are updated and added to the priority queue with a new priority if this discovery has caused them to become inconsistent. Figures (b) - (e) show the progress through the second call to ComputeShortestPath, until the search terminates in (f) when the start node becomes consistent and we can trace the path.



(a) A* - sense and plan

(b) D* - sense and plan

(c) A* - sense and plan

(d) D* - sense and plan

(e) A* - sense and plan

(f) D* - sense and plan

Figure 2.18: Sensing and Planning, A* vs D*. At each instance the nodes expanded in the last replanning step are shown in white. The traverse was simulated as the robot having a sensing range of $\pm$ 10 cells. Black cells represent unknown areas. The robot moved 5 cells along the planned path and re-scanned the environment. Note how many more cells are expanded by A* at each location.

2.5 Field D* Search

The problem with the Focussed D* and D* Lite algorithms is that they restrict the robot to moving in increments of 45 degree headings. The unnatural, jerky looking paths (see Figure 2.19) they produce often need to be smoothed in a post-processing step or used in combination with a local planner that takes into account the kinematic and dynamic constraints of the robot. Field D* (Ferguson and Stentz, 2005b, 2006b) seeks to rectify this by using interpolation during planning to produce path costs for arbitrary positions within a grid cell and thus allow a continuous range of headings. As Figure 2.20 shows, Field D* extends D* Lite. The critical difference is the introduction of the *ComputeCost* function called in line 4 of *UpdateState*.

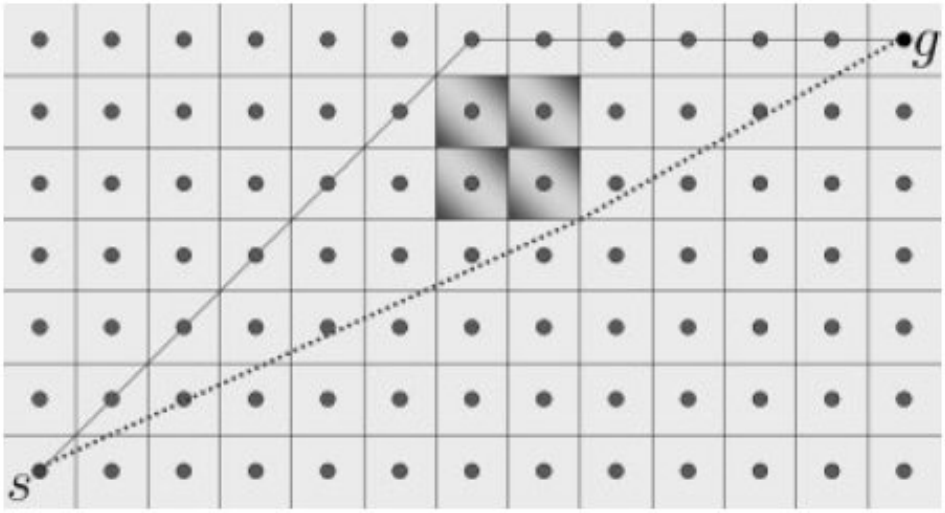


Figure 2.19: Traversing between the center of grid cells can lead to suboptimal paths. The solid line shows the path D* and D* Lite would produce. The dotted line shows an interpolated path like Field D* is capable of computing. Image from Ferguson and Stentz (2005b).

Here Field D* replaces D* Lite's simple rhs calculation of

$$rhs(u) = min_{s' \in \mathrm{Succ}(u)}(c(u, s') + g(s')); \tag{2.10}$$

with a more sophisticated interpolation-based cost calculation.

What happens here is that Field D* relaxes the requirement that transitions from

```
ComputeCost(s, s_a, s_b)
 1  if (s_a is a diagonal neighbour of s)
 2     then s_1 = s_b; s_2 = s_a
 3     else
 4          s_1 = s_a; s_2 = s_b
 5          c is traversal cost of cell with corners s, s_1, s_2;
 6          b is traversal cost of cell with corners s, s_1 but not s_2;
 7          if (min(c, b) = ∞)
 8             then v_s = ∞;
 9             else if (g(s_1) ≤ g(s_2))
10                     v_s = min(c, b + g(s_1));
11             else
12                     f = g(s_1) − g(s_2)
13                     if (f ≤ b)
14                        then if (c ≤ f)
15                                then v_s = c√2 + g(s_2);
16                                else
17                                     y = min( f/√(c^2 − f^2), 1 );
18                                     v_s = c√(1 + y^2) + f(1 − y) + g(s_2);
19                        else
20                                if (c ≤ b)
21                                   then v_s = c√2 + g(s_2);
22                                   else
23                                        x = 1 − min( b/√(c^2 − b^2), 1 );
24                                        v_s = c√(1 + (1 − x)^2) + bx + g(s_2);
25                                        return v_s

key(s)
1  return[min(g(s), rhs(s)) + h(s_start, s); min(g(s), rhs(s))];

UpdateState(s)
2  if s was not visited before
      then g(s) = ∞
3          if (s ≠ s_goal)
4             then rhs(s) = min_{(s',s'')∈connbrs(s)} ComputeCost(s, s', s'')
5                  if (s ∈ OPEN)
                      then remove s from OPEN
6                  if (g(s) ≠ rhs(s))
                      then insert s into OPEN with key(s)
```

Figure 2.20: The Field D* algorithm, continued next page

```
ComputeShortestPath()
 7  while (min_{s∈OPEN}(key(s)) < key(s_start) OR rhs)(s_start) ≠ g(s_start))
        do
 8          remove state s with the minimum key from OPEN;
 9          if (g(s) > rhs(s))
10             then g(s) = rhs(s);
11                  for all s' ∈ neighbours(s)
                        do UpdateState(s');
12                 else
13                      g(s) = ∞;
14                      for all s' ∈ neighbours(s) ∪ s
                            do UpdateState(s');

15  g(s_start = rhs(s_start) = ∞; g(s_goal) = ∞;
16  rhs(s_goal) = 0; OPEN = ∅;
17  insert s_goal into OPEN with key(s_goal);
18  while 1
19      do ComputeShortestPath();
20         Wait for changes in cell traversal costs;
21         for all cells x with new traversal costs
22             do for each state s on a corner of x
23                    do UpdateState(s)
```

Figure 2.20: Field D*, continued from previous page.

node s must be limited to the centre of a cell represented by one of its 8-connected neighbours. Instead, as shown by Figure 2.21 Field D* allows a transition from s to any point on the boundary of its grid cell. The location of a node is moved, instead of being at the centre of the cell, nodes are now located at the corner of grid cells. The edges of the graph connect nodes that reside at the corners of the same grid cell.

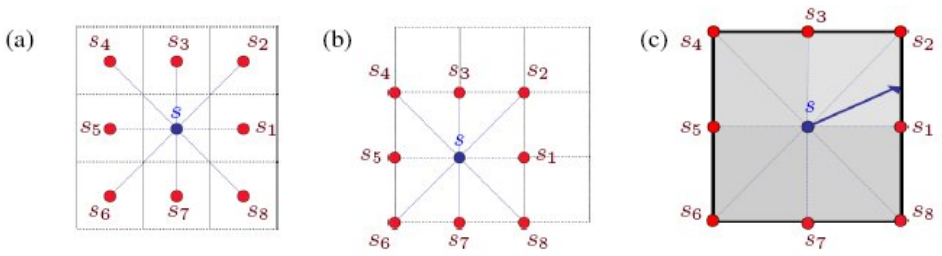


Figure 2.21: Field D* locates grid cell nodes at the corner of the cell (b) rather than at the centre (a). Shortest paths are calculated to the nearest edge (c) rather than to other nodes. Image from Ferguson and Stentz (2005b).

The optimal path stemming from node s in Figure 2.21 must pass through one of the eight edges $\overrightarrow{s_1s_2}, \overrightarrow{s_2s_3}, \overrightarrow{s_3s_4}, \overrightarrow{s_4s_5}, \overrightarrow{s_5s_6}, \overrightarrow{s_6s_7}, \overrightarrow{s_7s_8}, \overrightarrow{s_8s_1}$. If the value of every point along these lines was known then calculating the optimal traversal would simply be a matter of minimizing over all $c(s, s_b) + g(s_b)$ - but there are an infinite number of such points so computing $g(s_b)$ for all of them is not feasible. Field D* gets around this using linear interpolation to provide an approximation to $g(s_b)$ for each point on the boundary using the value of the corner nodes delimiting the edge on which it lies (say s_1 and s_2).

To compute the path costs to s using an edge $\overrightarrow{s_1s_2}$, the path costs $g(s_1)$ and $g(s_2)$ are used in conjunction with the traversal costs c of the center cell and b of the bottom cell. Figure 2.22 shows the relative locations of these entities, together with the four possible ways that a robot may be able to traverse from s to the edge $\overrightarrow{s_1, s_2}$.

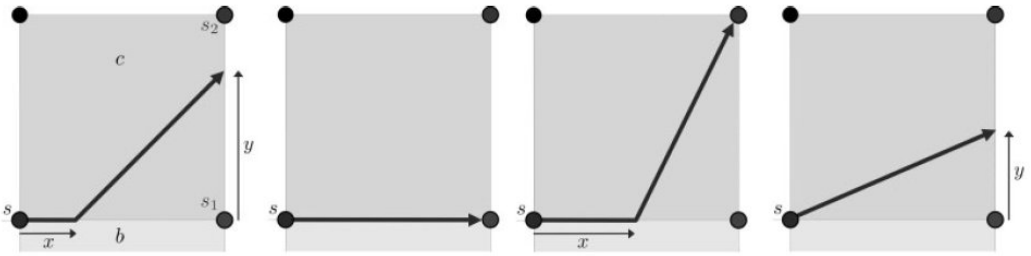


Figure 2.22: Possible ways of computing a shortest path from s to s_2. Image taken from Ferguson and Stentz (2006b).

The linear interpolation assumes the cost of any point s_y residing a distance $y \in [0, 1]$ along the edge $\overrightarrow{s_1s_2}$ is given by

$$g(s_y) = yg(s_2) + (1 - y)g(s_1). \tag{2.11}$$

Given this equation, the path cost of s given s_1, s_2, b and c can be computed:

$$v_s = \min_{x,y}[bx + c\sqrt{(1 - x)^2 + y^2} + yg(s_2) + (1 - y)g(s_1)]. \tag{2.12}$$

To solve for x and y it is useful to transform the problem. If we represent the cost

of traversing edge $\overrightarrow{s_1 s_2}$ as $f = g(s_1) - g(s_2)$ (the difference in path cost between the two nodes), then equation 2.12 can be rewritten as:

$$\min x, y[bx + c\sqrt{(1-x^2)+y^2} + (1-y)f + g(s_2)]. \tag{2.13}$$

Looking at this equation, the first term bx represents the cost of travelling a distance x from s along the bottom edge. The second term, $c\sqrt{(1-x^2)+y^2}$ is the cost of cutting across the centre square to point y on the right-hand edge, and the third term $(1-y)f$ represents the cost of traversing from that point y up to the node at s_2. So what we are in essence now trying to find the cheapest path from s through s_2. Note that this does not preclude the cheapest path via that edge being through $s1$, as we may find the cheapest path corresponds to a y-value of 0, in which case the path passes through s_1 and directly along the bottom edge to s.

A second point to note is that at least one of x, y that minimizes v_s will be zero or one. This means the situation described by the left-most example of Figure 2.22 will never arise. Although this is proven mathematically in (Ferguson and Stentz, 2006b), intuitively if it is cheaper to cut across the centre cell c than traverse the boundary, then it must be cheaper to avoid the boundary completely and traverse purely across the centre cell. Therefore no optimal path will traverse both the bottom and right edges and cut across the centre cell. Thus the only types of traversals we need to consider are those represented by the three right-most figures of Figure 2.22.

Which of these is cheapest depends on the relative traversal costs of c and b as well as the difference $f = g(s_1) - g(s_2)$ in path cost between the nodes at s_1 and s_2. If we let $k = f = b$ we can find the 'tipping point', the cost of the path from s through $\overrightarrow{s_1 s_2}$ is

$$v_s = c\sqrt{1+y^2} + k(1-y) + g(s_2). \tag{2.14}$$

Taking the derivative and setting the result to zero we obtain:

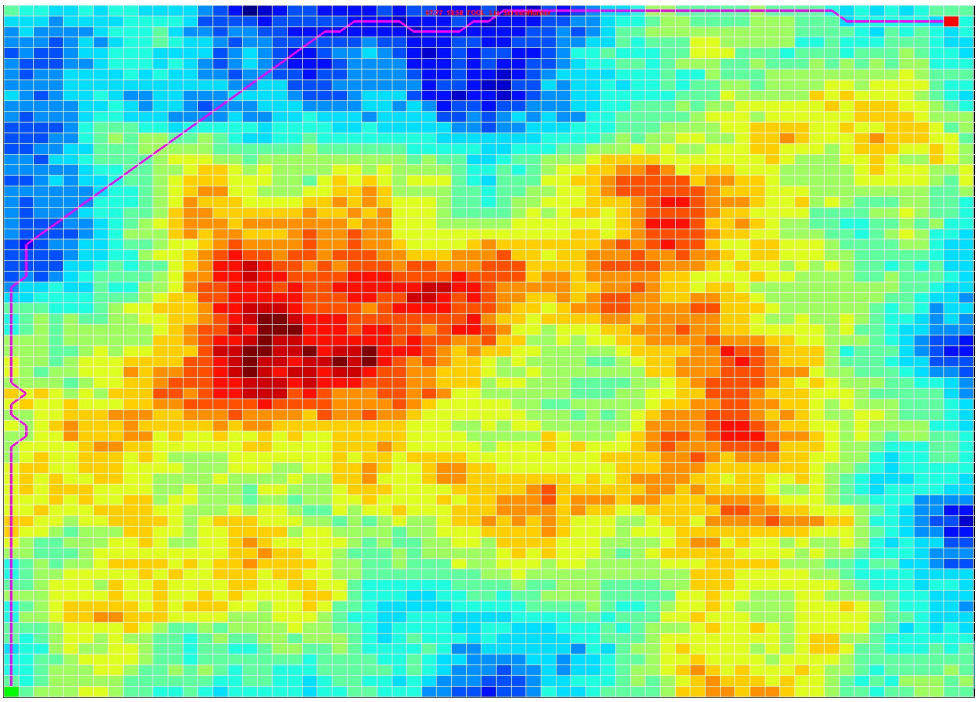
$$y* = \sqrt{\frac{k^2}{c^2 - k^2}}. \tag{2.15}$$

The same path cost computations apply whether we use the right edge or the bottom edge. All we need to work out is which edge is cheaper.

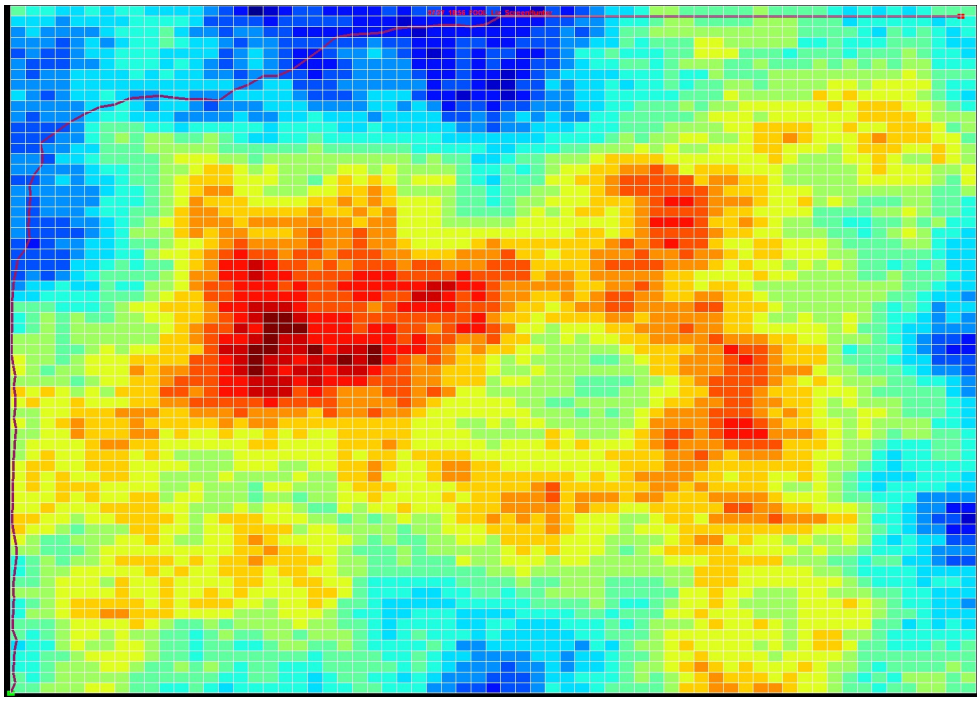
If:

- $f < 0$, the optimal path travels straight across to s_1 as represented by Figure 2.22 (ii). The cost is $v_s = min(c, b) + g(s_1)$ and represents line 10 of the *ComputeCost* function,
- $f < b$ then we use the right edge and compute y as per equation 2.15, plugging the result into equation 2.14,
- $f > b$ then use the bottom edge, substituting $k = b$ and $y = (1 - x)$ into equation 2.15,
- in either case, if $c \leq b$ then it is cheaper to cut across the center node so the path cost is $v_s = c\sqrt{2} + g(s_2)$.

These calculations are featured in the *ComputeCost* function of Figure 2.20. For each node under examination we compute the minimum interpolated edge cost for each of its surrounding 8 edges. Only the corner nodes go on the graph, but once we have completed a path search the interpolated path costs are computed in the path extraction step, and these interpolated nodes hence appear in the final path. Figure 2.23 contrasts the paths computed by the MRG D* lite implementation with those computed with the MRG Field D* implementation. Note the 'curviness' of the Field D* paths.



(a) D* Lite



(b) Field D*

Figure 2.23: Paths Generated from D* Lite vs Field D*, output generated via MRG Path Planning toolkit described in Section 3.1.

2.6 Chapter Summary

We have used this chapter to provide useful background to the path planning problem and to position our choice of Path Planner, the A*/D* search family amongst competing techniques. We concluded the chapter with a detailed overview of the workings of the A*, D* and Field D* algorithms. These algorithms represent the state-of-the-art in terms of path planning algorithms used in field robotic systems. They are capable of operating extremely quickly and efficiently over approximate representations of large-scale environments. What these algorithms fail to accommodate however, is the fact that the cost we attribute to grid cells is inherently uncertain. In the following chapters we will augment these fast planning algorithms with techniques to deal with uncertainty. First however, Chapter 3 will describe the implementation of these algorithms amongst other systems work necessarily undertaken as part of this thesis.

Chapter 3

Systems to Support Autonomous Exploration

This is a robotics thesis, and robots are systems. It follows then, that an essential component of a robotics DPhil is work devoted to the design, construction, maintenance and adaptation of robotic systems. In this chapter we document the effort that has gone into extending the robotic systems operated by the Oxford Mobile Robotics Group in support of the research aims of this thesis. To facilitate our goal of autonomous exploration, we require the capability to do both *path planning* and *navigation*. Here we detail software systems that we have successfully fielded which accomplish both tasks.

A substantial amount of time in this thesis was spent equipping MRG's robotic platforms with a path planning capability. Following on from the description of the A*, D* and Field D* algorithms provided in Chapter 2, we begin here by detailing the path planning software systems developed that run on MRG's robots in conjunction with the MOOS robot operating system. As you will see, implementing a path planning algorithm on the robots requires more than simply implementing the algorithm itself. There is a whole suite of support applications which take output from the robot's sensors (in our case 2D lasers and stereo cameras) and use this information to create costmaps which the robot can use for planning. We also need to be able to translate the output of the planner into actuation commands for the robot and this

required writing code to interface with the mobility platforms of the robots. This suite is described here.

A navigation capability is an essential part of any autonomous platform - without it we cannot ascertain where the robot is at any instant in time! In the final part of the chapter, we provide a brief treatise on robotic navigation - a capability which for the remainder of the thesis is an assumed competence. We also describe the MRG software systems that use Simultaneous Localization and Mapping (SLAM) to provide navigation data to the robot.

3.1 Path Planning Systems

In order to conduct research into robot autonomy, it is first necessary to have some kind of autonomous capability on the robots. A substantial amount of time in this thesis has been spent implementing the groundwork for autonomy - a path planning capability. The reasons for doing this are two-fold. In implementing it, understanding was gained of how the planning algorithms work and how they may be modified/extended. The second is that MRG previously did not have any planning capability for the robotic platforms so one needed to be implemented. In this section some description of the fundamental processes and their implementation is provided.

Figure 3.1 shows a high level overview of the MRG robotics suite. Processes are designed to work with the Mission Oriented Operating Suite (MOOS) (Newman, 2010). Processes communicate via a central hub by sending and subscribing to messages. Processes requiring high throughput may also send and receive data via the Fast Data Server - it establishes point to point links between processes to transfer arbitrary data faster and in bigger quantities than is possible through the MOOSDB message format. By nature, the planning and costmap generation processes are highly reliant on data from the robot's sensors (such as the laser and stereo camera processes

mentioned), on navigation data (from processes like **pNav**) to work out where the robot currently is, and on platform control software (such as **iPlatform**) to turn actuation commands from the planner into movement. Fortunately, these processes were already present and stable (although under continuing development), in the MRG source tree.

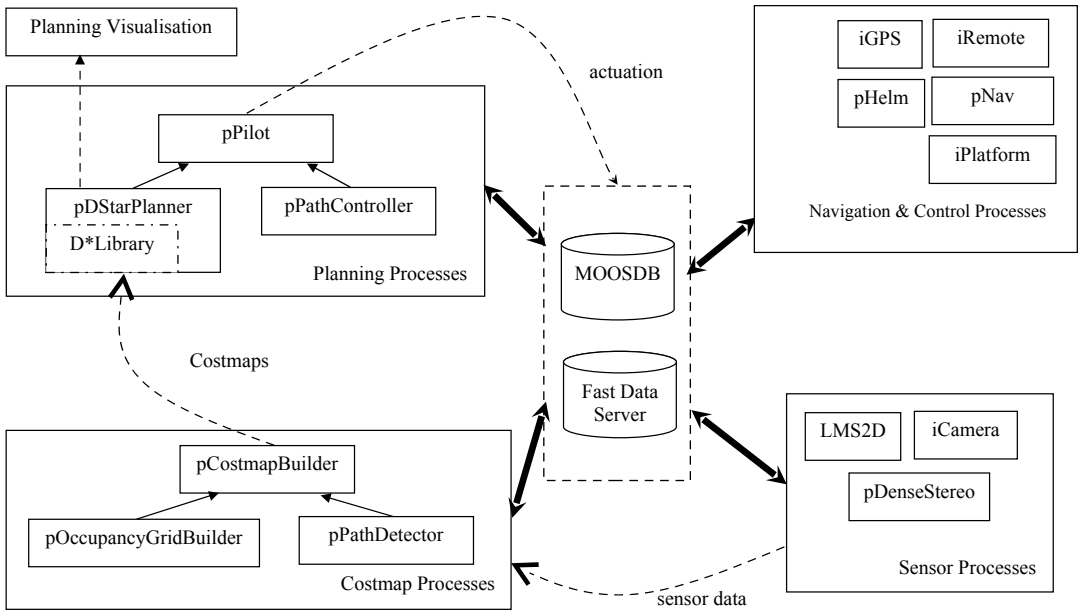


Figure 3.1: Block Diagram of the relation of the planning and costmap building processes to selected other MRG Software processes

3.1.1 The D* library

The D* library provides a graph based implementation of the D* Lite algorithm of Figure 2.15. Figure 3.2 shows the class hierarchy of the D* library. As the pseudo code shows, the algorithm (an instance of the class CDStar is invoked for each problem) requires a priority queue (CDStarQueue), and a graph (CGraph). We also invoke instances of the CTrajectory class to store the current planned trajectory as well as the historical trajectory of the robot. The class also contains an instance of the CDStarCostMap data type - by keeping the costmap separate we are able to store an implicit version of the search graph. This way nodes are only added to the graph when they

become relevant to the search. This 'slim-line' graph is faster to search and memory efficient, which is important when searching large-scale environments. Additionally, it is necessary for D* to maintain its own local copy of the costmap so that it can determine changes to the costmap that may have occurred between iterations of the planner. This ability to detect changes in the graph is critical to D*'s operation, and somewhere in the system it is necessary to cache the copy of the costmap last known to D* before a planning iteration. It was judged that the best place to do this is in the planning process itself as the costmap builder process is continually receiving sensor data and updating itself and the potential for synchronization errors in communicating between the two processes when planning has started and finished is too great a risk.

The CGraph class contains an instance of the Boost Graph Library (Boost, 2010) Graph type. The Boost implementation was chosen for its ready-made implementation of basic graph management functions - adding vertices, defining edges between vertices, finding the neighbours of vertices etc. We augment this graph type with a map data structure which relates $< x, y >$ locations represented in the cost map to vertex descriptors to enable easy determination of whether locations are currently stored in the graph.

Each vertex in the graph is assigned a unique vertex descriptor. The node stores a pointer to an instance of the CNode data type which encodes all the necessary data for a particular grid cell that has been added to the graph. Here is where data crucial to the operation of the D* search is stored; g-values, rhs-values, the key, the parent node for path retrieval, as well as the $< x, y >$ location of the node in the grid.

The CDStarKey data type inherits from the pair data type of the Standard C++ library. We augment it by adding a comparison operator to implement the queue ordering required by D* Lite.

The CDistanceHeuristic class is a functor which maintains the state of the D*

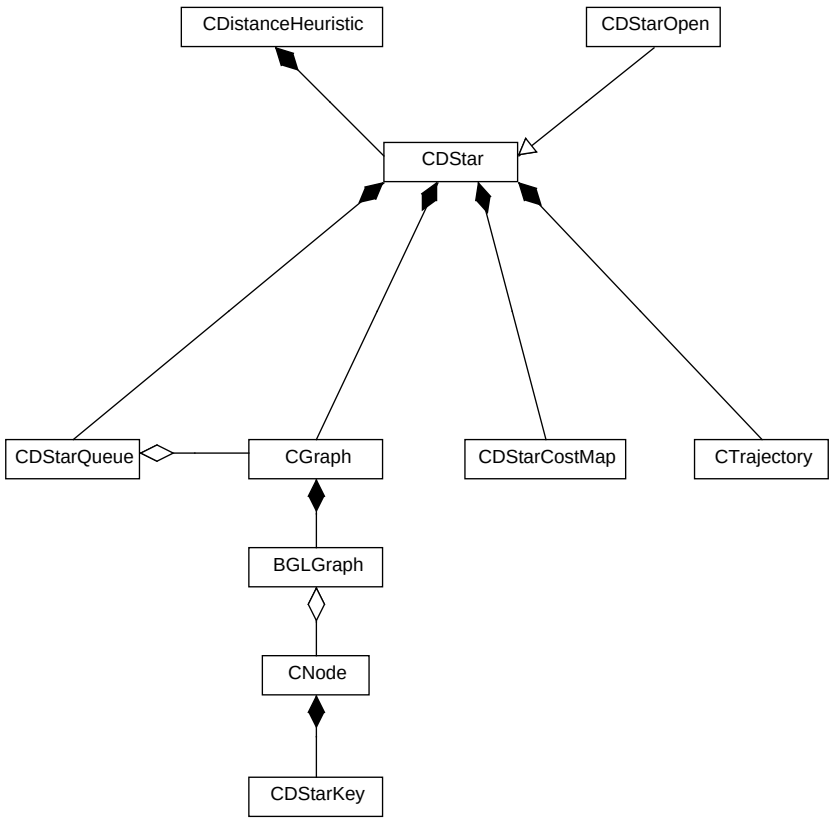


Figure 3.2: Class Diagram of the D* Library

algorithm via a pointer to the CDStar class. The operator returns the estimate of the distance to the start node for a given node.

CDStarOpen inherits from the DStar class and is used in conjunction with the standalone D* viewer. It implements extra functionality only required in a standalone context - such as methods to extract scaled costmaps from the planner.

Two different variations of D* Lite have been implemented. The first is the basic D* Lite version of Figure 2.15, the second is the optimized version contained in Appendix A, Figure A.1. The optimized version relies on the use of an updateable priority queue and requires the old cost of a changed cell to be stored.

In the vanilla implementation we have used a binary heap (created out of a vector using the STL make_heap function). In the optimized version we use a Fibonacci heap which makes the update operation more efficient as it supports a decrease-key

Type	Insert	Remove	Peek	Increase Priority
Binary Heap	$O(\log n)$	$O(\log n)$	$O(n)$	n/a
Fibonacci Heap	$O(n)$	$O(\log n)$	$O(n)$	$O(n)$

operation in (amortized) constant time. In the grid search applications D* Lite is tailored for there is some benefit in opting for a Fibonacci heap over a binary heap as the underlying graph is both large and dense. The LEDA (Mehlhorn and Naher, 2010) priority queue library which implements a priority queue using a Fibonacci heap was utilized.

Algorithmically the two versions differ slightly, the optimized version takes more lines of code to implement but introduces special cases which lead to unnecessary computation being eliminated.

3.1.2 Planning Processes

pDStarPlanner The pDStarPlanner process operates in conjunction with a local path planner to produce motion commands for the robot. pDStarPlanner contains an instance of the CDStar class and provides the functionality to read the robot's current position and translate this into an update of the start position for the DStar planner and trigger an update to the plan. Its main functionality is in an iterate loop. When the planner is offline it sits there reading costmap data and updating DStar's local cost map. While the robot is actively transiting along the path it publishes the path as well. The pDStarPlanner process actively reads the navigation data, relates it to the closest point on the global path, updates the start position accordingly and triggers a replan if necessary.

pPilot acts as a harness process linking the two operations. It contains instances of a local path generator which plans splines to the nearest point on the global path from the robot's current position. It also contains an instance of a path controller which modulates progress along the local path. The pPilot process sends actuation

messages to set the speed and heading of the robot.

3.2 The Costmap library and Costmap Processes

The costmap library is an integral part of the planning toolkit. Without a map of the environment planning is impossible. Our costmap library provides a means of taking information from either 2D laser sensors or stereo cameras - but the methods used are extensible to other sensors.

The costmap library supports the construction of two different environmental representations - occupancy grids and costmaps.

3.2.1 Occupancy Grids

Occupancy grids (Elfes, 1987, 1989; Moravec and Elfes, 1985) are used to generate consistent maps of obstacles in the environment from sensor measurements that are noisy and uncertain. It is a probabilistic technique that relies on the robot pose being known - hence it requires a navigation capability to have already solved the Simultaneous Localization and Mapping (SLAM) problem; which we will discuss in Section 3.3.1. While SLAM algorithms by necessity maintain some kind of map of the environment, the often sparse combination of features and/or poses that constitute a SLAM map is generally not suited to creating costmaps for path planning applications. Hence occupancy grid mapping is commonly used to post-process observational data into a useful map using the results of a navigation algorithm.

As in grid-based path planning the environment is discretized into an ordered, evenly-spaced grid. However each grid cell is now represented by a random variable m_i that has two possible states, occupied ($p(\mathbf{m}_i) = 1$) or free ($p(\mathbf{m}_i) = 0$). The challenge is to estimate the posterior probability $p(m \mid \mathbf{z}^t, x^t)$ of the map m given the data $\{\mathbf{z}^t, x^t\}$, where $\mathbf{z}^t = \{\mathbf{z}_1, \mathbf{z}_2 \ldots \mathbf{z}_t\}$ is the set of measurements taken at times 1 to

t, and $x^t = \{x_1, x_2 \dots x_t\}$ is the set of robot poses. Figure 3.3 provides a graphical model of the estimation problem.

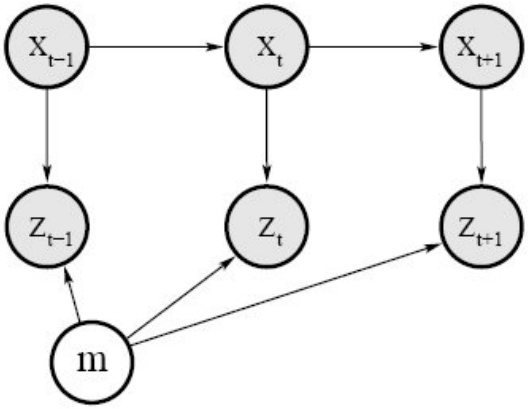


Figure 3.3: Graphical Model of the occupancy grid building Process. Shaded nodes represent observed variables ($x_{t-1:t+1}$ and $z_{t-1:t+1}$), unshaded nodes represent hidden variables whose values we want to estimate (the map m). Arrows in the Graphical Model represent causal relationships between variables. Image taken from Thrun et al. (2005).

The problem with estimating the posterior over the map, $p(m \mid z^t, x^t)$ is its dimensionality. As each grid cell in the map can be either free or occupied, the number of possible maps for a grid of n cells is 2^n. Computing the posterior probability for each possible map quickly becomes intractable when you consider that grid cells in planning generally have a resolution in the tens of centimetres, leading to maps with hundreds, if not thousands of grid cells.

Hence the standard approach to occupancy grid mapping is to factorize the posterior by (conveniently, but not necessarily honestly) assuming that the occupancy of a grid cell is independent of the state of its neighbouring cells.

$$p(m|z^t, x^t) = \prod_i p(\mathbf{m}_i \mid z^t, x^t) \tag{3.1}$$

So the problem now boils down to estimating the occupancy (a binary problem) of n separate cells. A *binary Bayes filter* can be used to do this estimation for each cell. As we assume the map is static (i.e. we are not dealing with any dynamic obstacles) we can omit the action update step typical of a Bayes filter and need only apply the

measurement update. Concretely, we derive the filter here. If the odds of a state x is defined as

$$\frac{p(x)}{p(\neg x)} = \frac{p(x)}{1 - p(x)}, \tag{3.2}$$

then we can define the log-odds ratio, which allows a more computationally elegant implementation of the filter and avoids truncation difficulties that would otherwise arise when dealing with probabilities close to 0 or 1.

$$l_{t,i} = \log \frac{p(\mathbf{m}_i \mid \mathbf{z}^t, x^t)}{1 - p(\mathbf{m}_i \mid \mathbf{z}^t, x^t)} \tag{3.3}$$

By rearranging the log-odds ratio we can retrieve the belief over the grid cell's state:

$$p(x) = 1 - \frac{1}{1 + \exp(l_{t,i})}. \tag{3.4}$$

We wish to estimate the posterior at the grid cell:

$$p(\mathbf{m}_i \mid \mathbf{z}^t, x^t) = p(\mathbf{m}_i \mid \mathbf{z}_t, \mathbf{z}^{t-1}, x^t). \tag{3.5}$$

We assume the state of the grid cell $p(\mathbf{m}_i)$ is static, so that the belief is a function of only the measurements and not the robot's position, such that $p(\mathbf{m}_i \mid \mathbf{z}^t, x^t) = p(\mathbf{m}_i \mid \mathbf{z}^t)$.

Bayes rule allows us to expand this expression as:

$$p(\mathbf{m}_i \mid \mathbf{z}^t) = \frac{p(\mathbf{z}_t \mid \mathbf{m}_i, \mathbf{z}^{t-1}) p(\mathbf{m}_i \mid \mathbf{z}^{t-1})}{p(\mathbf{z}_t | \mathbf{z}^{t-1})} \tag{3.6}$$

$$= \frac{p(\mathbf{z}_t \mid \mathbf{m}_i) p(\mathbf{m}_i \mid \mathbf{z}^{t-1})}{p(\mathbf{z}_t \mid \mathbf{z}^{t-1})}, \tag{3.7}$$

and then apply Bayes Rule to the measurement model $p(\mathbf{z}_t \mid \mathbf{m}_i)$:

$$p(\mathbf{z}_t \mid \mathbf{m}_i) = \frac{p(\mathbf{m}_i \mid \mathbf{z}_t) p(z_t)}{p(\mathbf{m}_i)}, \tag{3.8}$$

to obtain

$$p(\mathbf{m}_i \mid z^t) = \frac{p(\mathbf{m}_i \mid z_t)p(z_t)p(\mathbf{m}_i \mid z^{t-1})}{p(\mathbf{m}_i)p(z_t \mid z^{t-1})}. \tag{3.9}$$

Writing this equation for the opposite belief case ($\neg\mathbf{m}_i$):

$$p(\neg\mathbf{m}_i \mid z^t) = \frac{p(\neg\mathbf{m}_i \mid z_t)p(z_t)p(\neg\mathbf{m}_i \mid z^{t-1})}{p(\neg\mathbf{m}_i)p(z_t \mid z^{t-1})}, \tag{3.10}$$

and dividing equation 3.9 by 3.10 produces the odds ratio:

$$\frac{p(\mathbf{m}_i \mid z^t)}{p(\neg\mathbf{m}_i \mid z^t)} = \frac{p(\mathbf{m}_i \mid z_t)p(\mathbf{m}_i \mid z^{t-1})p(\neg\mathbf{m}_i)}{p(\neg\mathbf{m}_i \mid z_t)p(\neg\mathbf{m}_i \mid z^{t-1})p(\mathbf{m}_i)} \tag{3.11}$$

$$= \frac{p(\mathbf{m}_i \mid z_t)}{1 - p(\mathbf{m}_i|z_t)} \frac{p(\mathbf{m}_i \mid z^{t-1})}{1 - p(\mathbf{m}_i \mid z^{t-1})} \frac{1 - p(\mathbf{m}_i)}{p(\mathbf{m}_i)}. \tag{3.12}$$

The log odds ratio is thus written:

$$l_t(\mathbf{m}_i) = \log \underbrace{\frac{p(\mathbf{m}_i \mid z_t)}{1 - p(\mathbf{m}_i \mid z_t)}}_{\text{inverse sensor model}} + \underbrace{l_{t-1}(\mathbf{m}_i)}_{\text{previous log odds ratio}} - \log \underbrace{\frac{p(\mathbf{m}_i)}{1 - p(\mathbf{m}_i)}}_{\text{prior}}. \tag{3.13}$$

We use a log-odds representation of the probability of occupancy in the binary filter of Figure 3.4.

```
OccupancyGridMapping({l_{t-1,i}}, x_t, z_t)
1  for all cells m_i
       do
2      if m_i in perceptual field of z_t
3        then l_{t,i} = l_{t-1,i} + InverseSensorModel(m_i, x_t, z_t) − l_0
4        else
5             l_{t,i} = l_{t-1,i}
6             return {l_{t,i}}
```

Figure 3.4: Occupancy grid mapping with a binary Bayes filter. As per Thrun et al. (2005).

The algorithm implemented on the MRG robotics platforms, is as per Thrun et al.

(2005) and with pseudocode given by Figure 3.4. It loops through the cells currently in the sensor field of the robot and updates the probability of sighted cells accordingly. All computations are done in log space and the log-odds ratio of the prior is usually set to zero, representing maximum uncertainty about the state of the world going in.

Note that a crucial part of the binary Bayes filter update is the computation of the inverse sensor model - the probability of the map given the sensor readings. Different sensing modalities necessitate different inverse sensor models. Figure 3.5 details the inverse sensor model that has been implemented for MRG. It is tightly coupled with the binary Bayesian filter update in our implementation. While originally built to work with the 2D laser scanner, the inverse sensor model can be used to accept data from the stereo camera system by converting 3D points into points into the 2D plane. The inverse sensor model works by ray tracing. Data points from the laser (or the converted stereo data) represent obstacles. The data points themselves correspond to the inverse sensor model returning a value of l_{occ}, and any points that lie along a line drawn between that data point and the robot must therefore be free space and are assigned l_{free}.

We borrow the concept of the *Pose Bucket* from Konolige (1997), as without keeping track of how many times the robot has made a particular observation from the same spot there lies the potential to become falsely 'certain' about the measurements. The free and obstacle pose buckets implement map data structures indexed by the grid cell location, which store unique robot poses which have resulted in free/blocked observations respectively. Before updating the log-odds ratio we first check if the occupied/free observation of this cell has been made from this robot location before. If it has, the update does not take place.

```
InverseSensorModel(m_i, x_t, z_t)
 1  find the number of cells that the laser beam from {x_t, z_t} passes through
 2  for each cell
 3       do
 4          if it is the end point of the beam
 5             then if we have not observed an occupied value for this cell from this
                     pose before AND the cell value is less than the Occupied Threshold
 6                     then add this pose and occupied cell pair to the OccupiedBucket
 7                          return l_occ
 8                     else  return l_0
 9             else
10                   if we have not observed a free value for this cell from this pose before
                     AND the cell value is less than the Free Threshold
11                     then add this pose and free cell pair to the FreeBucket
12                          return l_free
13                     else  return l_0
```

Figure 3.5: The inverse sensor model is based on Ray Tracing and uses Pose Buckets to avoid becoming 'too certain' whilst the robot takes stationary measurements.

3.2.2 Costmaps

Costmaps are constructed over the same 2D grid as occupancy grids. The difference is that while occupancy grids encode the likelihood of the presence of an obstacle at a particular grid cell, cost maps represent the traversability of that cell for the robot. This is assumed to be somewhere on the scale from 'Free' to 'Lethal'. In an indoor environment it is possible to navigate using only the results of an occupancy grid. Typically in indoor environments any obstacles detected are rigid and static and attempting to drive over or through them would be dangerous for the robot. Outdoors however, the environment is more cluttered and there are obstacles (small rocks, tall grass) which the robot could drive over.

Creating useful costmaps relies on the ability to classify the environment into a number of distinct terrain types, for example bitumen, grass, dirt path, trees, cars etc. This classification relies on being able to fuse together multi-spectral information such as texture obtained from laser data, colour from imagery, and infra-red data amongst

many other possibilities. The costmap library provides an extensible framework to combine the results of classification and/or obstacle detection from these different sensors together.

3.2.3 Costmap Processes

pOccupancyGridBuilder builds a representation of the probability that each grid cell in the map is occupied. It subscribes to data from the stereo camera or 2D laser and invokes an instance of COccupancyGrid to store and update the occupancy grid. It publishes the data for consumption by pCostmapBuilder.

pPathDetector performs image segmentation and is capable of discriminating between terrain types and identifying drivable terrain. It uses data from a monocular camera and publishes the classified data in grid format suitable for consumption by pCostmapBuilder.

pCostmapBuilder combines information from both sources to produce cost maps with grid cells assigned different values according to terrain type and predicted occupancy. It is extensible so could easily fuse data from more sources in future.

A number of different viewers have been written to visualize the costmap, occupancy grid and output from the planner.

uOccupancyGridViewer and **uCostmapViewer** are OpenGL based viewers which provide 2D visualization. **uOccupancyGridviewer3D** uses the FLTK library's (FLTK, 2010) OpenGL implementation. It provides an integrated viewer that can swap between displaying occupancy grids and costmaps, a 3D view of terrain (with the ability to swap to 2D) and dynamic display of the robot's path. A sample screenshot is shown in Figure 3.6.

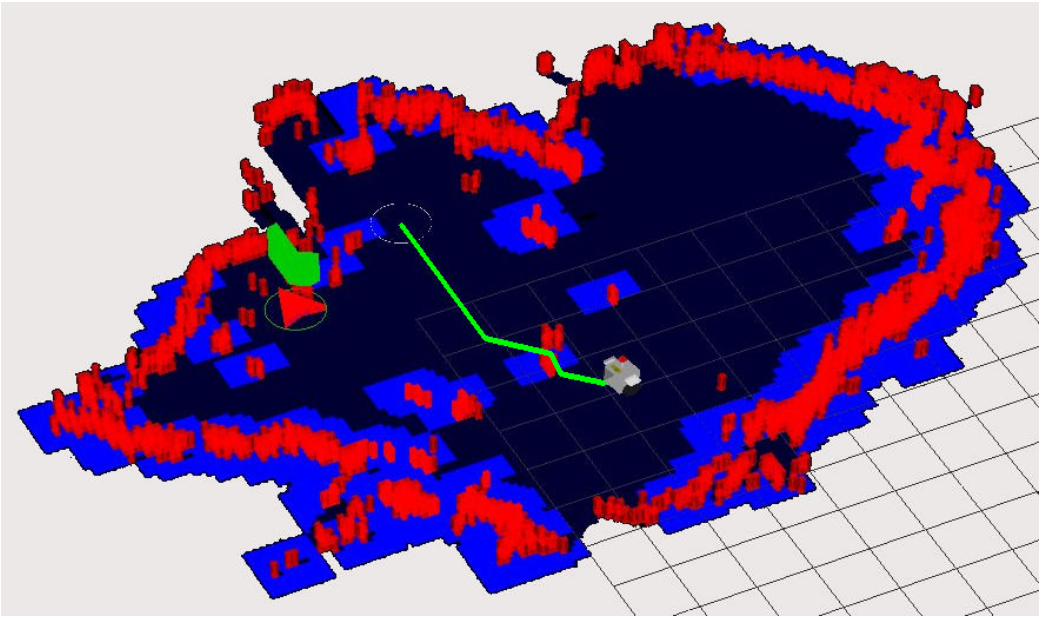


Figure 3.6: Screenshot of the uOccupancyGridViewer3D process showing the costmap of the basement lab and a planned path (green line). The downward pointing green arrow indicates a new goal location for the robot that the user has yet to send to the planner.

3.3 Navigation

In creating occupancy grids and costmaps, we are performing the task of mapping where the robot position is known. But how do we know what that position is? As we have highlighted repeatedly, the robot's sensors and actuators are noisy, so we cannot simply rely on raw odometry for an accurate estimate of our position. Yet when operating in an a-priori unknown environment, the only information the robot has available to it to discern its location is the map it has built by gathering information from its sensors into a consistent model. If that model is built based on an incorrect assumption of the robot's position, then the mapping errors will quickly escalate and the robot will very soon find itself lost. However, if both the assumption of the robot's position and the map it is building take into account the uncertainty of the sensors and actuators, it is possible to build a coherent map with which to navigate from. Simultaneous Localization and Mapping is the name given to the probabilistic technique used by robots to build a map within an unknown environment, and to

localize (keep track of its current position) within that map. As covered in Chapter 1, the desire for true robot autonomy underpins this work, so we discuss SLAM as a technique for self-localization rather than rely on an external system such as GPS.

Throughout this thesis we take the capability to navigate as a given. We make this assumption based on the maturity of the SLAM problem and the availability of tools (the processes **pSMSLAM** and **pNav**) within the Mobile Robotics Group which have proven to provide reliable data to our path planning and costmap creation tools. However, some understanding of the formulation of the SLAM problem is required for the next chapter, where exploration techniques are presented which draw upon SLAM. We include a brief overview of the problem here to make a natural progression from our treatment of mapping in the previous section to the problem of mapping with unknown position.

3.3.1 SLAM

SLAM is a paradoxical problem wherein a robot is tasked to map an unknown region (which requires accurate knowledge of its current location) and yet also know where it is currently located (which requires a known map). It is generally a passive problem, the focus of the majority of research remains on acquiring an accurate map and hence action selection is so far almost always performed by a human. The robot is only provided with sensor measurements $z^t = \{z_1, z_2, ..., z_t\}$ and motion controls $u^t = \{u_1, u_2, ...u_t\}$ relative to its own pose (location) which arrive over time. The subscripts refer to measurements and controls at discrete time indexes $\{t_1, t_2, ...t_t\}$. The problem stems from the fact that as the robot explores the unknown region, it accumulates error in its pose estimate and subsequently the acquired map becomes increasingly inaccurate, propagating the error in the pose estimate.

In probabilistic form, the SLAM problem is commonly represented in one of two ways. The *full SLAM* problem involves estimating the joint posterior distribution

over the entire trajectory of the robot (the set of poses $x^t = x_{1:t}$) and the map m,

$$p(x^t, m \mid z^t, u^t) \tag{3.14}$$

given the set of all observations z^t and control inputs u^t up to time t. No time index is associated with the map m as most approaches drop the distinction between the (static) physical world and the map. Of course, our knowledge of m can increase with time but as the underlying environment has not changed it has no bearing on the relationship between the estimate of the earlier poses, map and observations and controls.

The online problem estimates the posterior over only the last pose and the map, and is the result of integrating out past poses $x_1, x_2 ... x_{t-1}$ from the full SLAM problem.

$$p(x_t, m|z^t, u^t) = \int \int \dots \int p(x^t, m \mid z^t, u^t) dx_1 dx_2 \dots dx_{t-1} \tag{3.15}$$

The difference between these two formulations of the problem is reflected in the type of algorithms that can be applied to solving the problem.

Calculating the true posterior in both Equation 3.14 and Equation 3.15 is practically infeasible, because it is a distribution over continuous space and has extremely high dimensionality given that maps contain thousands of features each represented in two or three dimensional space. Practical SLAM implementations make assumptions which ease the computation of the problem, often by imposing specific parameterizations (e.g. Gaussian) on the involved probability density functions. It is these assumptions coupled with varying choices for the underlying map representation (such as landmark parameterizations or occupancy grid maps) that further differentiate the various approaches to SLAM which have arisen as a result of two decades of research into the problem.

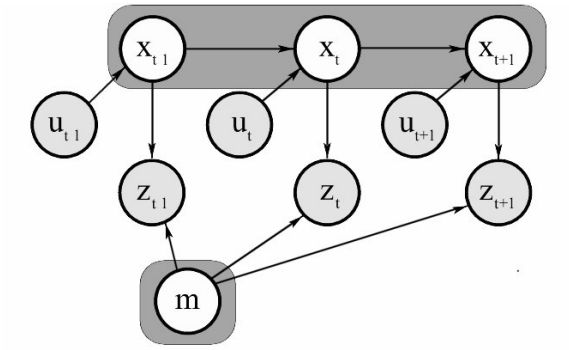


Figure 3.7: Full SLAM (from Thrun et al. (2005)), showing that the joint posterior (variables contained in the dark shaded oblongs) we seek is computed over the whole trajectory of the robot and the map. Observed variables are shaded grey circles, and those variables that are hidden and we wish to estimate (x_t and m) are represented by white circles. Arrows in the graphical model represent causal relationships between variables.

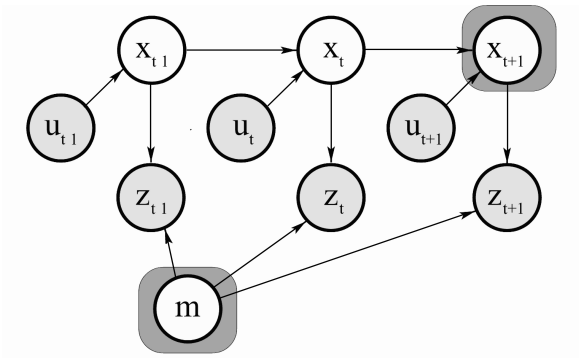


Figure 3.8: Online SLAM from(Thrun et al. (2005)), the joint posterior (shaded) is computed over the last robot pose and the map. Note the contents of the shaded region compared with Figure 3.7 is now reduced to just the circles containing m and x_{t+1}

3.3.2 Single Vehicle SLAM Formulations

EKF The EKF SLAM algorithm applies the Extended Kalman Filter (EKF) (Welch and Bishop, 2006) to the online SLAM problem. This approach was pioneered by Smith, Self and Cheeseman (Smith et al., 1987) who introduced the feature-based *"stochastic map"* representing observable landmarks and the relationships between them by geometric primitives parameterized by their mean and covariance. Many others (Thrun et al., 2005; Thrun, 2002; Dissanayake et al., 2001) have since developed this approach further and it remains in widespread use.

The EKF implementation is derived from the Bayes filter representation of the

online SLAM problem.

$$p(x_t, m|z^t, u^t) = \eta p(z_t|x_t, m) \int p(x_t|u_t, x_{t-1}) p(x_{t-1}|z^{t-1}, u^{t-1}) dx_{t-1} \tag{3.16}$$

Equation 3.16 is a function of two generative models of the robot and environment. The term $p(x_t|u_t, x_{t-1})$ is the *motion model* relating the vehicle state to the proceeding state and the applied control. The state transition is assumed to be a Markov process as the new state depends only on the preceding state and the applied control. It is independent of the map and the observations. The *observation model* is given by the $p(z_t|x_t, m)$ term and describes the distribution of an observation z_t given different poses x_t and maps m. Both the motion and observation models are usually assumed to be time invariant.

Equation 3.16 can be viewed as a standard predictor-corrector formulation consisting of a *time update*

$$p(x_t, m|z^{t-1}, u^t) = \int p(x_t|x_{t-1}, u_t) p(x_{t-1}, m|z^{t-1}, u^{t-1}) dx_{t-1} \tag{3.17}$$

and a *measurement update*

$$p(x_t, m|z^t, u^t) = \frac{p(z_t|x_t, m) p(x_t, m|z^{t-1}, u^t)}{p(z_t|z^{t-1}, u^t)}. \tag{3.18}$$

This formulation provides a recursive procedure for calculating the vehicle pose and map based on the set of observations and controls up to time t, that is a function of the measurement and motion models, suitable for implementation using a Kalman filter.

The EKF formulation represents the joint posterior probability density function of Equation 3.16 as a Gaussian of mean $\hat{x}$ and covariance P and assumes both the motion model and the observation model are linear with Gaussian noise. This requires

approximating both models with linear functions obtained through a Taylor-series expansion.

The motion model is expressed as

$$p(x_t|x_{t-1}, u_t) \leftrightarrow x_t = \mathcal{N}(f(x_{t-1}, u_t), Q) \tag{3.19}$$

where $f(\cdot)$ models the motion of the vehicle with applied control u_t. Q is the covariance term.

The observation model becomes

$$p(z_t|x_t, m) \leftrightarrow z_t = \mathcal{N}(h(x_t, m), R) \tag{3.20}$$

where $h(\cdot)$ relates the observations to the current state and the model has covariance R.

The EKF equations expressed as a time update and measurement update are:
Time Update

$$\hat{x}_{t|t-1} = f(\hat{x}_{t-1|t-1}, u_t) \tag{3.21}$$

$$P_{t|t-1} = \nabla f P_{t|t-1} \nabla f^T + Q_t \tag{3.22}$$

∇f is the Jacobian of f evaluated at the current estimate $\hat{x}_{t-1|t-1}$.
Measurement Update

$$\hat{x}_{t|t} = \hat{x}_{t|t-1} + W_t \upsilon \tag{3.23}$$

$$P_{t|t} = P_{t|t-1} - W_t S_t W_t^T \tag{3.24}$$

where the innovation

$$\upsilon = z(t) - h(\hat{x}_{t|t-1})$$

has covariance

$$S_t = \nabla h P_{t|t-1} \nabla h^T + R_t$$

and W_t is the Kalman Gain

$$W_t = P_{t|t-1} \nabla h^T S_t^{-1}$$

and ∇h is the Jacobian of h evaluated at $\hat{x}_{t|t-1}$.

Although the EKF remains a popular SLAM formulation, it is hampered by issues of quadratic complexity in the update, its fragility when faced with incorrect data association in detecting features, and inconsistency in solutions as a result of the linearization of the typically non-linear motion and observation models.

Information Filter The Information Filter results from the canonical representation of the Gaussian distribution, wherein the distribution is parameterized by an information vector $\eta = \Sigma^{-1}\mu$ and information matrix $\Lambda = \Sigma^{-1}$ instead of its mean and covariance. It is commonly referred to as the dual of the EKF as the difference in parameterization leads to different update equations. The Information Filter offers significant performance gains when used in conjunction with the full SLAM formulation as discussed in Eustice et al. (2006) due to the sparse, highly diagonal structure of the information matrix which occurs as a result of retaining the robot trajectory (rather than marginalizing it out as occurs in the feature-based SLAM formulation).

The sparsity in the information matrix is a direct result of the assumption that the evolution of the robot state occurs as result of a first-order Markov process. When a new state is added to the existing representation, the posterior we wish to estimate becomes

$$\begin{aligned} p(x_{t+1}, x_t, M | z^t, u^{t+1}) &= p(x_{t+1} | x_t, M, z^t, u^{t+1}) p(x_t, M | z^t, u^{t+1}) \\ &= p(x_{t+1} | x_t, u_{t+1}) p(x_t, M | z^t, u^t). \end{aligned}$$

In the past, a common approach to SLAM was to use an implementation of the online SLAM model in conjunction with feature-based maps consisting of landmarks

(distinct, recognizable features) and their associated uncertainties. A landmark's position is conditionally independent of other landmark positions given the current robot position. However, the online SLAM model requires maintaining a representation over only the current robot pose, which necessitates *marginalizing* out the past robot poses. This forces the landmark estimates to become correlated with each other. This process, and its effect on the information matrix, is illustrated by the graphical model of Figure 3.9.

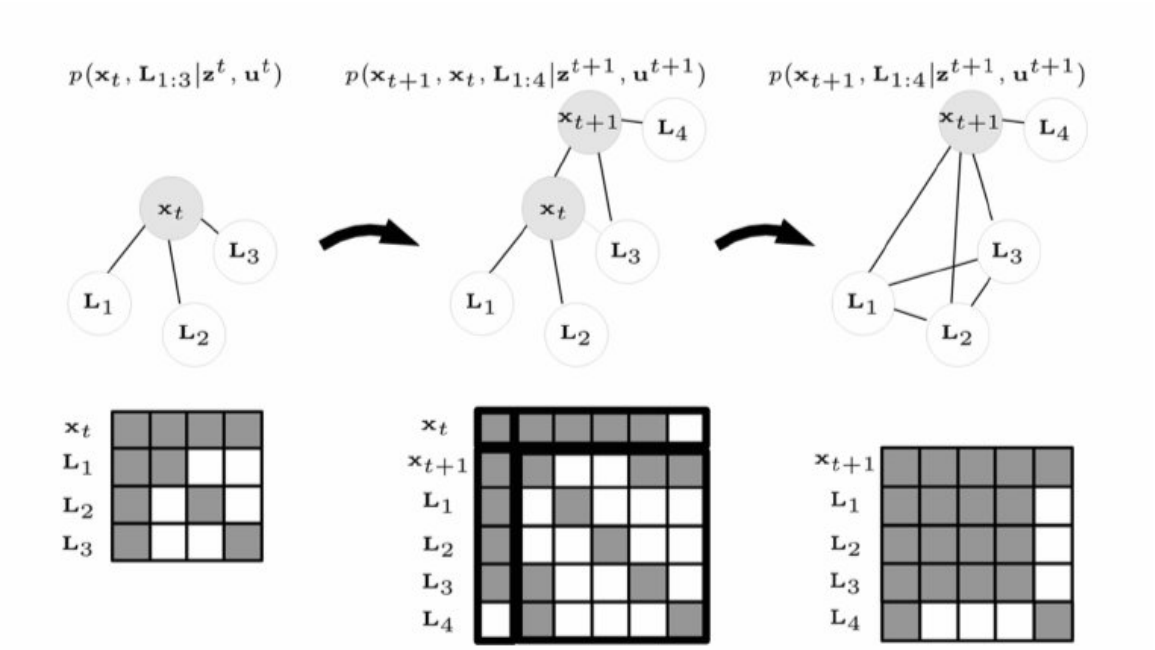


Figure 3.9: Updating a feature-based SLAM information matrix (Eustice et al., 2006). The original matrix at time t is shown at left. The information matrix is shaded to demonstrate that the vehicle pose is correlated with all three landmarks, but the landmarks are uncorrelated. The middle figure shows the effect of the time propagation of the estimate, the matrix is augmented with the robot pose at time $t+1$ and associated dependencies are shaded. The figure at right shows the *fill in* effect of marginalizing out the robot pose at t. The original landmarks, L_1, L_2 and L_3 are now correlated. This illustrates how the Markov network of the landmarks (map) becomes fully connected, and the corresponding feature-based information matrix is fully dense. Figure from Eustice et al. (2006).

In a view-based full SLAM approach (shown in Figure 3.10), maps represent the world via a series of past robot poses (the robot's trajectory) and the associated uncertainty in the poses. This representation was introduced in the seminal paper by Lu and Milios (1997). Associated with each pose is a snapshot of the portion

of the world observed at that point. Typically, this data is either visual (from a camera) or a point cloud taken from a range scanning device such as a laser scanner. Constraints, which geometrically relate poses to each other, can be obtained by finding the transformation between identical regions captured by successive poses.

Retaining the full robot trajectory means that the conditional independence of the new state and the map is maintained. The shared information between the new state and the map is always zero, meaning the information matrix takes a block tri-diagonal structure which allows motion prediction and measurement updates to be performed on the filter in time linear in the number of states.



Figure 3.10: A view based SLAM Map. Red triangles denote robot poses. The 2D Laser point map shown is the union of scans taken from all poses.

3.4 Chapter Summary

This chapter has focused on the implementation of the state of the art algorithms in path planning, costmap creation and navigation. We addressed techniques in occupancy grid creation and in navigation which take the uncertainty in the robot's world knowledge into account. We were unable to present analogous techniques for

path planning and cost map creation; the computational expense of planning optimal paths through grids means research to date has focused on deterministic approximation algorithms that are extremely fast. What these algorithms fail to do is take into account the possibility that the underlying assumptions are incorrect, and they disregard vital information in the process. In chapters to follow, we will augment these fast planning algorithms with techniques to deal with uncertainty.

Before we present our probabilistic approach to path planning and cost map creation, we first visit the problem of generating goals to feed to the path planner. The exploration technique we will present is tightly coupled with the SLAM navigation capability presented in the concluding stages of this chapter.

Chapter 4

Exploration

Of the three competencies we will examine in this thesis (path planning, navigation and exploration) - it is exploration that poses the most open, unanswered questions. A Google search on 'Robotic Exploration' turns up many examples of robotic systems in which humans, rather than the robot itself, make the crucial decision on where to go next. Yet as robotic hardware progresses, the challenge of true autonomy grows with it. In areas such as planetary exploration, oceanography and urban search and rescue, technology has progressed to the point where robots can navigate successfully for many kilometers in a single command cycle. Equipping the robot with the ability to plan its information gathering in an intelligent way so as to service the competing demands of visiting new territory; revisiting covered ground to aid navigation; and completing assigned tasks. All of the above falls under the banner of exploration, which remains one of the fundamental problems in robotics.

In this chapter we address the problem of exploration, specifically how to motivate a single robot system acting under uncertainty to explore its environment in a structured, deterministic manner. As humans, we explore by seeking out unseen territory. We look for doorways, corners and obscured regions and seek to find out what lies behind them. In this chapter we apply this intuitive approach to exploration to the robotics domain using an algorithm called the Gap Navigation Tree.

4.1 Motivation

The Gap Navigation Tree algorithm (GNT) was introduced by Lavalle and Tovar (LaValle, 2006; Tovar et al., 2004, 2007) and has been shown to produce optimal navigation paths in unknown planar environments. At its heart is a tree data structure which stores the 'gaps' in the environment in a systematic way as leaves of a tree. In so doing, it constructs a topological map that encodes a globally optimal path (in a Euclidean sense) from the robot's current location to any point of interest. Underpinning the algorithm is the assumption of the availability of an abstract *gap sensor*, an ideal infinite range sensor able to track and uniquely identify discontinuities in the depth of the range information at the robot's current viewpoint. These discontinuities are of interest as they correspond to the 'gaps' - regions not yet visible to the robot. While the algorithm's authors addressed its provability and correctness in 2D planar worlds, they did not address the issue of how to construct a robust gap sensor; nor provide experimental evidence of it exploring in a persistent fashion in a real-world setting.

Evidently, the GNT algorithm makes some strong assumptions stemming from its planar world origins and these must be addressed before the algorithm can be implemented on a real robot. The primary difficulty lies in the constructing a real world analogue of the ideal infinite range gap sensor. A secondary complication is that any 'events' concerning the appearance, disappearance, splitting or merging of the gaps are automatically encoded into the GNT, meaning a single error in the gap sensor can easily corrupt the topological map. Both of these issues are addressed in this chapter.

We have chosen to couple the GNT algorithm with an existing SLAM system (detailed previously in Chapter 3), to drive exploration and map making. In the text that follows, we detail a real world implementation of the gap sensor, and use the availability of a metric map from the SLAM system to add robustness to the gap

sensor by tracking the location of the gaps probabilistically. Experimental results from simulations are presented, showing that this approach results in coverage of indoor environments given a robust implementation of a gap sensor based on data from a 2D Laser scanner - certainly not the 'ideal infinite resolution sensor' required by the original algorithm.

Before discussing our approach to exploration using the GNT, Sections 4.2 to 4.3 first position the algorithm amongst existing approaches to exploration and provide justification for choosing it as a suitable driver for exploration. Original contributions are described in Section 4.5 onwards.

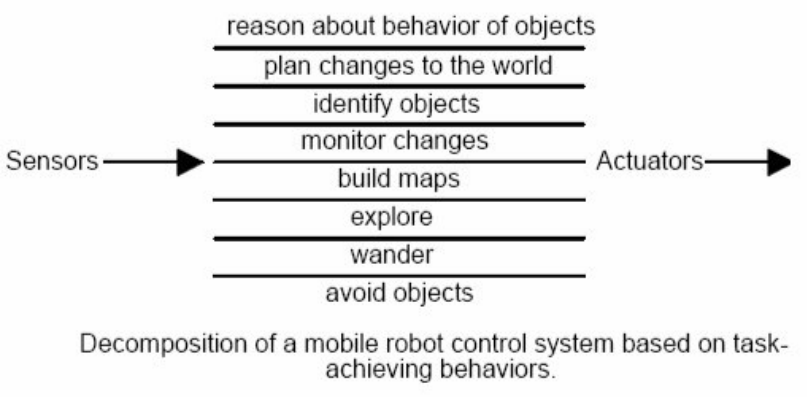
4.2 Background to the Exploration Problem

Exploration occurs when a robot is placed in an unknown environment and asked to construct a map which can be used for subsequent navigation as it moves through the world. The ability to answer the question of *"where to go next?"* informed only by data contained in a partially complete map, is key to the development of truly autonomous mobile robots.

4.2.1 Exploring by bumping into things - Early Work

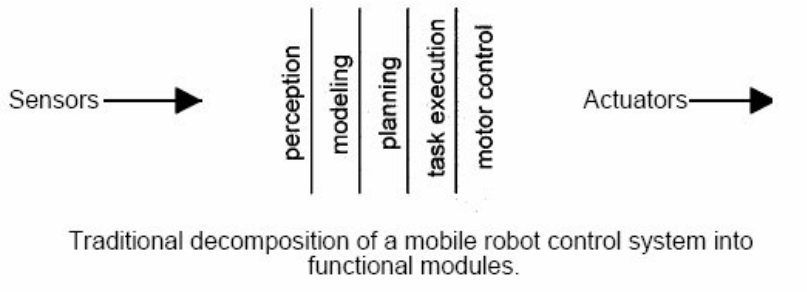
Robot autonomy has been the focus of research for almost 50 years. Between 1966-1972 work at the Stanford Robotics Institute lead to the development of *Shakey* (Nilsson, 1984), the first mobile robot to visually interpret and interact with its surroundings. Equipped with a world representation based on formal logic sentences and sensors including a TV camera, Shakey was able to reason about the surrounding world to accomplish simple planning and route finding tasks to enable it to move objects around a room. It could accept general instructions over a radio link and autonomously break down the instructions into smaller steps to accomplish simple

tasks. Its logic-based world representation was brittle however (Moravec, 1987), and Shakey often required some human intervention to complete tasks.



Decomposition of a mobile robot control system based on task-achieving behaviors.

(a) Behavioural



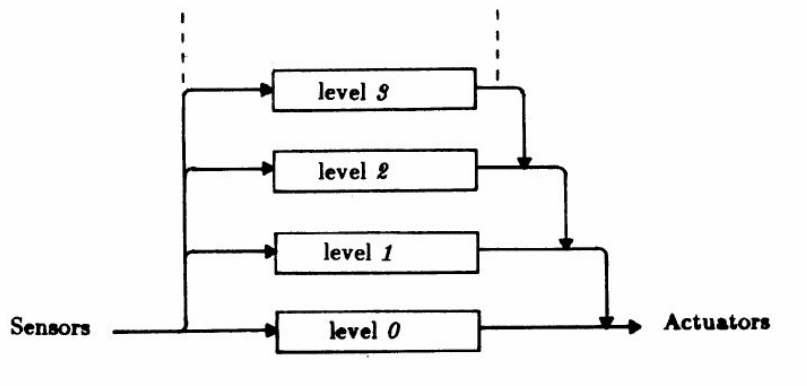
Traditional decomposition of a mobile robot control system into functional modules.

(b) Functional

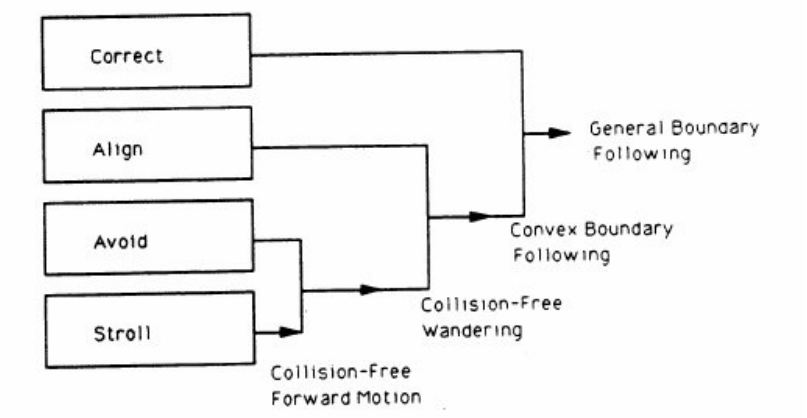
Figure 4.1: The difference between a traditional functional control system and the Behaviour Based Control architecture of Brooks (1986).

Brooks (1986) deemed that the problem with mobile robot control systems such as Shakey's lies in the surfeit of processes separating sensing and eventual action. The subsumption architecture of Figure 4.2(a) consists of lightweight parallel modules allowing for tight coupling between sensors and actuators. Each module implemented a specific behaviour, and these were arranged in a hierarchy of increasingly sophisticated tasks (see Figure 4.2(b)) each with the capability to inhibit outputs of and suppress inputs to other behaviours. This architecture was successfully implemented on Toto (Mataric and Brooks, 1990), an autonomous sonar based mobile robot which performed wall following around a static office environment at MIT. Toto did this without constructing an explicit geometric map. Instead it navigated by reasoning

about its own activity - learning and storing characteristic repeated behaviours in a simple list structure which higher level behaviours were able to use to inform their actions. If Toto got lost and could not recognize a particular pattern of sensory input, it would default to the lowest level of control (wall following), until subsequent inputs allowed it to relocate itself and reactivate higher level behaviours.



(a) Subsumption Architecture (Brooks, 1986).



(b) Toto's Architecture (Mataric and Brooks, 1990).

Figure 4.2: The Subsumption Architecture and an illustration of how it was used to develop wall following behaviour on Toto, a mobile robot at MIT.

The early subsumption architecture produced purely reactive agents such as Toto which did not maintain any world model or map. They were fast to respond to changes in the environment and built-to-task through the implementation of the behaviour modules. However, waiting on the world to force the robot into action is not always desirable, it is preferable to combine reactive techniques with some form of

deliberative reasoning about what to do based on state information. The coupling of these two philosophies of robot control has lead to the now predominant hybrid deliberative-reactive approach (Arkin and Balch, 1997; Gat, 1992; Hawes et al., 2009; Konolige et al., 1997; Murphy, 1998).

In sections to follow deliberative components of this approach, which use the information in the content of a partially complete map as a basis for motivating further exploration, are discussed.

4.2.2 Exploring by making maps

While reactive architectures may be adequate for accomplishing simple tasks in structured indoor environments, the true driver for work in robotics is to develop robots that can do the tasks that are too dangerous, inefficient or expensive for humans to do. These tasks, such as mining; exploring other planets or the ocean floor; searching for and rescuing survivors from fire or earthquake damaged buildings; and performing repetitive and intricate manufacturing processes, require repeatability. Imagine trying to coordinate a mining operation with multiple robotic vehicles all interacting in a continually changing environment - this is simply not possible if the robot has to relearn everything about its environment en-route to accomplishing each new task.

Hence the most efficient way to equip a robot to perform a task is to provide it with a map, and exploration can then be driven by the wish to expand the boundaries of that map or revisit areas within the map which are not well known. A simple taxonomy of the literature divides the field into metric (grid) based approaches which represent the geometric properties of the environment; feature-based techniques which accompany the SLAM approach to robotic mapping; and topological approaches representing the connectivity between places.

4.2.2.1 Grid Based Exploration

A fundamental approach to grid based mapping is the occupancy grid mapping algorithm introduced by Moravec and Elfes in 1985 (Moravec and Elfes, 1985; Elfes, 1987, 1989), the implementation and operation of which we have covered in Section 3.2.1. To recap, occupancy grids represent the map as an evenly spaced grid of random variables where each random variable corresponds to the probability of occupancy of the location it covers. Initially, all cells are assigned a prior probability of occupancy and subsequent sensing leads to this value being updated and classed as either open, unknown or occupied. By associating probabilities with grid cells, occupancy grid mapping lends itself well to both information theoretic techniques which seek to minimize uncertainty in the map, as well as those techniques that seek to maximize coverage of the unseen area.

Frontier Based Navigation A popular approach to exploration is to focus solely on covering the terrain in its entirety as quickly as possible. Yamauchi (1997) proposed the frontier-based navigation strategy which theorizes that to gain the most new information about the world, a robot should move to the boundary between open space and previously unexplored territory. A process similar to edge detection or region extraction in computer vision is then used to find the boundary regions which border open space and unknown space in the occupancy grid map. Adjacent frontier cells are grouped into frontier regions, and any region exceeding a threshold size is considered a frontier. The robot then applies a greedy approach of navigating to the nearest accessible frontier. Upon reaching the frontier, the frontier detection is repeated and a new frontier is targeted. Frontier-based approaches have had many successful applications, particularly in indoor environments. With perfect sensing and motor control the frontier based method will eventually lead to the exploration of all accessible space in the world. Koenig et al. (2001) has shown that the worst

case travel distance of greedy frontier-based exploration is of order $|V|ln|V|$ edge traversals (where V corresponds to the number of detected frontiers) and argues that any performance disadvantage is outweighed by the simplicity of its implementation. Nevertheless, this greedy exploration technique does not always lead to exploration strategies that are in the best interests of the robot - most often it is beneficial to also take into account the quality of the underlying map we are building and how that will affect our ability to navigate and reason about the world.

Information Gain Exploration Hence other grid based approaches use a statistical formulation to motivate exploration by seeking to maximize the information known about the world and/or robot position. From information theory, the *information* held in a probability distribution p is equal to its entropy

$$H_p(x) = -\int p(x) \log p(x) dx \tag{4.1}$$

and is maximal for a uniform distribution and minimal for a point-mass distribution. The information gain for the belief of the distribution I_b after executing action u and obtaining the observation z of the hidden state x' is

$$I_b(u) = H_p(x) - E_z[H_b(x'|z,u)] \tag{4.2}$$

where E_z is the expectation over the measurement z and $H_b(x'|z,u)$ is the entropy of the conditional distribution $p(x'|z,u)$. The maximal value of information gain can be used to arbitrate between available actions u.

A number of strategies exploit the proportional relationship between the entropy of a Gaussian and the determinant of its covariance to produce exploration policies which maximize the expected information gain. A multivariate Gaussian distribution

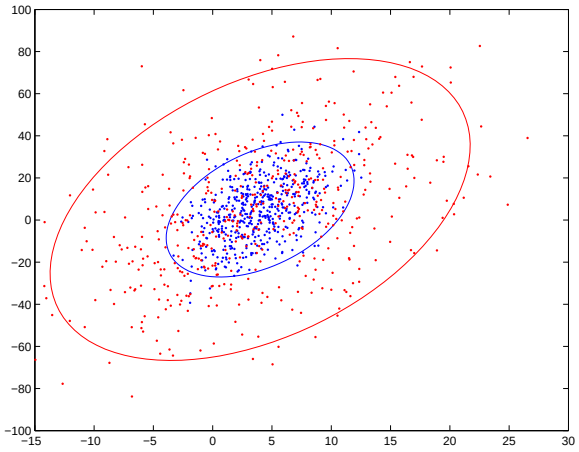


Figure 4.3: The covariance matrix and its geometric interpretation. The blue distribution is drawn from $p(x_1;\mu_1,\Sigma_1)=\mathcal{N}\left(\begin{bmatrix}4\\5\end{bmatrix},\begin{bmatrix}10 & 20\\1 & 200\end{bmatrix}\right)$ the red from $p(x_2;\mu_2,\Sigma_2)=\mathcal{N}\left(\begin{bmatrix}4\\5\end{bmatrix},\begin{bmatrix}60 & 103\\103 & 989\end{bmatrix}\right)$ which was created by scaling up the covariance of the blue distribution. This was done by multiplying the component eigenvalue matrix of the blue distribution's covariance by 5 times the identity matrix and recalculating the covariance from its decomposition. Notice the inflated covariance of the red distribution leads to greater scattering of the data points, and hence what we refer to as greater uncertainty. The axes of the covariance ellipses drawn here are given by the eigenvectors of the covariance matrix. Their lengths are given by the square roots of the corresponding eigenvalues.

of dimension d, with mean μ and covariance matrix Σ:

$$p(x;\mu,\Sigma)=\frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}}\exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right) \tag{4.3}$$

has entropy

$$H=\frac{d}{2}(1+log2\pi)+\frac{1}{2}\log|\Sigma|. \tag{4.4}$$

As Figure 4.3 shows for 2-dimensions, the covariance matrix lends itself to a handy geometrical interpretation. The eigenvectors of Σ correspond to the principal axes of ellipsoids which equate to surfaces of equal density of the distribution. The determinant of the covariance in equation 4.4 hence provides a measure of the volume of the hyper-ellipsoid bounding the uncertainty of a Gaussian estimate.

Bourgault et al. (2002) combine the feature-based approach of Feder et al. (1999) (discussed in Section 4.2.2.2) in a weighted utility function with a second metric

which seeks to simultaneously maximize the information gain over an occupancy grid maintained additionally to the SLAM map. This attempts to counter-balance the localization behaviour described in Feder et al. (1999) (which tends to restrict the robot's trajectory to navigating around known features in the map) by directing the robot to unknown areas.

In Thrun et al. (2005) and Stachniss et al. (2005) the entropy of the SLAM posterior (equations 3.14 and 3.15 discussed in Section 3.3.1) is decomposed into two terms, one representing the entropy in the robot's trajectory and the other representing the expected entropy of the map averaged over all paths. The method evaluates possible control sequences by adding the two entropy terms and selecting the control that maximizes the combined information gain. Decomposing the problem in this manner produces an algorithm that exhibits loop closing behaviour to aid in localizing the robot, as well as seeking to explore unmapped terrain. However, the scalability of this approach is limited as it employs a particle filter SLAM formulation in which each particle must maintain its own occupancy grid, rather than a list of landmarks. It is thus applicable only to small regions and limited to a one-step lookahead.

Chli and Davison (2009) use an information theory approach to guide the (image) feature by feature matching search (akin to data association between features in subsequent images) which is integral in determining correspondences between one state and another in a visual SLAM process. Visual SLAM is SLAM with a single camera (and therefore no robot motion control input or motion model), so the correspondences are needed to estimate directly the camera's motion from the image data. In this work the *Mutual information* $I(\mathbf{x}; \mathbf{z_i})$ - how many bits of information we expect to learn about the uncertain state vector $\mathbf{x}$ by determining the exact value of the candidate measurement $\mathbf{z_i}$ - is used to prioritize the order in which features and search regions are examined when a new image arrives. While the authors describe this technique in the context of visual tracking; the notion of using mutual information

between the state estimate and possible measurements has obvious carry-over to the exploration problem - where moving the robot in the direction of the best candidate measurement is a desirable behaviour.

4.2.2.2 Feature Based Exploration

Feature-based approaches rely on the ability to uniquely identify landmarks in the environment during mapping. In Feder et al. (1999) the authors seek to maximize the information contained in the posterior probability density function of the SLAM process (again, see equations 3.14 and 3.15). Information gain is defined in terms of the *Fisher information matrix*. It is shown that the action that maximizes information gain also minimizes the uncertainty in the stochastic map. This allows the definition of a metric which calculates the total area of the error ellipses of the features and vehicle position in the map. This metric is used to achieve locally optimal adaptive sensing over the discrete action space of the robot.

In Makarenko et al. (2002) landmarks are extracted from laser range scans and a utility function trades off information gain, cost of travel and the potential reduction in pose uncertainty in evaluating the next robot sensing location. In Newman et al. (2003) features within the map are used to determine nearby unexplored areas that are likely to lead to exploration. Each feature generates a set of surrounding goals which depend on the feature's own geometry and type. Goals are sited such that if selected, they would contribute to the objective of exploration of sparse or open areas with a preference for local exploration. A graph structure is built as the robot traverses the environment as a free space road map for use in path planning when global exploration is required.

Sim and Roy (2005) use the trace (rather than the determinant) of the covariance matrix to provide a measure of the average uncertainty of the SLAM map. Minimizing the trace of the matrix is used as the objective function for exploration, possible

actions are drawn from the eight-connected neighbourhood surrounding the current pose, and a maximally informative trajectory is chosen by integrating observations along the trajectory to a particular point. The complexity of the algorithm is reduced by prohibiting self intersecting trajectories. This algorithm is globally optimal in contrast to the greedy local approaches of those detailed above.

4.2.2.3 Topological Exploration

Topological exploration methods represent the environment with a coarse, graph like structure. The representation is attractive as its compact nature should scale more easily to larger environments than fine grained metric representations, which exhibit both memory and time complexity. Such representations are also compatible with the large body of existing graph based problem solving techniques. Purely topological mapping algorithms suffer from a lack of truly distinctive landmarks (which are integral to the technique as they form the nodes of the graph). Landmarks that are distinctive to the human eye (such as corners) are often difficult to sense and localize against. Typically, topological strategies are used in conjunction with metric mapping techniques.

In Kuipers and Byun (1991) a *spatial semantic hierarchy* is used to address the autonomous exploration problem. Nodes of a graph correspond to locally distinctive landmarks which are identified through the calculation of the local maxima of specific functions of sensor readings. The nodes are connected via local control strategies for travelling between the distinctive landmarks and metric information is first accumulated at the node level and later merged into a global metric map. Ambiguity between nodes is resolved at the topological level by a *rehearsal procedure* which uses previous local control strategies to travel between landmarks and hence differentiate between unexplored territory and previously seen landmarks.

In Tovar et al. (2007) a data structure called the *Gap Navigation Tree* (GNT)

enables locally optimal navigation in terms of Euclidean distance traveled in the exploration of a simple planar environment. Underpinning the algorithm is the assumption of the availability of an abstract *gap sensor*, able to track and uniquely identify discontinuities in the depth of the range information at the robot's current viewpoint. These *gaps* are of interest as they correspond to regions not visible to the robot. The gaps are represented in the tree in a particular order corresponding to the current local view of the robot. Gaps that are not children of the root of the tree correspond to gaps that have been merged to create those gaps that are currently visible to the robot. The algorithm distinguishes between *primitive* gaps that correspond to regions that were once visible but are now occluded, and *non-primitive* gaps that correspond to unexplored regions. By continually pursuing non-primitive gaps until the tree contains only primitive nodes, the Gap Navigation Tree has been proved to achieve coverage of unknown planar environments.

We decided to pursue an implementation of the GNT for our robotics platforms as the anthropomorphic intuitive nature of 'looking for gaps' and the promise of complete coverage of planar environments is highly desirable. In comparison with the other approaches discussed here, 'gaps' are simple, obvious features and pursuing them would lead the robot to sensible exploration strategies (i.e. to the end of corridors, to doorways or to try and peek around corners). The detection and tracking of gaps also promised a lightweight process that would add little overhead when used as a driver for an underlying metric mapping system consistent with the overall aim of creating detailed maps autonomously. In the remainder of the chapter we detail our implementation of the GNT.

4.3 The Gap Navigation Tree

This section summarizes the GNT of LaValle (2006); Tovar et al. (2004, 2007) as it is central to the work presented in the remainder of the chapter. The GNT arose out of a desire to explore what is the minimum amount of information needed by a robot to accomplish simple tasks, with the idea of countering popular belief amongst roboticists that more information is better (Tovar et al., 2004). Under the minimalist approach, the authors seek to avoid costly sensors, and shirk costly metric modelling of the environment. As a result the GNT requires only one sensor, capable of tracking the direction of depth discontinuities but not requiring it to measure any angles, distances or localize itself in the environment. The world in which the GNT operates is assumed to be a simply connected, closed polygonal environment.

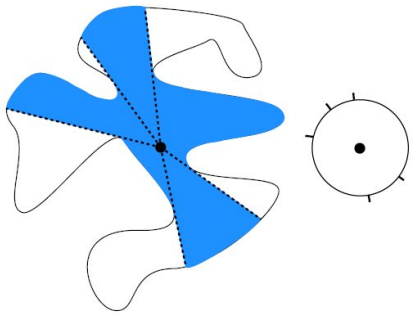


Figure 4.4: The blue shaded area of the planar world on the left shows which areas of the world are visible to the robot (the black disc). The image on the right shows the gap sensor output, notches correspond to the angles at which gaps appear to the robot. Image courtesy of Tovar et al. (2007).

Figure 4.4 illustrates how gaps are used to differentiate between areas of the environment the robot can currently see and those it can not. The gap sensor reports the cyclical order of depth discontinuities of the environment boundary relative to the robot's current position. The angles are not important, just the correct cyclical order. In Tovar et al. (2007) the authors suggest a number of possible physical implementations of the sensor, including sonars, cameras, a laser pointer used in conjunction with an omni directional camera, and their own implementation of Figure

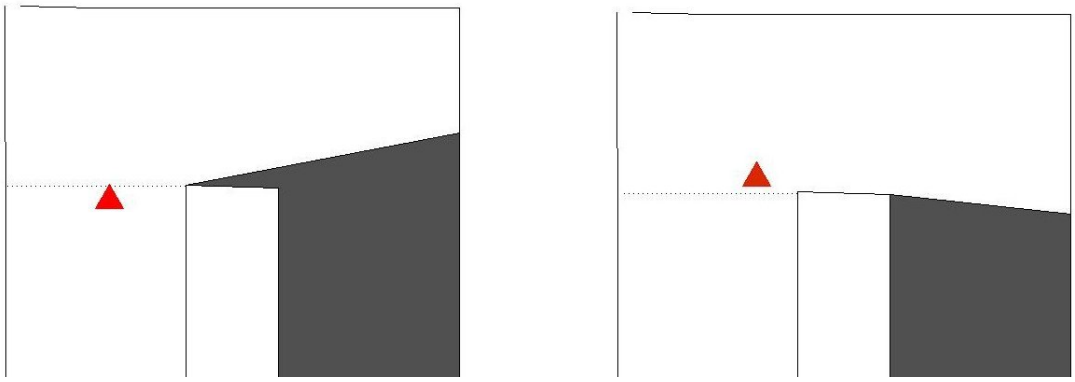
4.5 using two laser range finders.



Figure 4.5: The gap sensor used by Tovar et al., constructed from two 2D laser range finders - one forward facing, one backward facing. Gaps are detected by detecting large differences between neighbouring returns in the raw laser data. Image courtesy of Tovar et al. (2007)

Each of the *gaps* returned by the sensor should correspond to a region of free space that is occluded. To the sensor, the gap is distinguished by the bearing that results from drawing a line between the robot's current position to the closest point on the boundary of the occluded region. It is assumed that the gap sensor is able to track and distinguish between the observable gaps at all times, even when the robot moves across a non-smooth part of the boundary and the location of the gap *jumps* discontinuously (see Figure 4.6 for an example of what this means). Any planar environment can be decomposed into regions of similar *visibility*, Figure 4.8(a) shows a simple environment broken up into 7 such regions. We require the gap sensor to return the same number and relative bearing of gaps from any location within one of these regions even though the size and shape of the occluded area will morph with the robot's position. As no geometric information is returned by the gap sensor, the gap sensor output is simply an ordered list of unique identifiers assigned to each gap $G(x) = [g_1, ..., g_n]$, where $G(x)$ represents the set of gaps observed at position x.

Under the GNT paradigm the robot need only be equipped with a single motion primitive which enables it to rotate itself towards the location of a gap and approach the gap at a constant speed. This is referred to as *chasing* the gap. The *chase gap*

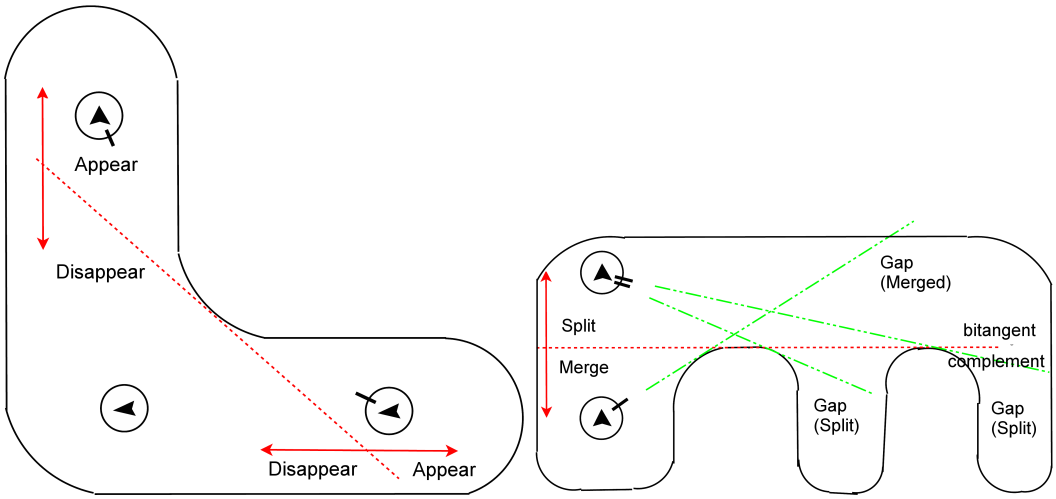


(a) About to cross a point of inflection

(b) Crossing the point of inflection. Note the dramatic shift in the location of the occluded area.

Figure 4.6: When the robot moves across a non-smooth part of the boundary (crossing the corner), the location of the gap and the occluded region it represents jumps discontinuously along the wall.

operation can only terminate when the gap disappears from the gap sensor.



(a) Gap Appearance and Disappearance, the robot crosses a point of inflection in the planar world.

(b) Gap Splitting and Merging - occurs when the robot crosses a bitangent to the planar world's boundary.

Figure 4.7: Visual Events

The algorithm constructs a topological representation of the environment in the form of a tree with the aid of the abstract *gap sensor*. Construction of the GNT begins with the addition of n nodes as children of the root of a tree, where n is the number of gaps returned by the gap sensor in its initial observation of the environment - one node for each gap. As the robot moves through the environment, changes to

the tree are triggered by the occurrence of *gap critical events*. These events occur when the robot crosses either a *generalized inflection* or a *generalized bitangent* of the environment boundary, and represents transiting into a different *visibility region*, such as transiting between any of the 7 cells in Figure 4.8(a). As illustrated in Figure 4.7, the *appear* and *disappear* events are associated with crossing a line of inflection, and crossing a bitangent line will trigger a *split* or *merge* of two gaps, depending on the direction of crossing. Each event requires updating the GNT:

Appear As the robot moves through the environment a part of the world that was once visible becomes occluded again. A node g is added as a child of the root node, its neighbouring nodes are those that it neighbours in the gap sensor. The cyclical ordering of the gaps as they appear to the robot must be maintained in the tree.

Disappear When a previously hidden section of the world becomes visible due to the environment geometry and the movement of the robot, a gap is said to disappear. By definition any disappearing gap must currently be represented by a leaf node. This node is removed from the tree.

Split A gap previously associated with a single occlusion splits into two separate regions due to the robot's vantage point and the structure of the environment boundary. If the gap g which has split into gaps g_1 and g_2 is a leaf node, then two new vertices are added to the tree in place of g. Otherwise, they must already exist in the tree as children of g and become children of the root node upon the removal of g.

Merge The opposite of the split event, merging occurs when two occluded regions meld into one. When gaps g_1 and g_2 merge into gap g, g is added to the tree as a child of the root node, preserving the order of the gap sensor. The existing nodes g_1 and g_2 become children of g.

These four operations are sufficient to represent all feasible changes to the environment. Although more complex tritangents may exist, both Tovar et al. (2007) and Forsyth and Ponce (2002) argue that tritangents would not survive even a small deformation of the environment whereas the bitangent or inflectional tangent would merely change position.

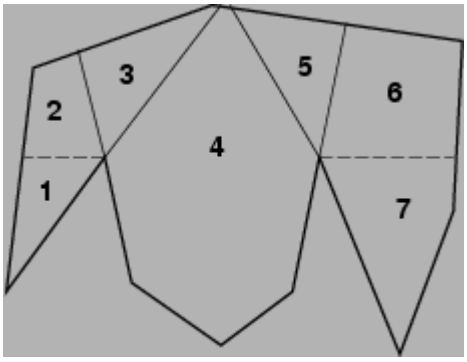
At any time τ, the children of the root of the GNT reflect the gaps currently visible to the gap sensor. Any other gaps in the tree were visible at some time $t < \tau$ but are now obscured due to merging.

Nodes in the tree are only used to represent gaps, and gaps fall into two categories. *Non-primitive* nodes are used to motivate exploration, they correspond to unexplored occluded regions and arise as a result of appearing in the gap sensor's initial observation, or from the *splitting* of one of the initial nodes or its non-primitive children. *Primitive* nodes are added to the tree as a result of a previously visible area becoming occluded and hence causing a gap to appear. Chasing a primitive gap will only result in its disappearance and the retracing of previously covered territory.

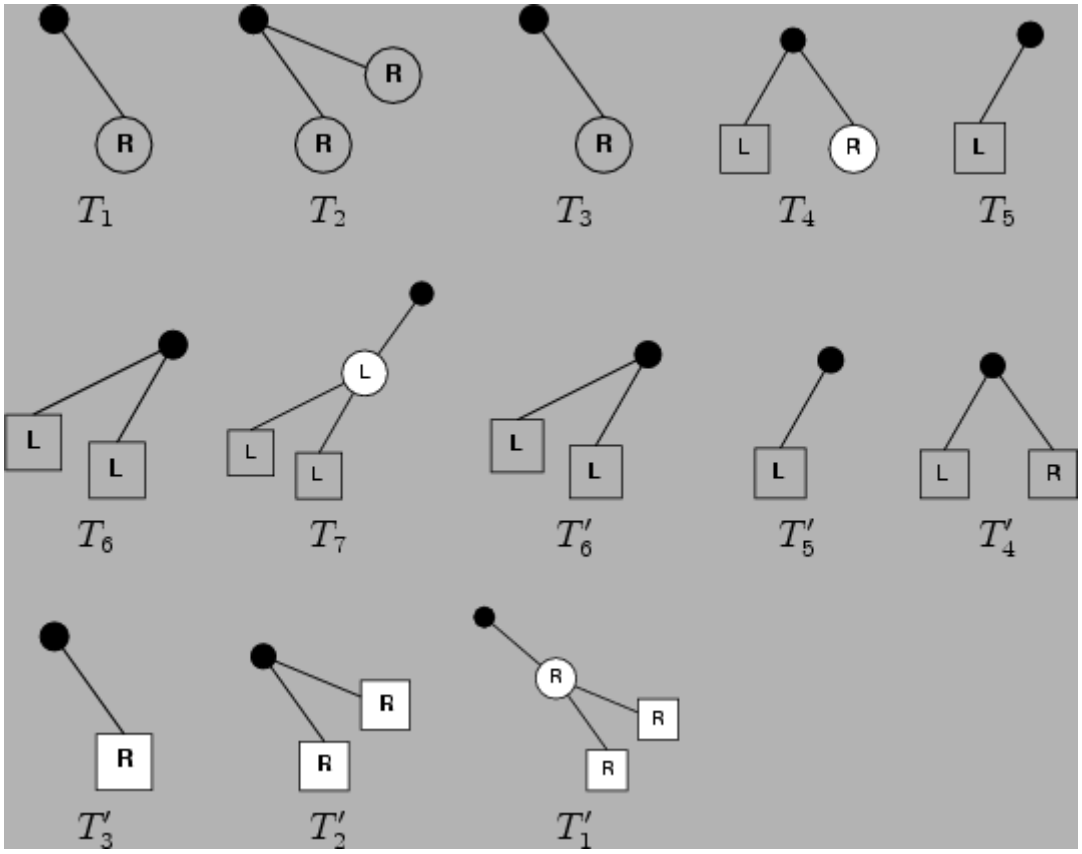
A *complete* GNT encodes a path from the robot's current position to any other location in the environment. To construct the complete GNT, the robot must achieve sensor coverage of the entire environment, and the GNT is forced into completeness by iteratively chasing the non-primitive leaves of the tree until they, and any children that result from their subsequent splitting into other gaps, disappear. When a leaf disappears, another non-primitive leaf is selected for chasing and the procedure is repeated until all leaves are primitive.

The algorithm of Figure 4.9 summarizes the process of exploring an unknown environment with the GNT.

The GNT can also be used for optimal navigation. Although it is impossible using this topological representation to express a goal state in $\mathbb{R}^2$, we can express goals in terms of the vertex that needs to be chased down in order to make the goal state



(a) A simple environment



(b) Construction of the GNT for this environment as a robot navigates from regions 1 - 7. Circular nodes denote non-primitive (unseen) gaps, square nodes represent primitive gaps denoting previously visible regions. "L" or "R" inside each vertex denotes whether the shadow component is to the right or left of the gap. Note that once cell 5 has been reached, the robot has seen the entire environment.

Figure 4.8: GNT construction for a simple planar environment. Image courtesy of LaValle (2006)

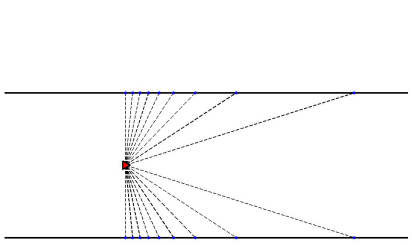
```
Gap Navigation Tree Algorithm
    Require: Set of gaps G(x)
1   Initialize GNT from G(x)
2   while ∃ nonprimitive g_i ∈ GNT
3         do choose any nonprimitive g_k ∈ GNT
4            for each time step s
5                  do CHASE(g_k) until critical-event
6                     update GNT according to critical-event
7   return GNT
```

Figure 4.9: Gap Navigation Tree Algorithm

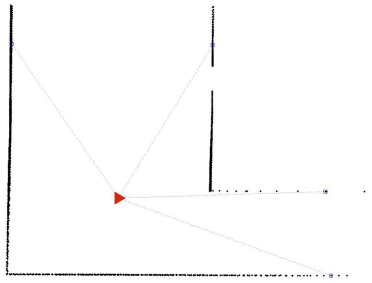
visible. In Tovar et al. (2003) it was shown that in when executing the *chase-gap* procedure the robot is following the paths of the shortest-path roadmap, described in Section 2.1.1.

Although the GNT has some nice properties which lends itself well for use in next action selection in conjunction with a metric SLAM system, it has some crippling limitations in its original form. As already discussed (and outlined by the original authors in Tovar et al. (2007)), the most important practical issue prohibiting the use of the GNT in a real setting lies with the gap sensing and tracking process. Implicit in the GNT algorithm is the assumption of the perfect gap sensor/gap tracker. Preliminary investigations using our simulated system showed some critical problems:

- The presence of sensor noise can cause gaps to be detected where they do not exist.
- When the robot is close to a wall (see Figure 4.10(a)) the distance between consecutive data points in a one-shot laser range scan can be great enough for a gap to be (falsely) detected. It is impossible to set a threshold on discrepancy that both ameliorates this problem and allows true gaps like doorways to still be detected.
- The limited range of the laser scanner means that gaps are detected at the



(a) Angular resolution and range affect the operation of the gap sensor, causing faux gaps to occur



(b) Faux gaps caused by sensing obliquely to the wall, and by the limited range of the laser scanner. This screenshot is taken from our simulated system.

Figure 4.10: Difficult cases for a real world gap sensor.

extents of the known environment (see Figure 4.10(b)). As there is no way of knowing whether they correspond to corners, or just the extent of the wall visible using the sensor, they must be assumed to be gaps and tracked accordingly.

- When the robot crosses between visibility regions (as illustrated in Figure 4.6) the location of the gap and the area it represents can jump dramatically. Tracking this using a one-shot sensor is extremely difficult.

Should the gap sensor not prove robust to these situations, the robot will fail to detect important visual events leading to malformations of the tree structure, or causing the robot to chase the 'wrong' gap. The GNT does not have a probabilistic basis and our experience shows it to be fragile when driven with real sensors.

4.4 A Probabilistic Gap Sensor

Inspired by these limitations, the remainder of this chapter details an implementation of the gap sensor which utilizes a partially-complete SLAM map as a basis for gap detection and tracking. The success of view-based SLAM methods, in particular those based on mobile robot systems equipped with laser scanners, has lead us to

focus on the application of the GNT to a SLAM system producing point cloud maps. A SLAM system using point clouds will still lead to the sparse sampling of some surfaces, depending on the current orientation of the sensor relative to the surface and also due to the sensor's distance from the surface. However, we show the use of a SLAM map not only reduces the number of *faux gaps* due to sparsity, but also allows us to derive a model for expected gap motion hence allowing for robust tracking.

The following sections detail the novel contribution made to improving the real world performance of the Gap Navigation Tree algorithm. Methods of detecting and tracking gaps are described, as well as how the GNT's *critical events* are detected under this framework. Results showing the improved gap sensor in operation are provided and discussed.

4.5 Gap Detection

Before gaps can be tracked, we must first define a capability that allows the detection of gaps in the accumulated SLAM map. A visibility based formulation similar to Landa et al. (2007, 2006) is used to accomplish this.

To compute the visibility map we take a point cloud $\mathbf{P}$ sampled from the environment Ω. We limit the view direction from the current vehicle position to the set of angles $\theta \in [0, \theta_d, 2\theta_d ... 2\pi)$ where θ_d is the desired angular resolution of our gap sensor. The view direction from the vehicle (at x_0) to any given point (x) is calculated as

$$v\left(x_0, x\right) = \left(\frac{x - x_0}{|x - x_0|}\right). \tag{4.5}$$

The visibility function is then

$$\rho_{x_0}(\theta) = \begin{cases} min_{x \in \Omega}\{|x - x_0| : v(x_0, x) = \theta\} & \text{if exists} \\ r_{max} & \text{otherwise} \end{cases} \tag{4.6}$$

where r_{max} is the maximum range of the sensor. This visibility function provides an approximation to the visible region (the visibility map) at the robot's current viewpoint.

The visibility function is then differentiated to locate gaps in the visible region

$$\frac{d\rho_{x_0}}{d\theta} \approx \frac{\rho_{x_0}(\theta_{k+1}) - \rho_{x_0}(\theta_k)}{\theta_{k+1} - \theta_k} = \frac{\Delta p}{\Delta \theta}. \tag{4.7}$$

Figure 4.12 graphs the visibility function used to produce the visibility map and associated gaps of Figure 4.11.

Gaps are classified as existing at those locations where $\frac{d\rho_{x_0}}{d\theta}$ exceeds a threshold $\mathbf{T}$. Note that this threshold is set to be dependent on characteristics of the environment (say known doorway widths or corridor widths in an indoor environment, or spaces between buildings in an urban setting). Prudent selection of this threshold aids in limiting the appearance of *spurious gaps*. We assume each gap can be uniquely parameterized by a *tangent point* $[x_g, y_g]$; the closest point on the boundary which intersects a line drawn in the view direction from the robot's current position.

It is important to note that due to the discretization of the data and the discontinuous representation of the environment offered by a point based SLAM map, the gap detector often returns gaps in locations when there are none. For this reason, we employ the notion of persistence (explained in Section 4.7) before adding gaps to the gap tracker or gap navigation tree.

We also employ filtering to reduce the effect of these spurious gaps. Multiple gaps appearing close together tends to indicate that the particular area of the map is not well known. As explained later, the gap tracker uses measurements returned by the gap detector directly in detecting splits/merges of gaps that need to be reflected in the gap navigation tree. It is important therefore to reduce the incidence of spurious gaps being returned by the gap detector. We are able to use this blunt approach

to filtering out unwanted gaps as real gaps will continue to be detected by the gap detector as the map is incrementally built, while the spurious gaps around them will disappear over time.

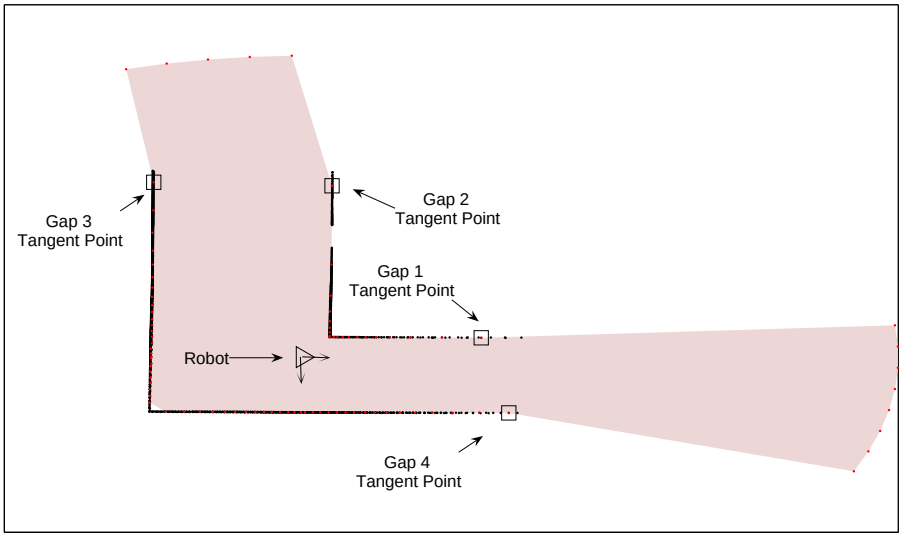


Figure 4.11: A Visibility Map showing an approximation to the area visible from the robot's current position. The visible region is shaded in red.

In addition to the tangent point, the gap detector also records whether the gap results from a discontinuity which stretches from a near point on the environment boundary to a distant point or conversely from far to near, relative to the anti-clockwise direction with zero degrees being at the robot's current heading. This information enables us to identify whether the occluded region lies to the left or right of the tangent point, aiding in selection of the best viewpoint to travel to next and expediting the gap's disappearance.

4.6 Gap Tracking Metric

To introduce our gap tracking algorithm we present the gap tracking problem from a probabilistic point of view. We are interested in maintaining the distribution

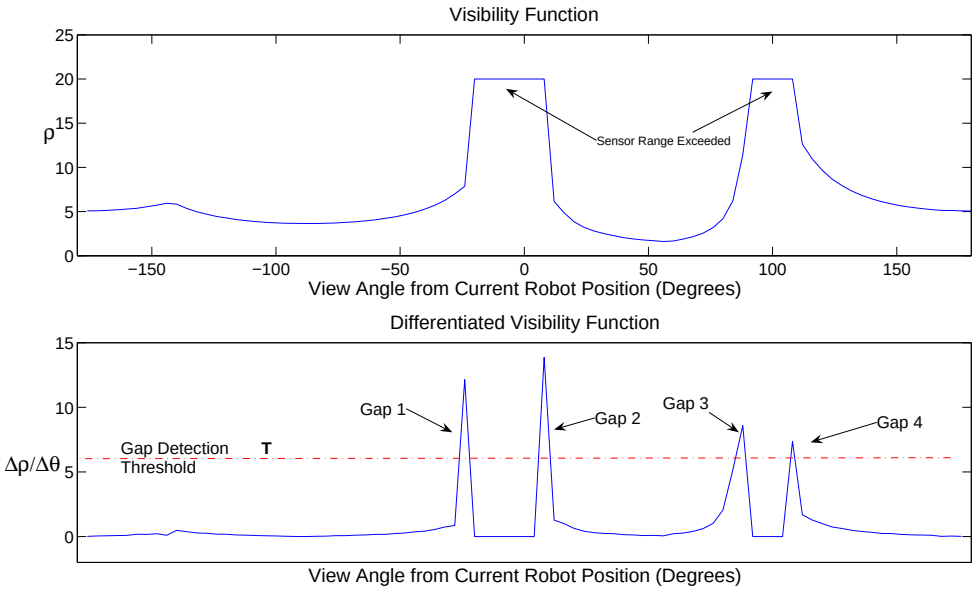


Figure 4.12: Visibility Function (top) and Difference Function (bottom) corresponding to Figure 4.11. In this example four gaps are detected.

$p(g_k|D^k, m_k)$ over the location $[x_g, y_g]$ of gap g at time k given the history of our gap detector outputs $D^k = \{d_1, d_2 \dots d_k\}$ and a 3xN matrix of points representing the current state of the map m_k.

We assume that our gap tracking problem is a Markov process, such that knowledge of the state at $k-1$ provides all the information needed to propagate the state at k. This allows us to write the *gap motion model* $p\left(g_k|g_{k-1}, m_k\right)$ and the *gap observation model* $p\left(d|g_k, m_k\right)$ as probabilistic models adhering to the Markov property. The graphical model of Figure 4.13 illustrates the relationship between the map, measurements and state of the gaps.

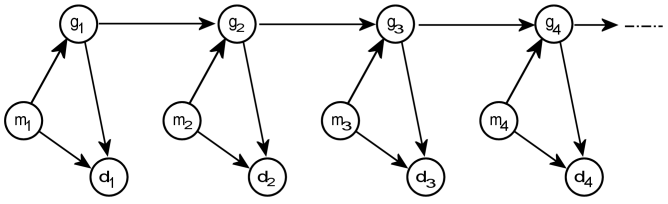


Figure 4.13: A graphical model showing the relationship between the measurements, map and gap position

Assuming that the current state of the gaps is independent of the previous state, gap prediction becomes:

$$\begin{aligned} p\left(g_k \mid D^{k-1}, m_k\right) &= \int p\left(g_k, g_{k-1} \mid D^{k-1}, m_k\right) dg_{k-1} \\ &= \int p\left(g_k \mid g_{k-1}, D^{k-1}, m_k\right) \times \\ &\quad p\left(g_{k-1} \mid D^{k-1}, m_k\right) dg_{k-1}. \end{aligned} \tag{4.8}$$

Once we have the relevant measurements we are able to update the state representation:

$$\begin{aligned} p\left(g_k \mid D^{k}, m_k\right) &= \frac{p(g_k, D^k, m_k)}{p(D^k, m^k)} \\ &= \frac{p(d_k \mid g_k, D^{k-1}, m_k) p(g_k \mid D^{k-1}, m_k)}{p(D^k, m_k)} \times \\ &\quad \frac{p(D^{k-1}, m_k)}{p(D^k, m_k)} \\ &= p(d_k \mid g_k, m_k) p(g_k \mid D^{k-1}, m_k) \frac{p(D^{k-1}, m_k)}{p(D^k, m_k)} \\ &= \frac{p(d_k \mid g_k, m_k) p(g_k \mid D^{k-1}, m_k)}{p(d_k \mid D^{k-1}, m_k)} \\ &= \frac{p(d_k \mid g_k, m_k) p(g_k \mid D^{k-1}, m_k)}{\int p(d_k \mid g_k, m_k) p(g_k \mid D^{k-1}, m_k) dg_k}. \end{aligned} \tag{4.9}$$

Combining equations 4.8 and 4.9 allows us to write

$$p(g_k \mid D^k, m_k) = \frac{p(d_k \mid g_k, m_k) \int p\left(g_k \mid g_{k-1}, m_k\right) p\left(g_{k-1} \mid D^{k-1}, m_k\right) dg_{k-1}}{\int p(d_k \mid g_k, m_k) p(g_k \mid D^{k-1}, m_k) dg_k}. \tag{4.10}$$

The above equation is recursive and allows well known (Kalman) closed form updates if we assume Gaussian distributions.

Critically important to our gap tracking solution is the gap motion model:

$$p(g_k \mid g_{k-1}, m_k). \tag{4.11}$$

This function represents the gap's change in position from time k to $k+1$. We assume it is a Gaussian of the form

$$p\left(g_{k+1} \mid g_k, d^k, m_k\right) \sim N(g_k, P_{k-1} + \Gamma(g_k, m_k)). \tag{4.12}$$

Γ is a function parameterized by the gap and the SLAM map and P_{k-1} is the gap's covariance at time $k-1$. Γ is a positive semi-definite matrix which factors as

$$\Gamma(g_k, m_k) = VD\Lambda V^T, \tag{4.13}$$

where V is a rotation matrix of ordered eigenvectors, D is a scaling matrix used to stretch the covariance in the direction of the largest eigenvector (following the process described in Figure /reffig:covuncertainty), and Λ is a diagonal matrix of eigenvalues resulting from the eigenvalue decomposition of the covariance matrix of the nearest neighbours; such that the following equation holds

$$V\Lambda V^T = cov[K(m_k, g_k)]. \tag{4.14}$$

$K(m_k, g_k)$ is the neighbourhood operator on m_k in the region of g_k, which returns n closest points to the estimated gap position. Figure 4.14 illustrates this process.

While *genuine* gaps that appear in the gap sensor as a result of generalized inflections and bitangents discussed in Section 4.3 can be adequately represented and tracked using a circular covariance function, we are also required to maintain representations of *faux gaps* that result from the limited range of our sensor. These gaps tend to jump with the arrival of new data. Take, for example, a gap marking the extent of our vision along a wall. As the next chunk of data comes in, we see an extra x metres of the wall and the gap moves by x metres in accordance.

Note that as these tangent points are located at the map extremity, they are

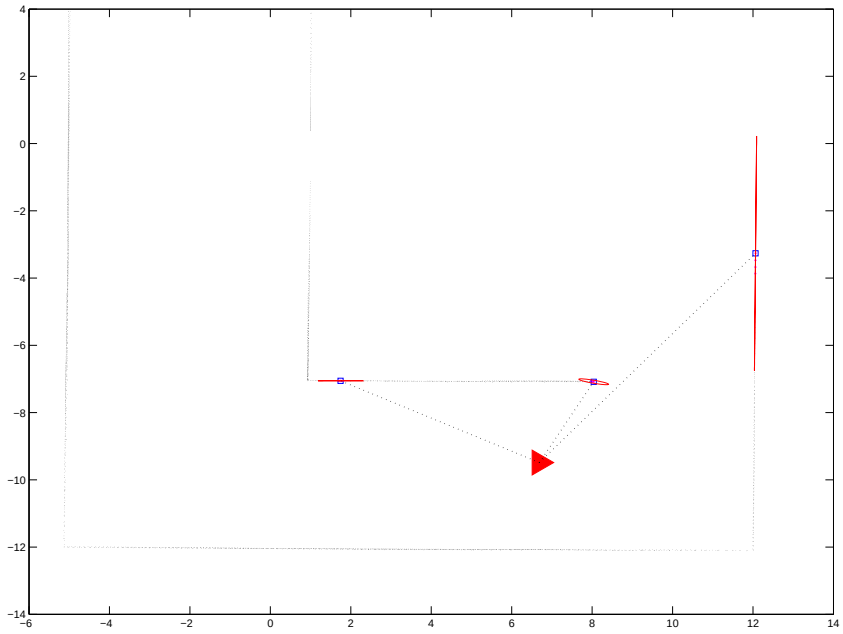


Figure 4.14: Calculating covariances for different gaps. Dot points indicate the current SLAM map. Magenta points are the closest points returned by the neighbourhood operator and the red ellipses denote the uncertainty in our estimate of the gap position in accordance with the gap motion model. The left most gap is currently estimated to lie along the wall and is estimated to move in accordance. The middle gap actually corresponds to an as-yet-unseen corner location so its covariance is not parallel to the wall. The right-most gap is a *faux gap* located at the extent of a partially obscured wall - you can see how far apart the magenta dots (nearest neighbours) are. As the full extent of the wall is yet to be seen, data points at its end are sparsely located. Hence the covariance is large, reflecting our uncertainty as to the true length of the wall.

also surrounded by few data points. We allow the local map shape and point cloud density to dictate the motion of perceived gaps by examining the nearest neighbours of the tangent point (within radius R), calculating the covariance of these points as in Equation 4.14 and exaggerating the major axis with an appropriate choice of scaling matrix D in Equation 4.13. Note that this method when applied to *genuine* gaps tends to produce circles of small radius due to the agglomeration of data corresponding with their typical physical presence as corners in a structured environment and reflecting the belief that these gaps are unlikely to move when next observed. In contrast, our *faux gaps* tend to lie on the visible extremity of a wall, so the direction of minimum variance is perpendicular to the wall (and unlikely to change) while the maximum

variance is in the direction parallel to the wall. The large covariance parallel to the wall reflects the belief that subsequent observations will see this gap shift along the wall with the addition of new data.

4.7 Data Association

A nearest neighbour χ^2 test is used to find the most probable association between measurements $D = [d_1, d_2, ...d_n]$ returned by our gap detector and $\hat{G} = [g_1, g_2, ...g_m]$ the predicted locations of the gaps currently being tracked. This association is used to update the gap positions.

We also maintain an association matrix A of all measurements that gate with current gap positions given a less stringent bound γ

$$A_{ij} = \begin{cases} 0 & \chi^2_{ij} > \gamma \\ 1 & \chi^2_{ij} \leq \gamma \end{cases} \tag{4.15}$$

where χ^2_{ij} is the Mahalanobis distance between the detected gap d_i and known gap g_j. This association matrix is used to spot splitting and merging of gaps for use in the construction of the gap navigation tree. A split corresponds to multiple measurements being associated with one gap, and conversely a merge equates to multiple gaps associating with the one measurement.

Unexplained measurements (those that do not associate with any existing gaps) are used in the initiation of new gaps. Spurious gaps are problematic and we do not want to initialize a gap based on a single appearance in the gap sensor. A list of putative gaps that have associated with prior putative gaps is maintained, these gaps are added to the gap tracker and the gap navigation tree once they have been observed a threshold number of times and we are hence confident they are actual gaps.

4.8 Results

4.8.1 Performance of the Gap Motion Model

We first evaluate the performance of the gap motion model prior to examining the completion properties of the GNT. Figure 4.15 shows the progress of the robot through a simulated 2D world where our gap sensor is informed only by noisy, sparse data from a 2D laser scanner. The current state of the tree is shown at each time step, the l or r notation at each node shows whether the associated gap occludes an area to the left or right of the robot as defined in Section 4.5. The accumulated point cloud map is shown together with square markers indicating the position of currently tracked gaps. The covariances of each gap are shown as red ellipses. The dotted lines emanating from the robot indicate the direction of gaps detected by the gap detector (note that in some instances these gaps may have recently appeared and are hence not yet tracked by the gap tracker).

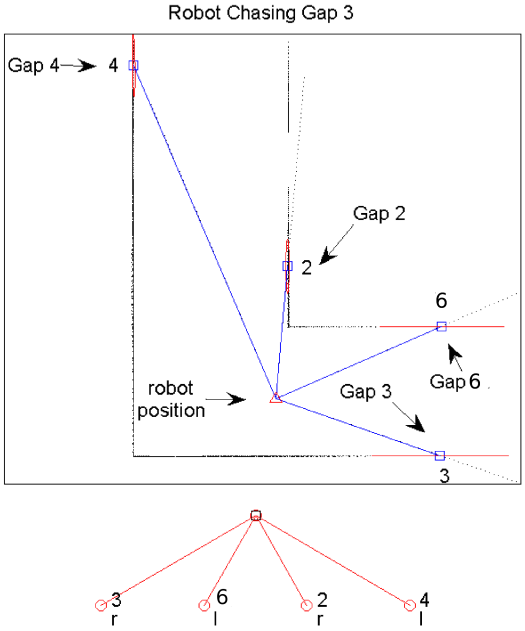
In Figure 4.15(a) four gaps are being tracked and the robot is in pursuit of gap 3. All four gaps lie at the extent of known walls and hence the covariances (in red) are maximal in the direction parallel to the wall and are large in size, reflecting the belief that these gaps are likely to move as the map is incremented.

Figure 4.15(b) shows that as the robot moves past the corner, gaps 2 and 3 have disappeared as the map has been *filled in*. The occluded region which lies to the left of the tangent point of gap 4 in Figure 4.15(a) now forms part of the larger occluded region to the right of gap 4's updated tangent point.

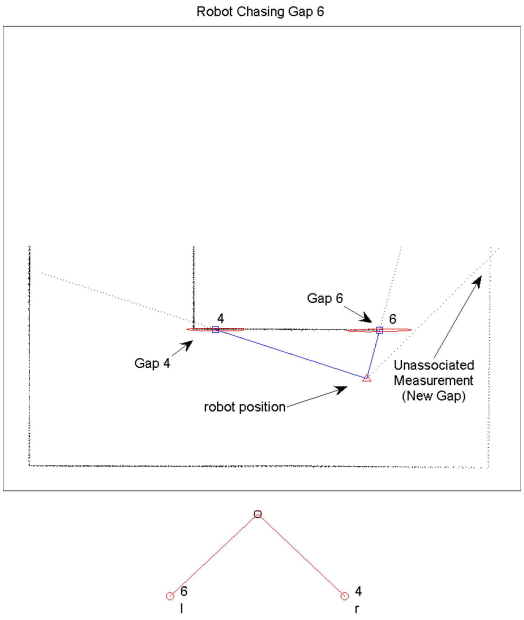
In Figure 4.15(c) the robot has turned around a second corner point but has still managed to track gap 4. Gap 4 now represents the entire occluded region of the known map located to the right of the tangent point. Note the altered shape of the covariance of gap 4 at this point - it is now located at a well-defined corner of the map with a greater point cloud density than in the previous examples.

Note that gap 4 is a *non-primitive* gap, meaning at least part of its occluded region is unexplored. In Figure 4.15(d) the robot is retracing its trajectory in pursuit of gap 4. As expected, the location of the tangent point to gap 4 travels back along the wall towards its previous position in Figure 4.15(b). The correct association between the region represented by gap 4 is maintained throughout Figures 4.15(a)-4.15(d).

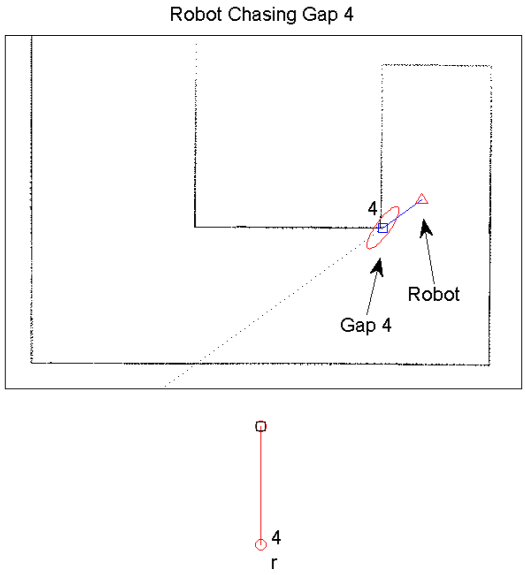
Figure 4.16 illustrates the need for a gap tracking capability. Our first implementation of the gap sensor used the *tangent point* as the sole means of identifying gaps as the robot moved through the environment, and primitively tracked them as the robot moved using their bearing relative to the robot to detect splitting and merging. While this worked satisfactorily as the robot moved up a straight corridor and the location of the gaps remained firmly anchored at the far corners of the corridor, this method failed as soon as the robot rounded a corner and the gaps started to move. It is not enough to attempt to track using the gaps' bearing relative to the robot as the incremental change in the bearing as the robot moves depends not only on how fast the robot is moving, but on its location relative to the gap and the density of the data surrounding the gap. In Figure 4.16 it is shown that a robot following the same trajectory as in Figure 4.15 fails to track gap 4 completely.



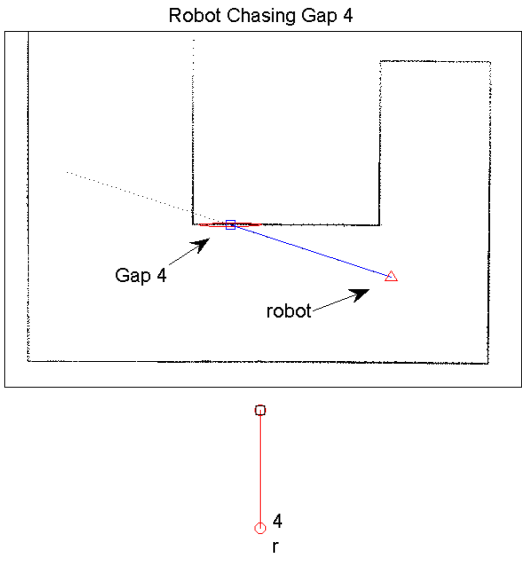
(a) Time step 1



(b) Time step 2

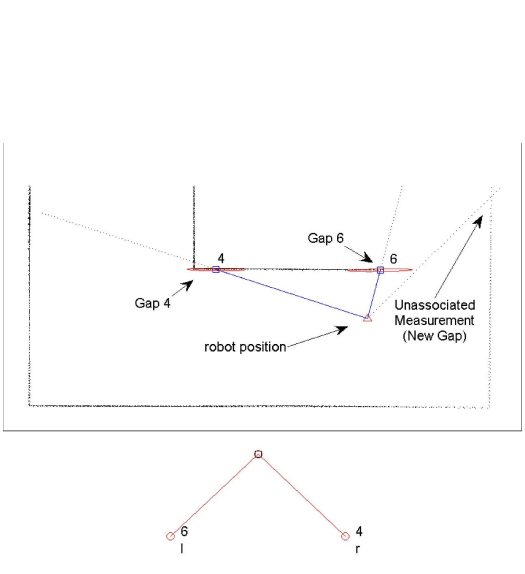


(c) Time step 3

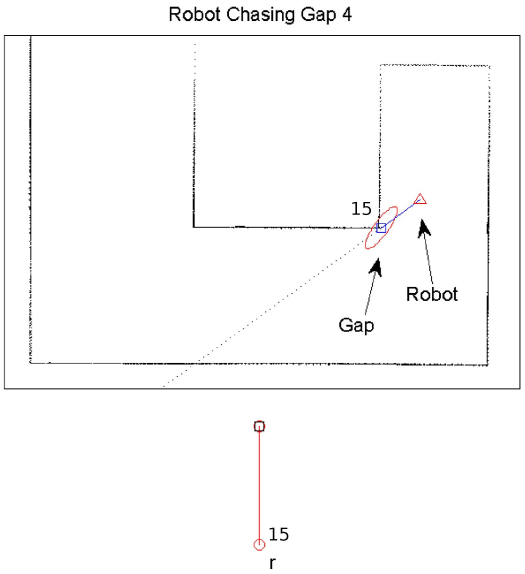


(d) Time step 4

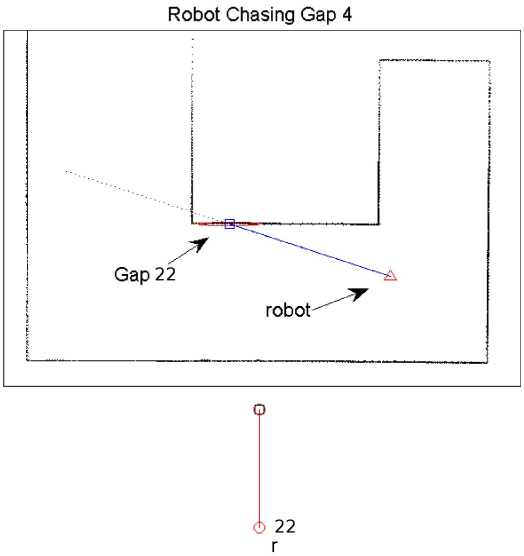
Figure 4.15: Exploring with the GNT using the gap tracker. Steps (a) through (d) follow the robot's progress through the lower part of the environment and illustrate the evolution of the topological tree.



(a) Time step 1



(b) Time step 2



(c) Time step 3

Figure 4.16: Exploring with the GNT, with an **imperfect** sensor based on gap position alone. Everytime the robot moves the location of what was originally gap 4 in the (a) changes. As a result, the gap corresponding to its previous location is thought to have disappeared and is deleted from the tree. A new gap is added to the tree corresponding to the new location, even it actually represents the same occluded area as its predecessor. As gaps that appear after the robot's initial sensing cycle are primitive nodes, using the gap navigation tree with such an imperfect sensor removes the driver for exploration.

4.8.2 Coverage exploration with the GNT

Figures 4.18 and 4.19 show the results of attempting to completely explore a simulated world (Figure 4.17 using the hybrid GNT. Here, we used processes from the MRG software code base to set up a simulated environment and to mimic our actual robotic platforms navigating in this environment, including the generation of noisy sensor outputs. We fed the output of the platform simulator into the *pNav* process for navigation. The example in Figure 4.18 also uses the *pSMSLAM* process for accurate navigation. Note the higher quality of the maps it produces compared with those in Figure 4.19, where navigation is performed on the basis of raw odometry alone. The GNT system was implemented in MATLAB, and we used the *iMATLAB* interface to the MRG software suite to communicate with the simulated platform. Both Figure 4.18 and 4.19 show the output of software tools written in MATLAB to display the map, robot and position of gaps as they evolve, together with the current state of the GNT.

While both cases we present here do exhibit exploratory behaviour and for the most part implement the functions of a gap sensor well, neither case actually succeeds in a bona-fide exploration of the environment guided perfectly by the GNT structure. In Figure 4.18(g) we fail to spot a critical merge event, which means the search terminates in Figure 4.18(t) with the top right portion of the environment left unexplored.

In the second case we made a minor amendment to the GNT algorithm to take into account the limited range of our sensor. This time, if a gap appeared in an area of the map we had not seen previously we classed it as a non-primitive node. Non-primitive status is usually reserved only for those nodes that the gap sensor detects on start up. Detecting whether or not a gap lies in an unexplored area is not a luxury afforded to the abstract gap sensor, but in our hybrid metric map implementation we have access to this capability. We found the addition of this step adds robustness

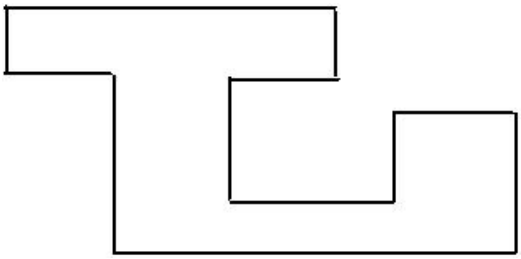
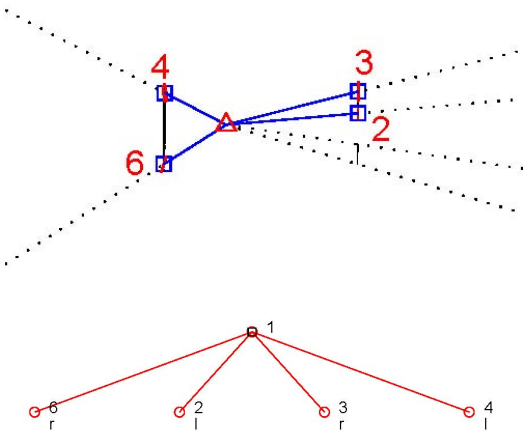


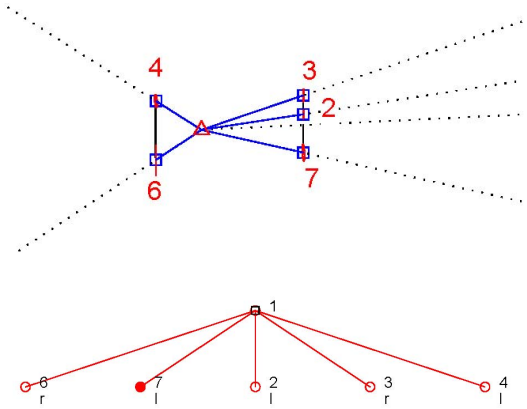
Figure 4.17: The environment under exploration in Figures 4.18 and 4.19.

to the algorithm. This is evidenced in the second example of Figure 4.19 where thanks to the presence of one of these 'late appearance' gaps, we are able to recover from a failure to spot a critical split event in Figure 4.19(j) and go on to completely explore the environment (Figure 4.19(n)) despite having an inaccurate topological representation in the tree. The effect of the addition of this fall-back clause is to add a default wall-following behaviour should all else go wrong.

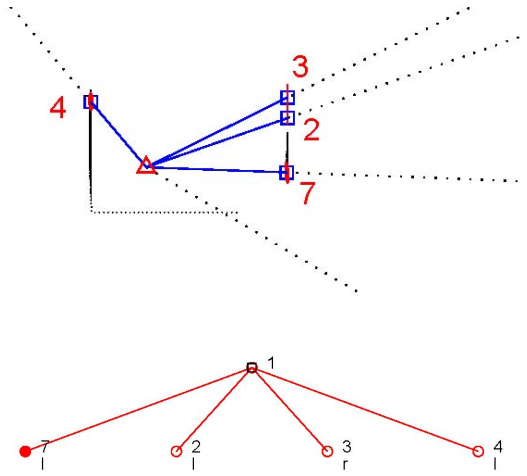
Both examples illustrate that our gap sensor works well most of the time. It represented a massive improvement over a deterministic implementation of the sensor which fails to take into account the gap motion and in our experience and that of Gordeski (2008), always fails to produce a correct tree. We can successfully track gaps across thresholds, and can detect appearance, disappearance, splitting and merging. In always choosing to follow the first gap found in an anti-clockwise direction from a heading which corresponds to a line drawn between the robot and the left edge of the page in our examples, the Gap Navigation Tree provided us with a smooth, sensible exploration strategy. Unfortunately, there is no robustness built into the algorithm itself to account for the gap sensor's very occasional failings - and this proved the undoing in the experiment presented in Figure 4.18, and the experiment in Figure 4.19 was only saved by our slight modification to the algorithm outlined above.



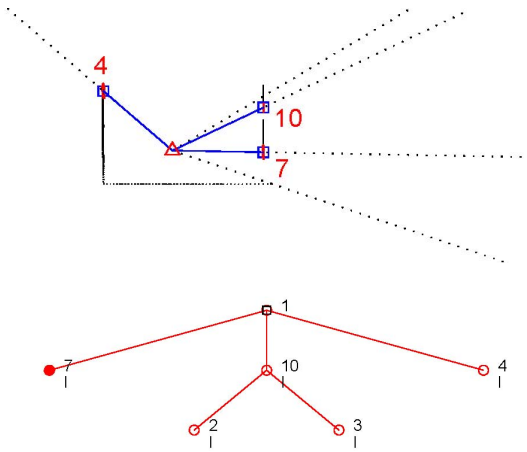
(a) Step 1 - The robot is switched on in the middle of the vertical corridor of the planar environment of Figure 4.17. It is situated in between 2 planar walls, the right hand wall is not fully sensed and hence here it has detected 2 persistent gaps (blue squares 2 and 3) and 2 other possible gaps (dotted lines). Two gaps lie at the extent of the left hand walls. We report the relative orientation of the gaps in an anti-clockwise direction starting from a line drawn from the robot to the left hand side of the page. The robot sets off in pursuit of gap 6.



(b) Step 2 - gap 7 has been seen enough by the robot to be declared persistent and is added to the tree. Still chasing gap 6.

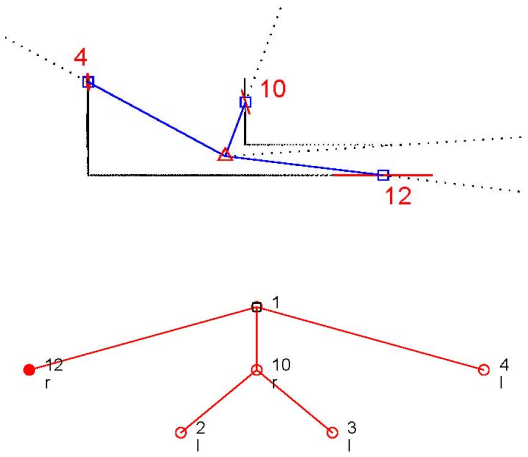


(c) Step 3 - The pursuit of gap 6 has led the robot to detect the bottom wall. Gap 6 disappears as a result. The chase switches to gap 7.

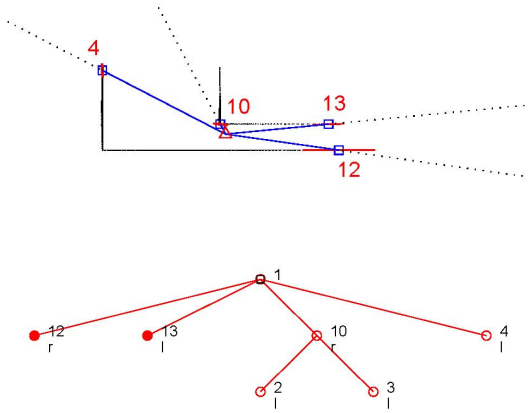


(d) Step 4 - As the robot moves towards gap 7 it positions itself more obliquely to gaps 2 and 3, they merge to form gap 10. More of the corridor to the bottom right of the image is now visible although the location of the gap at the end of the corridor continually changes and is not persistent enough for inclusion in the tree.

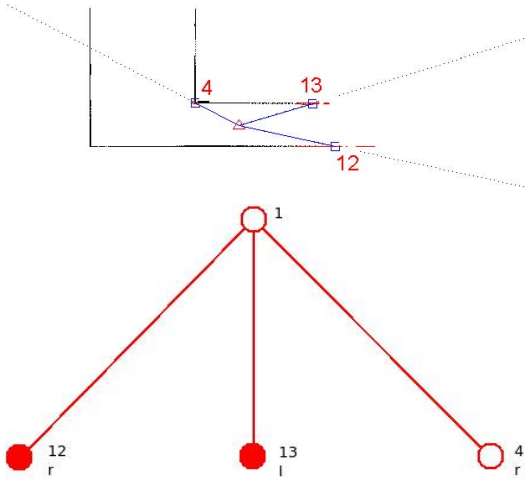
Figure 4.18: Completion properties of the GNT - example 1. Continued overleaf.



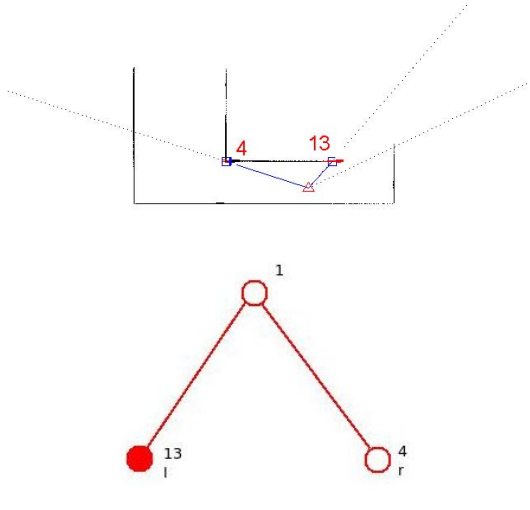
(e) Step 5 - Pursuit of gap 7 has led us to the position at which it disappears. The gap at the bottom right of the corridor has now been seen enough to be judged persistent and is added to the tree as gap 12. We start to detect a corresponding gap on the opposite side of that bottom corridor. We are now chasing gap 12.



(f) Step 6 - Still chasing gap 12, we see that the gap at the top of the corridor has been added to the tree as gap 13.

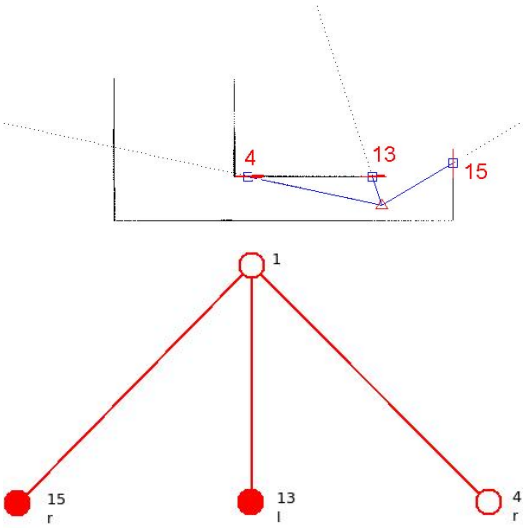


(g) Step 7 - An error case. Gap 10 has disappeared. This is actually an error as gap 4 and gap 10 (comprised of 2 and 3) represent different occluded areas as you can see by referring back to (a). Rather than disappear, gap 10 should have been merged with gap 4 to accurately represent the entire occluded area at the top of the image now obscured by the corner on which gap 4 lies. Without the merge, when the robot returns to this portion of the environment it will only chase down gap 4 and the associated area at the top left of the image and complete coverage will not be attained.

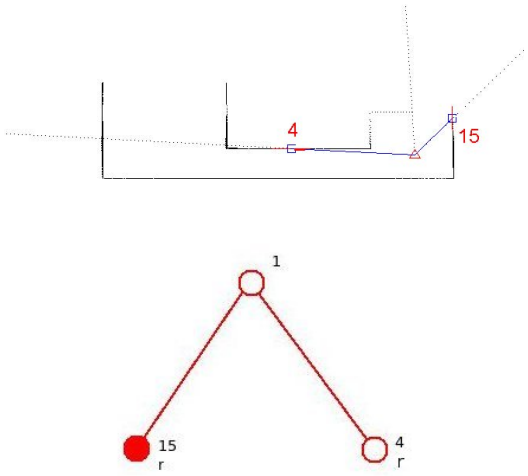


(h) Step 8 - The pursuit of gap 12 has led us to the end of the corridor where the detection of the back wall has meant that gap 12 has disappeared. Pursuit now switches to gap 13, and a new gap is in the process of being detected at the top of the right hand wall.

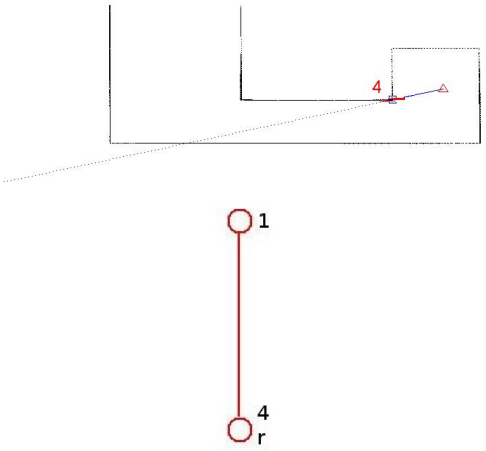
Figure 4.18: Completion properties of the GNT - example 1. Continued...



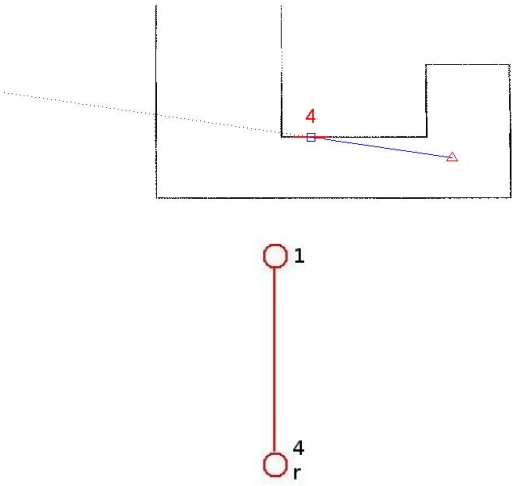
(i) Step 9 - Still chasing gap 13. The gap at the top of the right hand wall is persistent enough to be added to the tree as gap 15.



(j) Step 10 - We are now in the best position to view gap 13, and as expected this has caused it to disappear. The chase switches to gap 15.

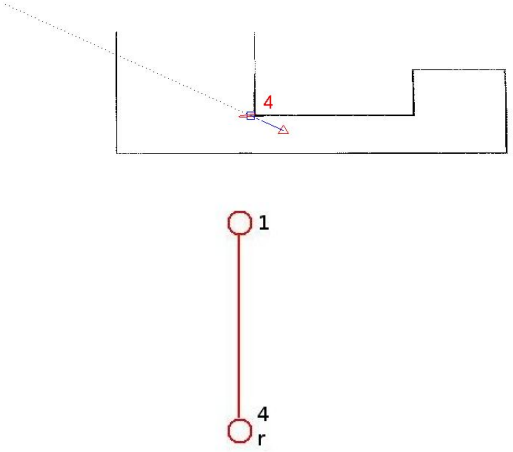


(k) Step 11 - The wall at the top of the right-hand extent of the image has now been fully detected and with it gap 15 has disappeared. With the bottom right hand part of the map now fully explored we return towards our starting point in pursuit of one of the original (non-primitive) gaps - gap 4.

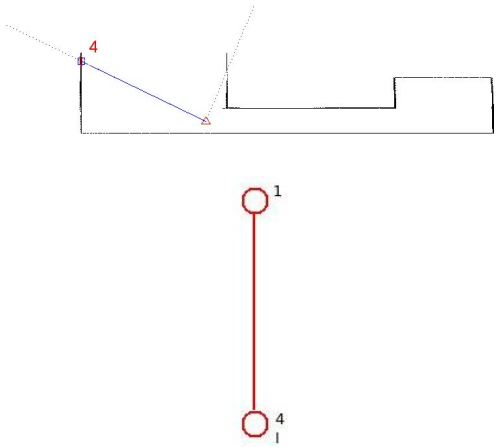


(l) Step 12 - Still chasing gap 4. Note how the covariance of the gap is elongated along the wall as we track the obscured region, and contrast this with its compact representation in the previous step when the gap was located on a corner.

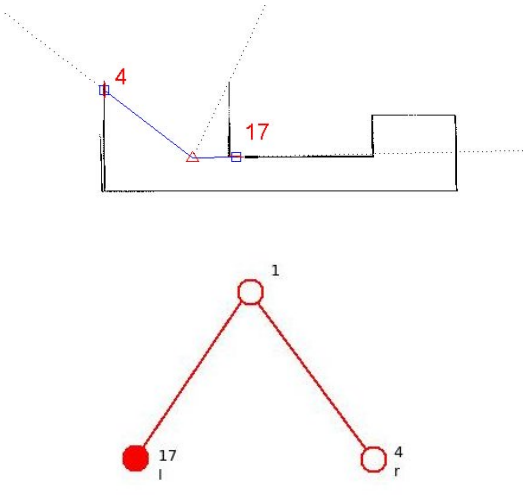
Figure 4.18: Completion properties of the GNT - example 1. Continued...



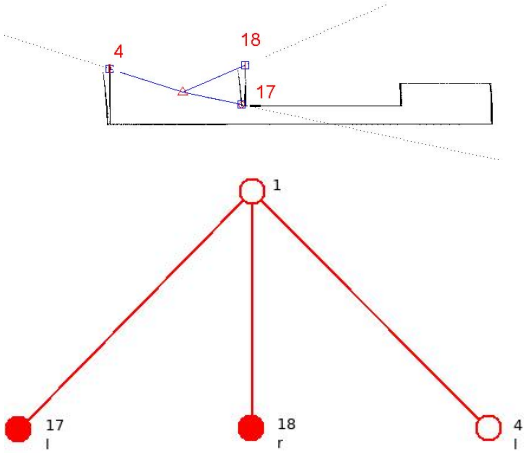
(m) Step 13 - gap 4 has now been successfully tracked along the extent of the wall to the far corner.



(n) Step 14 - As we move into the best position to view gap 4 it jumps discontinuously back to its original position (see (a)). Unfortunately as we failed to detect the merging of gap 4 and gap 10 in Step 7, the new, as yet unassigned gap will be treated as an appearing gap. Any gap that appears during the course of exploration (after the initial run of the gap sensor) is treated as primitive. This means that the occluded area off to the right of the vertical corridor is incorrectly treated as explored.

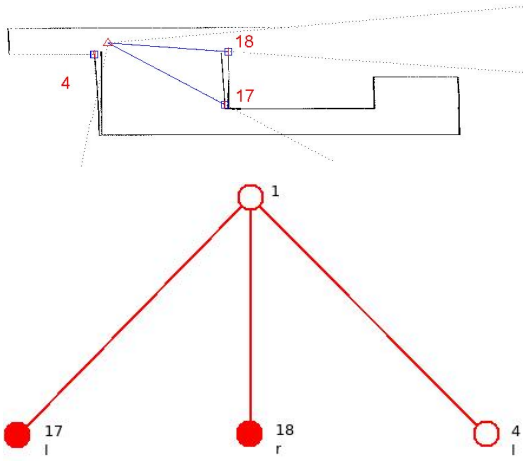


(o) Step 15 - Still in pursuit of gap 4. The area we have just traversed, off to the right of the image, is now obscured by the corner. The new gap that appears is added to the tree as a primitive node (shown as a filled circle).

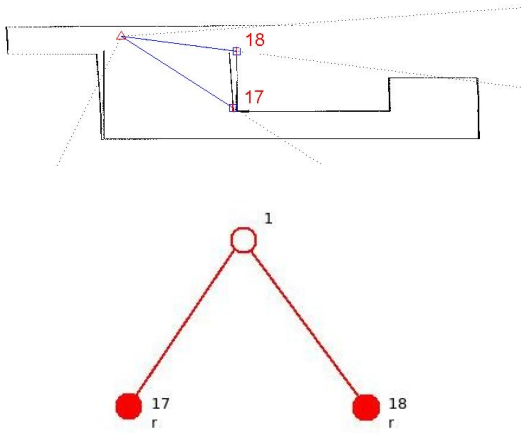


(p) Step 16 - gap 18 is added to the tree. This actually represents the area denoted by gap 3 back in Step 1, but due to the failure to detect the merge in Step 7 we add it to the tree here as a new, primitive node.

Figure 4.18: Completion properties of the GNT - example 1. Continued ...

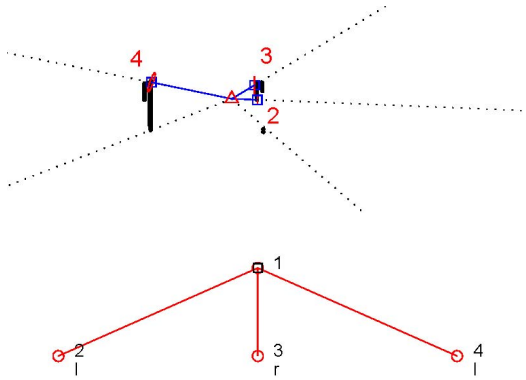


(s) Step 17 - As we move towards the best view position for gap 4, the top wall comes into view.

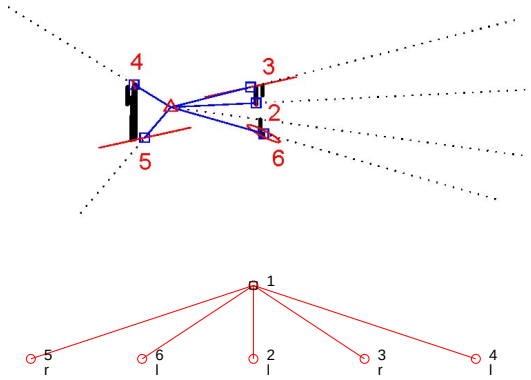


(t) Step 18 - gap 4 disappears. Only gaps 17 and 18 are left and these are primitive nodes supposedly representing already explored areas. Thinking that it has fully explored the environment, the algorithm quits.

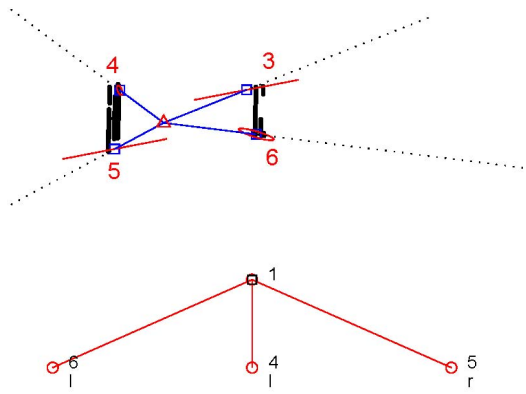
Figure 4.18: Completion properties of the GNT - example 1. While for the most part the gap sensor works well, we miss a crucial merge event in step 7. The Gap Navigation Tree is thus corrupted and the search terminates early, failing to explore the top right hand alcove which is the area that was occluded by the original (non-primitive) gap 3, subsequently merged into gap 10, whose merger with gap 4 we failed to detect. These figures were produced using a MATLAB program which used the *iMATLAB* interface to the MRG software suite to communicate with a simulated platform with on-board SLAM navigation system.



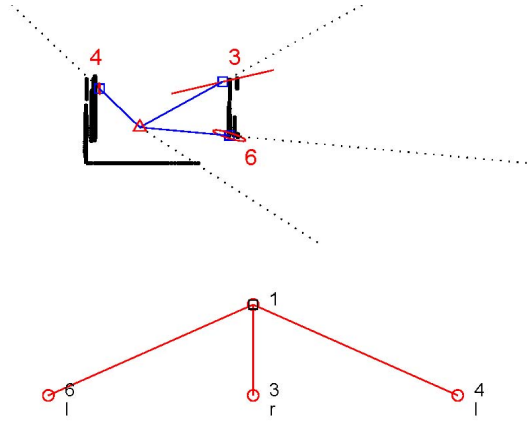
(a) Step 1 - On start up the robot sees 3 gaps (numbered blue squares), with a potential gap (denoted by the dotted line) being monitored. The robot sets off in pursuit of gap 2.



(b) Step 2 - The robot moves towards the bottom of the image. More of the right hand wall is detected. Gaps 5 and 6 - located in areas we haven't yet seen - are added as primitive gaps.

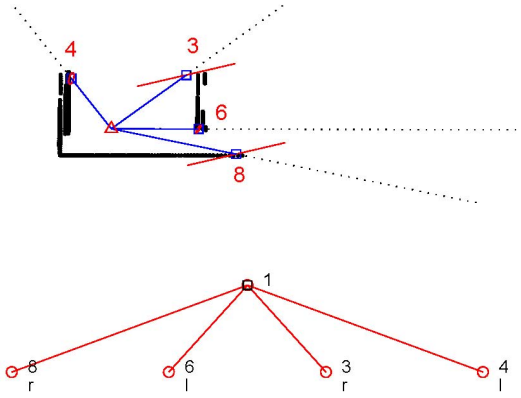


(c) Step 3 - As more of the right hand wall is seen the spurious gaps disappear. Gap 6 is now being chased.

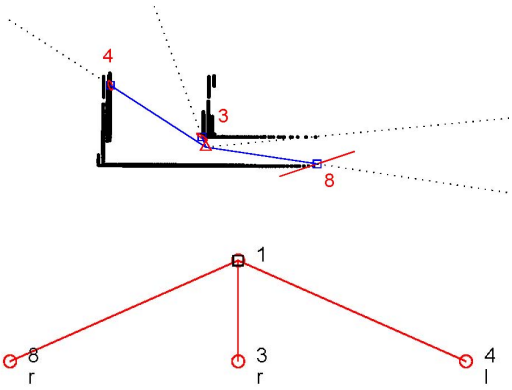


(d) Step 4 - As we chase down gap 6 we start to see more of the bottom wall of the image. A tentative gap appears on the bottom wall, but we have not yet seen enough of it to add it to the tree.

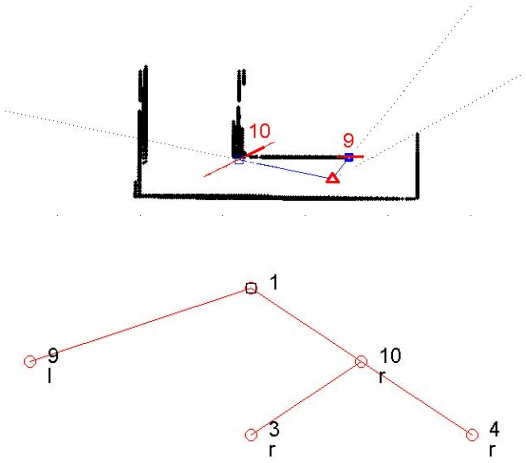
Figure 4.19: Completion properties of the GNT - example 2, continued overleaf.



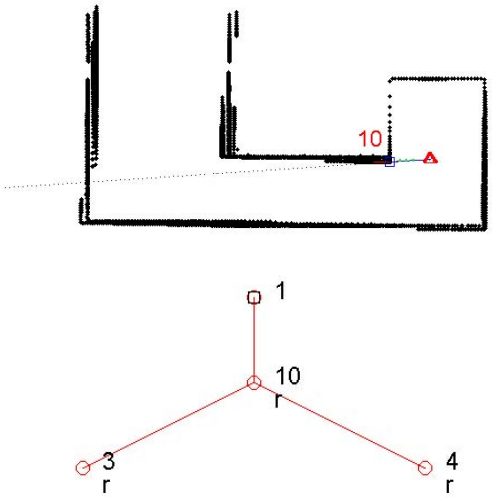
(e) Step 5 - Still in pursuit of gap 6, we now confirm the furthest extent of the bottom wall as gap 8 and add it to the tree.



(f) Step 6 - As we reach the viewpoint for gap 6 it disappears. The dotted line projects where it should be but there is no longer a gap in the map to track. We are now pursuing gap 8.

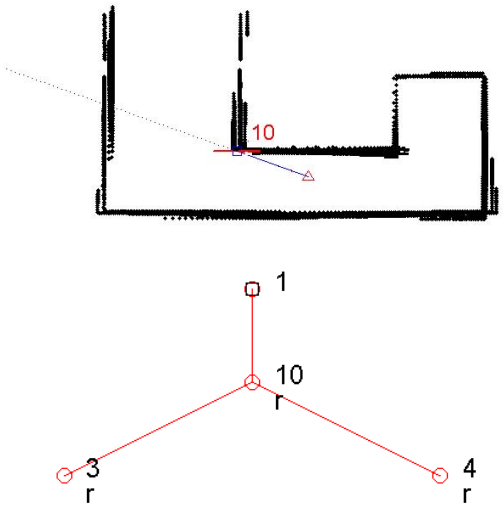


(g) Step 7 - As we move away from the corner, gaps 3 and 4 merge to form gap 10. Note that the failure to detect this merger proved the undoing of the example in Figure 4.18 - so this shows that the gap sensor can indeed detect merging successfully. As we reach the end of the bottom corridor, gap 8 disappears. We are now chasing gap 9.

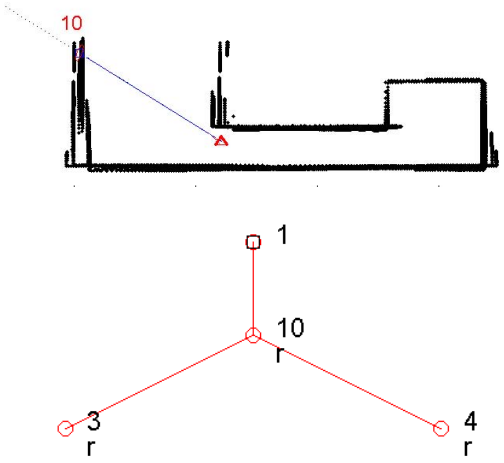


(h) Step 8 - The pursuit of gap 9 has led us to the position where we can sense all of the little alcove at the bottom right of the image. Gap 9 has been chased to the point of disappearance, but we have successfully tracked gap 10 as it has moved from one end of the corridor to the other. We now chase gap 10.

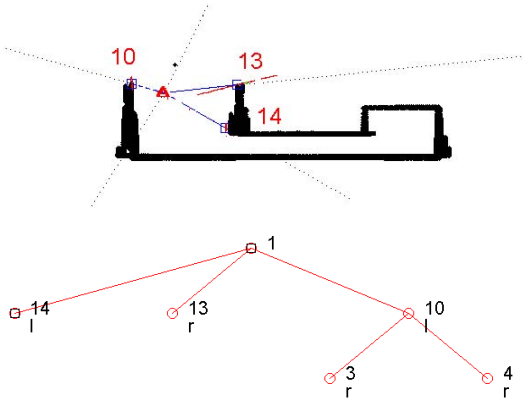
Figure 4.19: Completion properties of the GNT - example 2, continued...



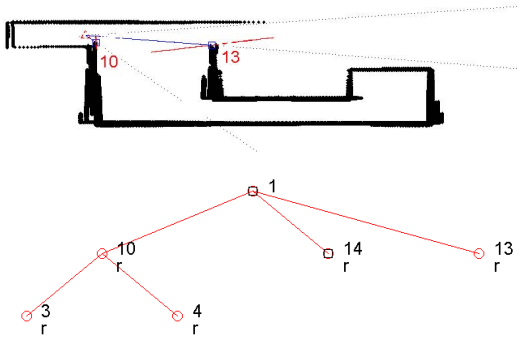
(i) Step 9 - We have successfully followed gap 10 back down the corridor.



(j) Step 10 - Error Case. Affected somewhat by the quality of the map (particularly the messy agglomeration of data points at the corner above the robot), here we only sense one gap when we should see two. This means we fail to detect what should be a split of gap 10 back into its component regions represented by gaps 3 and 4. From this point onwards the tree representation is incorrect.

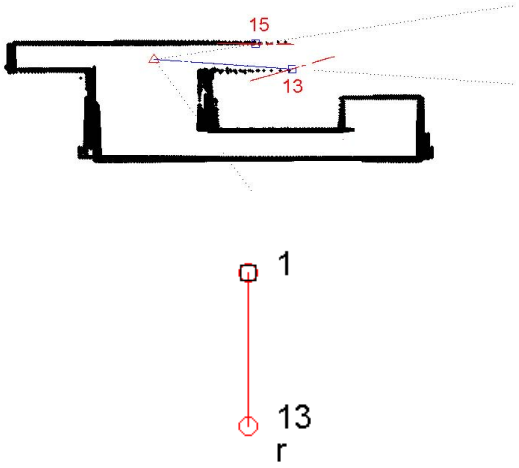


(k) Step 11 - Still pursuing gap 10 we move up the vertical corridor. The region we have just traversed is now represented by a primitive gap (square node in the tree).

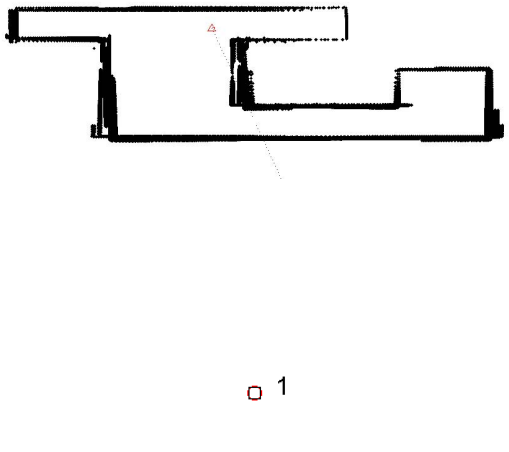


(l) Step 12 - We are now in the best position to view gap 10, but cautious updating of the tracker means that it has not yet failed to be detected enough to be removed from the tree.

Figure 4.19: Completion properties of the GNT - example 2, continued...



(m) Step 13 - After gap 10 disappears we set off after gap 13 (a non-primitive gap in this version). gap 15, although previously detected, has not matched the gap measurement recently so does not appear in the tree at this instant. It remains persistent in our application until it has failed to associate with a measurement enough times that we can safely disregard it.



(n) Step 14 - The slight modification to the GNT algorithm which meant that gaps appearing in unseen areas (a benefit of having an underlying metric map) means that in this case we successfully explored the entire area. This was despite missing a crucial split in Step 10, which in the vanilla GNT implementation would have curtailed exploration early, as it did in Figure 4.18.

Figure 4.19: Completion properties of the GNT, example 2. This time we made a slight modification to the algorithm which makes use of the fact that we have an underlying metric map. When new gaps 'appear' in the gap sensor, we first check if they correspond to an area of the map we have not yet visited before deciding whether they are primitive or non-primitive gaps. As this figure shows, in comparison to the previous example in Figure 4.18, this adds some robustness to the algorithm which means exploration can continue despite missing a critical gap event. These figures were produced using a MATLAB program which used the *iMATLAB* interface to the MRG software suite to communicate with a simulated platform. Navigation in this instance was performed using raw odometry (no SLAM system was running) hence the maps are of a lower quality than those in Figure 4.18.

4.9 Chapter Summary

Overall, we found the primary effect of adapting the GNT to real world conditions is the addition of *faux gaps* to the tree. They were not found to adversely affect the operation of the GNT, as they tend to be associated with simple geometric structures such as walls and simply disappear (rather than split) when chased. Gap 3 of Figure 4.15(a) is an example of such a gap, as the infinite range gap sensor of Tovar et al. (2007) at the robot position would detect the entirety of the wall on which gap 3 is located as well as the wall at the end of the corridor that is evident in Figure 4.15(c). No discontinuity would be ever be detected in the region of gap 3.

A major weakness of the algorithm is the dependency of the gap detector sensitivity on the structure of the local environment. Despite offering improved robustness over non-probabilistic implementations of the GNT, the data structure is still prone to failure as a result of failing to detect the splitting or merging of gaps. Successful detection is controlled by the threshold level γ described by equation 4.15, and its dependence on the robot's speed and characteristics of the local environment make γ difficult to estimate. A rollback/journaling facility could be added to overcome this and provide capability to maintain multiple hypotheses about the possible tree and environment structure at any one time. An alternative method of dealing with this problem is posed in Gordeski (2008), where a particle filter approach is used to maintain a distribution of likely trees. However, this approach is also fragile, and fails when more than one critical gap event occurs at one instant.

Additionally, in needing to specify a threshold for gap detection (see Section 4.5) we require the gap sensor to be fine tuned for each environment in which it operates, to take into account the average widths of corridors and door sizes. The rate of update of the underlying map, the quality of the map and the speed of movement of the robot all affect the ability to spot critical events and so the gap sensor also relies on estimating the effect of these parameters on gap motion correctly.

Overwhelmingly our simulations showed the Gap Navigation Tree to be too fragile for reliable use on a fielded robotic system. We were unable to guarantee the correct topological structure of the tree and hence could not guarantee its benefits over wall following, random exploration or frontier-based search as a simple, lightweight exploration strategy.

In this chapter we discussed the mechanism for providing a robot with true autonomy - equipping it with the means to decide 'where to go next?'. We provided a brief background to the exploration problem, illustrating that exploration is often tightly coupled with the underlying navigation system whose job it is to provide the answer to 'where am I?'. A hybrid exploration technique was introduced which coupled an existing topological exploration algorithm - the Gap Navigation Tree, with the existing SLAM systems developed by the Mobile Robotics Groups. Although ultimately judged too fragile for reliable, continued use on an autonomous robot, our implementation of a probabilistic gap sensor to aid exploration with the GNT proved much more successful than a simple deterministic approach and went some considerable way towards realizing the ideal 'abstract sensor' required by the algorithm.

Having discussed how a robot might go about choosing where to go next, in the following two chapters we revisit the path planning problem and will discuss probabilistic-based strategies to get a robot from A to B.

Chapter 5

Probabilistic Cost Maps - Enabling Planning with Uncertainty

In previous chapters we have discussed algorithms for planning paths for a mobile robot operating outdoors. Although this field has seen much progress in the past few years, the question of how to deal with uncertainty in the robot's knowledge of the environment is still very much an open one. Common to the algorithms in Chapter 2 is a grid based representation of the environment and the need to associate a traversal cost with each cell. In the absence of perfect sensing and classification, this cost is actually an uncertainty distribution over the terrain associated with a particular cell. The various approaches to generating cost maps and hence planning differ in how they choose to deal with this distribution. In this chapter we address the problem of obtaining sensible distributions and then using the full distribution to plan viable paths.

5.1 Motivation

The volume of information presented to a robot operating outdoors is such that it can only feasibly plan over a simplified representation of its environment. In Section 2.1.3.2 we saw how a principled probabilistic approach to handling uncertainty in sensing and actuating is available through POMDPs, but this technique remains

computationally intractable for outdoor robot navigation at present. The accepted way to plan outdoors remains the assumptive method, wherein we discretize the environment into a grid and create a costmap. Each grid cell takes on a scalar cost proportional to parameters such as robot stability, visibility, probability of presence of an obstacle, energy cost of traversal, etc. Because it is easier and faster, it is typical to simplify the cost of driving over grassland to 42 when it may be anywhere between 35 and 61! (These numbers are for the purposes of example only). Nevertheless, what we are really doing in existing approaches is seeking to simplify what is actually a distribution, and in doing so we throw away potentially valuable information. In this chapter we attempt to combine the fast graph search technique of A* with full knowledge of the uncertainty in the terrain cost assigned to the cell.

While the vast majority of the literature dealing with the creation of cost maps (Bradley et al., 2007; Konolige et al., 2006; Gerkey and Konolige., 2008; Vandapel et al., 2004) uses onboard sensors to create cost maps online, there is a growing body of literature (Silver et al., 2006, 2008; Vandapel et al., 2003) that demonstrates the benefit of using a-priori overhead data in path planning. We are not advocating a move away from sensing and replanning in this work. On the contrary, what we examine here is a method of pre-arming a robot with a set of likely paths between two waypoints it wants to visit, using aerial maps. Since the evolution of Google Maps, Bing Maps etc aerial imagery has become ubiquitous and it makes sense to utilize this freely available data to simplify the planning task. The paths generated from aerial imagery could act as an "oracle" for an autonomous traverse task. It provides the robot with an initial direction in which to explore which has been evaluated as most likely to be successful given the prior data. Should this initial traverse fail then subsequently ranked paths can be drawn from the distribution of paths to give the robot a useful restarting point.

Coming up with cell costs from aerial imagery presents two difficulties common to

the costmap construction task in general. First, we must decide which of the terrain types under consideration is best aligned with our cell. Secondly, we must associate a cost with each terrain type. The difficulty in the first is illustrated in Figure 5.1 where open terrain is interleaved with woodland. It would be possible to drive a car through this terrain but training a classifier to correctly discriminate between trees and open space in a low resolution image as typically obtained from aerial imagery is a difficult task. In Figure 5.2 we see the problems with the latter point. Here we have two images of dirt road in a similar area, however one has been washed away and would clearly require much more time and energy to traverse and hence requires a higher traversal cost. So within one map, the cost of traversing a particular terrain type might vary depending on the robot's location in the map.



Figure 5.1: Uncertainty in Classification: Forest or Open Terrain? Traversable or Not?



(a) High Traversal Cost



(b) Low Traversal Cost

Figure 5.2: Spatial Variation in Terrain Cost: Dirt Roads

The technique presented in this chapter describes a framework for incorporating both the uncertainty in the cell classification, and the potential for terrain costs to vary spatially, into the cost attributed to an individual cell.

Figure 5.3 provides a block diagram of the process, while Figure 5.4 shows a pictorial overview. First, we decide on a fixed number of terrain types we think are represented in our map (in this case *road*, *park* and *buildings*). Then we decide on a resolution for the map and divide the map up into a corresponding number of cells. We use a Gaussian Process classifier to obtain the probability of membership of each of the n classes represented in the map. In a separate operation, we assume we have data on how the cost of a terrain type varies spatially over the area represented by our map. We use a Gaussian process regressor to compute the cost of that particular terrain type at the given location and obtain a distribution of the likely cost at that grid cell.

The probability of class membership and the individual class cost are then combined in a cost function. The resulting cost map can be sampled during planning to produce a distribution of likely paths between points A and B in the map or the mean of the cells can be used to produce a most-likely path.

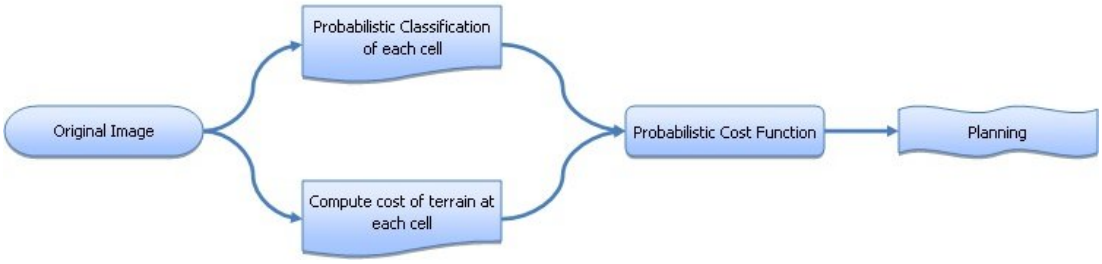


Figure 5.3: A Block Diagram of the Costmap creation process

5.2 A Probabilistic Planning Framework

Imagine we are given an aerial image of the area over which we wish to plan. We are going to use the aerial data to classify each grid cell as one of n terrain classes, so in order to do this we require some labelled training data which relates features of the image (e.g. x,y location; r,g,b color; hyper spectral data if available) to a single class

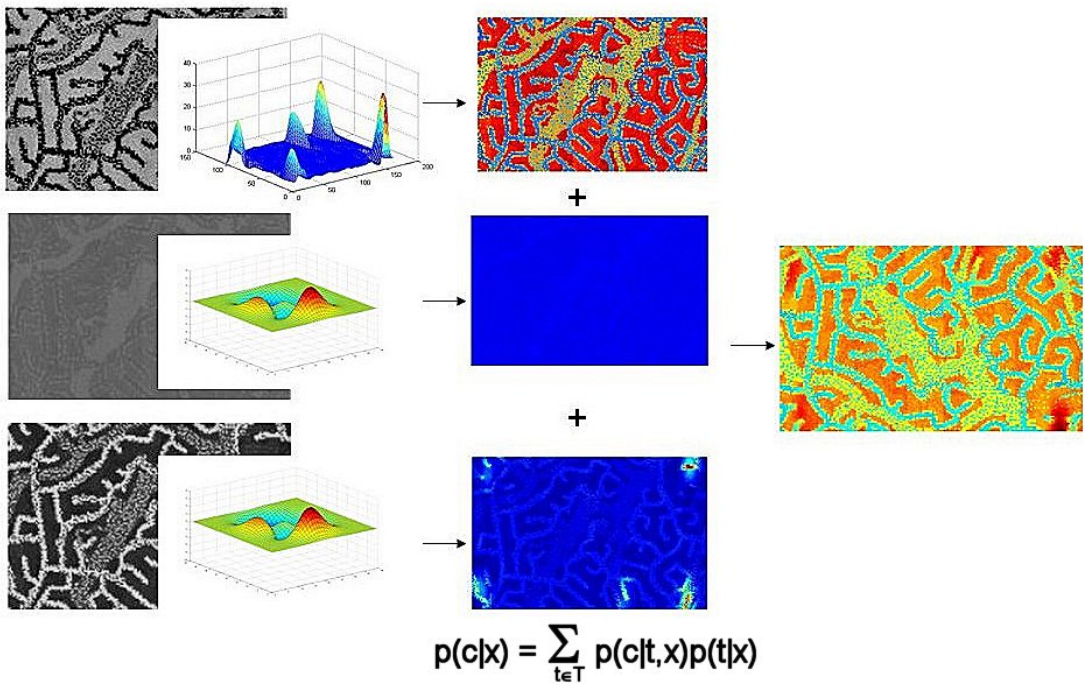


Figure 5.4: Overview of the costmap creation process.

label.

Independent of this we are given a priori cost data for the region for *each* of the n terrain classes. The reason for approaching the problem this way is that it frees the planner from having to know the class of a cell or area before placing a prior on traversability. We are hence able to incorporate vague information only relevant to a particular class(es) into our cost map. Examples of this would include *'the area to the south west is swamp land'*, which would affect the cost of anything classed as open terrain, but not the cost of roads or impassable obstacles such as houses. Likewise *'all roads around London are congested'* affects only the cost of roads, not open terrain or buildings. Attributing a separate spatial Gaussian Process (GP) to each class allows us to model the entire map from a much smaller representative set of training data than would be required if we were attempting to model all classes together using just one GP. As training data we assume some prior knowledge of the blanket cost of a

particular terrain type together with some local spatial variations.

We begin by discretizing the environment into a grid. This can be done at the pixel or super pixel level. We classify the grid cell using the aerial imagery to obtain $p(T|x)$, the probability of terrain class membership at that location. Here we use T to represent the terrain class, and x is the location in the map. Separately, we evaluate our spatial Gaussian process regressor for each of the n classes to obtain $p(c|T, x)$, the probability distribution function representing the traversal cost c *of that particular class* at the given location. Because our spatial GPR can be viewed as modeling a cost-contour map for each class we assume that a smoothing between data points is appropriate and hence used a squared exponential covariance function for the regression.

Next, the cost function combines the terrain class membership and terrain cost at a point x in a probabilistic form, integrating out over the class types to produce the overall cost of the cell.

$$p(c|x) = \int_T \underbrace{p(c|T,x)}_{GPR} \underbrace{p(T|x)}_{GPC} dT \tag{5.1}$$

$$= \sum_{t \in T} p(c|t,x)p(t|x) \tag{5.2}$$

Finally, we pass the information in the cost function to a planner. Because our cost function is a probability distribution over terrain cost at each grid cell location $\{x, y\}$ we are able to not only produce the most likely shortest path between point A and point B, but by repeatedly sampling the cost function we may also produce a likely distribution of paths between the two points.

5.3 Required Tools

Building this probabilistic cost map therefore requires two tools. The first is a classifier which takes sensory inputs and transforms this to a terrain type at a given location in the map. A wide ranging array of classification techniques have been applied to this problem. We begin this section with a discussion of why we preferred to use Gaussian Processes to perform the regression and classification tasks, and include a brief discussion other probabilistic classification and regression techniques and why we chose not to use them.

5.3.1 Why Gaussian Processes?

There are numerous approaches to tackling the problem of probabilistic classification and regression. However, this thesis has made exclusive use of Gaussian Processes for probabilistic learning and would not be complete without a discussion as to why this choice was made.

In lieu of the detailed treatment to follow in Section 5.4, Gaussian processes are a discriminative model which aims to predict $p(y|x)$ by placing a prior directly over the space of functions $p(y)$, without parameterizing $p(y)$. In order to construct a model and predict new values using a Gaussian Process, the user need only specify a kernel (covariance function) that describes how data points in the input space vary with respect to one another. From the data, a limited number of *hyperparameters* necessary to specify the covariance can be trained, and prediction can begin. The only model the user need prescribe is that of the covariance function.

5.3.1.1 Why Use Gaussian Processes for Terrain Classification?

We contrast the Gaussian Process approach to classification with other approaches. Support Vector Machines (SVMs) (Cortes and Vapnik, 1995) provide a supervised,

discriminative approach to classification and arguably offer the most popular approach for 'off the shelf' supervised learning (Russell and Norvig, 2010). They operate by attempting to construct the optimal separating hyperplane so that instances of class c_1 occupy a different subspace from those of class c_2. If the hyperplane cannot be found in the original space occupied by the data, the 'kernel trick' (Aizerman et al., 1964) is used to map the data to a higher dimensional space in which it is separable. The *Support Vectors* are those training data points that lie closest to the decision boundary and hence have influence on the positioning of the line.

SVMs have been widely used in robotics for terrain classification (Posner et al., 2007; Weiss et al., 2006; Mou and Kleiner, 2010). However, they have two major downsides. The first is that, in contrast to the other methods we describe, they do not provide any kind of posterior probability (although an additional sigmoid function can be trained to map the output of an SVM to probabilities (Platt, 2000)). This means there is no level of confidence expressed in the classification. The second is that multi-class classification is only supported through the use of many binary one-vs-many discriminators.

Graphical Model based techniques such as Bayes Networks, Markov Random Fields (MRFs) and Conditional Random Fields (CRFs) are a prevalent probabilistic approach to terrain classification in robotics. Here a graph is used to encode a probability distribution, and each node represents a random variable. The challenge is to work out the values of the hidden nodes given the observed nodes. The edges in the graph can be directed or undirected, giving rise to different graphical models and a range of different machine learning algorithms to solve them.

A simple Bayes network is used in Sofman et al. (2006b) to model the dependencies between features, parameters and terrain traversability estimates. In Bajracharya et al. (2009) a short range classifier based on geometric properties of the immediate surroundings is built using a naive Bayes classifier. It is complemented by longer

range colour based SVM classifiers.

In Vernaza et al. (2008) MRFs were used to perform stereo image classification on the LAGR ground robot and were also used in Dolgov and Thrun (2008) to discern the principal driving direction in Car parks and other unstructured terrain during the DARPA Urban Challenge. Posner et al. (2008) use MRFs to model the relationship between local patches (super pixels) in a scene and give weight to the local classifier's prediction of the presence of a class in the image.

However, given a typical problem addressed by this thesis, exact inference over the model is usually computationally intractable and approximate techniques such as sampling, variational methods, and loopy belief propagation are most often employed. These solution techniques are more complicated and time consuming than that offered by Gaussian Processes. In addition, specifying the graphical model equates to making strong prior assumptions about the data and how the relevant variables are related. This is not required when using Gaussian Processes.

Gaussian Mixture Models (GMMs) are a means of unsupervised learning and a popular form of a group of algorithms known as 'unsupervised clustering'. Instead of labeling the training data, we assume that there are k classes and attempt to fit a combination of k Gaussian distributions which best describe the data. GMMs were used with great success by the winning entry in the DARPA Grand Challenge (Thrun et al., 2006) to differentiate between the desert road and everything else. However, GMMs have their downsides in that there is no inbuilt penalty in the learning process that prevents a Gaussian component from shrinking to cover just a single data point, or preventing two components from merging. For this reason we favour the supervised learning approach of Gaussian Processes.

In summary, Gaussian processes offer a flexibility in approach to modeling the data that none of the above methods can offer. There is no need to specify conditional dependencies amongst the data, parameters, observations and labels. This is all

handled internally by the scheme. Like SVMs, Gaussian Processes invoke the kernel trick which allows the functions described by the process to have infinite support (basis), and Bayesian inference is done in this latent function space. The kernels are additive, so modular, functional kernels can be constructed intuitively (i.e. a periodic kernel can be added to a smooth one to add seasonal variation in predicting climate change). Desirably from the point of view of this application, Gaussian Processes also return a probabilistic prediction, in the form of a Gaussian, fully described by its mean and variance.

5.3.1.2 Why Use Gaussian Processes for Terrain Costing?

The second tool we require is a means of mapping terrain types to cost. This cost can be proportional to the time spent traversing that terrain type with respect to others. It may be the relative fuel cost of that terrain relative to others, or it could express the added mechanical vibration experienced by the vehicle when traversing such terrain relative to the other terrain types under consideration. Most often this is done by hand, for example in Sofman (2009) it is described how the Unmanned Ground Vehicles operated by Carnegie Mellon University work off traversal costs in the range 16 - 65535. The minimum (16) is assigned to *roads*, *grass* warrants a 48 and *dense vegetation* is assigned something in the range 10 000 plus. There is a gradual move away from this ad hoc approach towards having the robot learn online the cost of terrain through vibration sensors (Komma et al., 2009) or by attempting to explore obstacles (Gerkey and Agrawal, 2008).

Our approach to the problem of determining costs from terrain type is novel. Instead of associating a single value with each terrain class, what we do is assign a spatially varying function (a map) with each terrain class under consideration. Known terrain costs at certain locations are placed in the map. A spatial Gaussian process is used to interpolate and extrapolate the map, smoothing the costs between

locations and responding with high certainty in the estimates close to where readings have been taken and low certainty elsewhere. As with the classification task, we felt the simplicity and flexibility of Gaussian Processes over comparable probabilistic regressors best suited the task at hand.

5.4 Gaussian Process Modeling

Gaussian process modeling is just one way of performing empirical modeling of relationships in high-dimensional data using a Bayesian framework. Here we assume that a nonlinear function $f(\mathbf{x})$ underlies the data $\{\mathbf{x}^{(n)}, y_n\}_{n=1}^N$. We are given a set of input vectors $X \equiv \{\mathbf{x}^{(n)}\}_{n=1}^N$, and their corresponding target values denoted by $\mathbf{y} \equiv \{y_n\}_{n=1}^N$. If these targets are real numbers we are seeking to model a regression or interpolation task, while if they are categorical labels such as $y \in \{0, 1, 2, 3\}$ then the problem represents a classification task. We can infer $f(\mathbf{x})$ as the posterior probability distribution

$$P(f(\mathbf{x})|\mathbf{y}, X) = \frac{P(\mathbf{y}|f(\mathbf{x}), X)P(f(\mathbf{x}))}{P(\mathbf{y}|X)}. \tag{5.3}$$

In Gaussian Process Modeling, we seek to do this inference directly in *function space*. The second term on the right hand side $P(f(\mathbf{x}))$ is the prior distribution of functions assumed by the model. As it turns out, the simplest type of prior we can place over functions is called a *Gaussian Process*.

Gaussian Processes extend multivariate Gaussian distributions to infinite dimensionality. So while a Gaussian distribution specifies a probability distribution over a random variable that is a scalar or vector, a Gaussian Process specifies a distribution over functions.

Just as a Gaussian distribution is fully specified by a mean and a variance, a Gaus-

sian Process can be fully described with a *mean function* and a *covariance function*:

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})], \tag{5.4}$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))]. \tag{5.5}$$

Hence we can write the Gaussian process as

$$f(\mathbf{x}) = \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')). \tag{5.6}$$

Most often, it is assumed that the mean of the GP is zero everywhere. In the absence of any prior knowledge this is not a ridiculous assumption to make, and it does not restrict the mean of the *posterior* process to be zero. However, it is not unusual to specify the mean as a non-zero constant for all inputs, or as a non-constant polynomial of some order (Rasmussen and Williams, 2006).

In the case of the zero mean function, it is only the covariance function that relates one observation x to another x' and encodes our assumptions about the underlying function we wish to model. It is the covariance function that determines the properties of sample functions drawn from the Gaussian process distribution. *Stationary* covariance functions are functions of $\mathbf{x} - \mathbf{x}'$ and are invariant to translations in the input space. An *isotropic* covariance function is a function of $|\mathbf{x} - \mathbf{x}'|$ and is invariant to all rigid motions. *Non-stationary* covariance functions allow the model to adapt its smoothness depending on the inputs. Note that the covariance between outputs is actually specified as a function of the inputs.

A common example of a covariance function is the *squared exponential* covariance function

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp[\frac{-(\mathbf{x} - \mathbf{x}')^2}{2l^2}]. \tag{5.7}$$

It has 2 *hyperparameters*. The maximum allowable covariance is given by σ_f^2. For

functions that have a broad range on the y-axis this parameter should be large. If $\mathbf{x} \approx \mathbf{x}'$ then $k(\mathbf{x}, \mathbf{x}')$ approaches this maximum. The length parameter l determines the 'sphere of influence' one input has on another. If $\mathbf{x}$ is a long way from $\mathbf{x}'$ then $k(\mathbf{x}, \mathbf{x}')$ approaches zero.

To see the effect of the covariance function on the Gaussian process prior in Figure 5.5 below we draw multiple functions from different priors all generated with a squared exponential covariance function.

Typically however, the data we observe y is noisy due to measurement errors and sensor noise. So our observations y are related to the underlying function $f(x)$ we wish to model through a Gaussian noise model:

$$y = f(\mathbf{x}) + \mathcal{N}(0, \sigma_n^2). \tag{5.8}$$

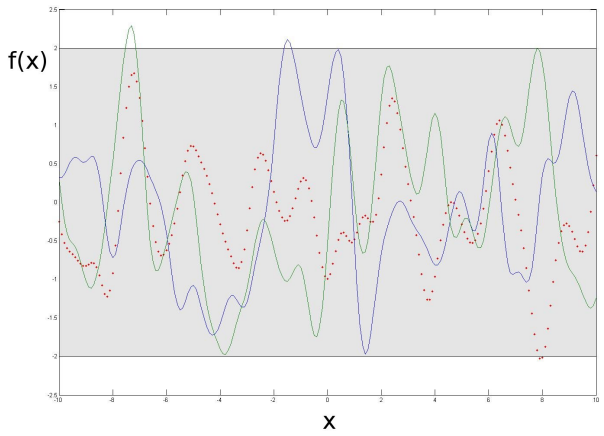
For simplicity, we fold the noise into the covariance function

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left[\frac{-(\mathbf{x} - \mathbf{x}')^2}{2l^2}\right] + \sigma_n^2 \delta(\mathbf{x}, \mathbf{x}'). \tag{5.9}$$

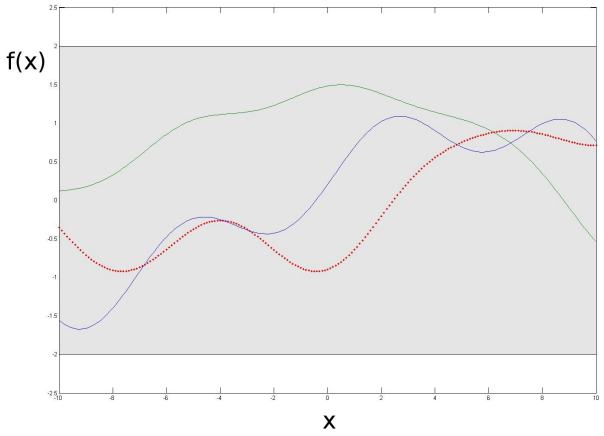
Here $\delta(\mathbf{x}, \mathbf{x}')$ is the Kronecker delta function which is one iff $\mathbf{x} = \mathbf{x}'$ and zero otherwise.

In essence, what we are trying to predict, and this holds for both regression and classification, is not the actual *underlying* function f_* but the target values y_*. According to equation 5.8 their expected values are the same but their variances will differ owing to the observational noise.

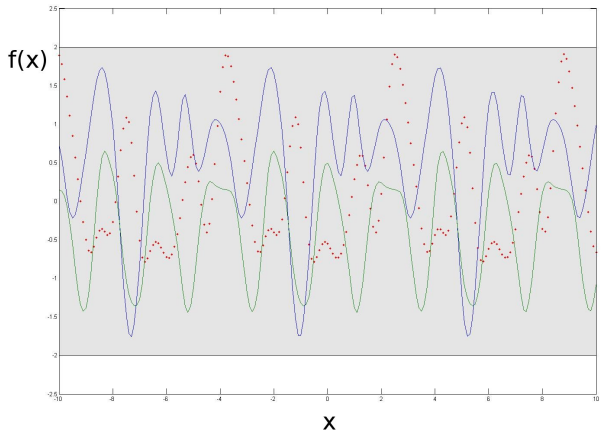
To prepare for regression and classification, we calculate the covariance function amongst all possible combinations of observed and target values; namely the covariance between the observed values and all other observed values ($K(X, X)$), the covariance between the observed inputs and the target inputs ($K(X, X_*)$) and the covariance between target inputs (($K(X_*, X_*)$).



(a) Squared Exponential: Short Length Scale (0.5)



(b) Squared Exponential: Long Length Scale (2.0)



(c) Periodic Covariance

Figure 5.5: Covariance Functions - Note how the choice of covariance function influences the shape of functions which constitute the prior

$$K(X,X) = \begin{bmatrix} k(\mathbf{x}_1,\mathbf{x}_1) & k(\mathbf{x}_1,\mathbf{x}_2) & \dots & k(\mathbf{x}_1,\mathbf{x}_n) \\ k(\mathbf{x}_2,\mathbf{x}_1) & k(\mathbf{x}_2,\mathbf{x}_2) & \dots & k(\mathbf{x}_2,\mathbf{x}_n) \\ \vdots & \vdots & \ddots & \dots \\ k(\mathbf{x}_n,\mathbf{x}_1) & k(\mathbf{x}_n,\mathbf{x}_2) & \dots & k(\mathbf{x}_n,\mathbf{x}_n) \end{bmatrix} \tag{5.10}$$

$$K(X,X_*) = \begin{bmatrix} k(\mathbf{x}_*,\mathbf{x}_1) & k(\mathbf{x}_*,\mathbf{x}_2) & \dots & k(\mathbf{x}_*,\mathbf{x}_n) \end{bmatrix} \tag{5.11}$$

$$K(X_*,X_*) = \begin{bmatrix} k(\mathbf{x}_*,\mathbf{x}_*) \end{bmatrix} \tag{5.12}$$

The full covariance matrix is represented by:

$$K = \begin{bmatrix} K(X,X) & X(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}. \tag{5.13}$$

5.4.1 Gaussian Process Regression

In the previous section we discussed how the covariance function determines the shape of functions that constitute our Gaussian Process prior. The prior knows nothing about the training data, it only specifies properties of the sort of functions that could be used to fit to the training data. In Gaussian Process regression we update the prior in terms of the training data $\mathbf{y}$, and compute the *posterior* function which can be used to make predictions for unseen test cases.

We can write the joint distribution of the observed target values $\mathbf{y}$ and the function values at the test locations under the prior f_* as

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X,X)+\sigma_n^2 I & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right). \tag{5.14}$$

Because we know the values of the training set y we are interested in the conditional distribution of the test set f_* given the training data y. We can apply the

formula for conditioning a joint Gaussian distribution

$$\begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix}, \begin{bmatrix} \mathbf{A} & \mathbf{C} \\ \mathbf{C^T} & \mathbf{B} \end{bmatrix} \right) \tag{5.15}$$

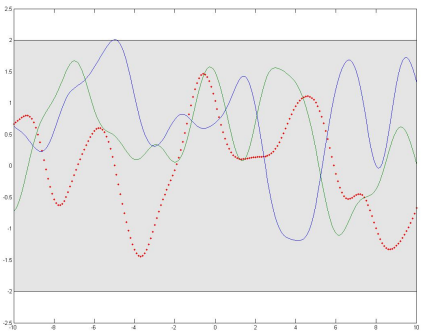
$$\rightarrow \mathbf{x}|\mathbf{y} \sim \mathcal{N}(\mathbf{a} + \mathbf{CB^{-1}}(\mathbf{y} - \mathbf{b}), \mathbf{A} - \mathbf{CB^{-1}}(\mathbf{y} - \mathbf{b}), \mathbf{A} - \mathbf{CB^{-1}C^T}) \tag{5.16}$$

to arrive at the expression for the the conditional distribution of f_* given f

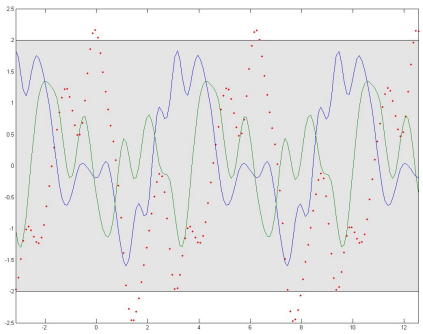
$$f_*|X, \mathbf{y}, X_* \sim \mathcal{N}\big(K(X_*, X)[K(X, X) + \sigma_n^2 I]^{-1}\mathbf{y}, \\ K(X_*, X_*) - K(X_*, X)[K(X, X) + \sigma_n^2 I]^{-1}K(X, X_*)\big). \tag{5.17}$$

Note that the posterior variance is equal to the prior variance minus a positive term which is a function of the training inputs. Hence the posterior variance is always smaller than the prior variance - which is somewhat intuitive given that the training data has provided us with some information about the shape of the process.

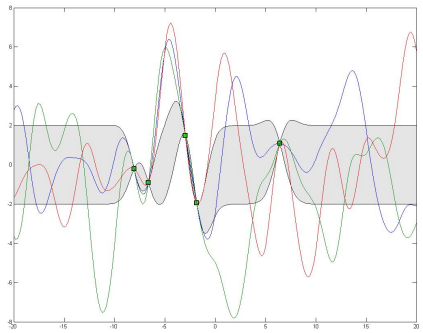
As Figure 5.6 shows, by conditioning on the training data what we have done is to restrict the set of candidate functions for our Gaussian Process posterior to only those functions that pass (with some tolerance) through the training data. The posterior process is used to make predictions for unseen test cases. Note however, as illustrated in Figure 5.6, the variance of the GP grows with distance from the training data points. Figure 5.7 illustrates using a Graphical Model how the marginalization property of a joint distribution allows us to start with an infinite dimension function space and still perform inference using a finite amount of computation in equation 5.17.



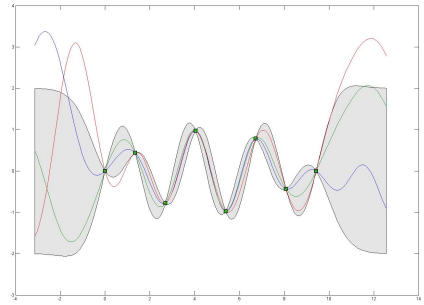
(a) Squared Exponential Prior



(b) Periodic Prior



(c) Squared Exponential Posterior



(d) Periodic Posterior

Figure 5.6: The Gaussian Process posterior.

5.4.2 Gaussian Process Classification

Gaussian Processes can also be applied to classification. A simple, intuitive approach for binary classification uses a *transfer function* $\pi(x)$ to *squash* the output of the regression and produce a probability that x belongs to one of two classes. For a two-class problem candidate transfer functions can be any sigmoidal function. These include the logistic function ($\text{sig}_{\text{logit}}$) and the probit function ($\text{sig}_{\text{probit}}$) stated below:

$$\text{sig}_{logit}(x) = \frac{1}{1+e^{-x}}, \tag{5.18}$$

$$\text{sig}_{probit}(x) = \int_{-\infty}^{x} \mathcal{N}(\tau|0,1)d\tau. \tag{5.19}$$

Obviously, the big difference between Gaussian Process classification and Gaussian

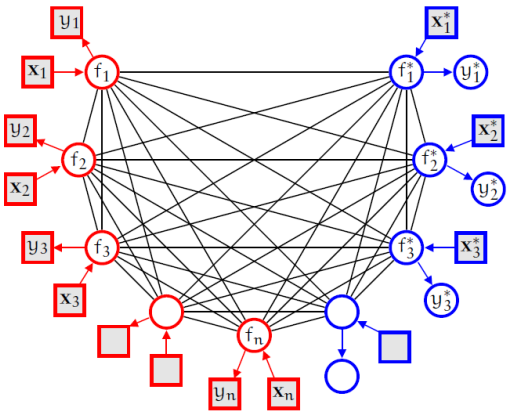


Figure 5.7: A Graphical Model of a Gaussian Process. Square nodes are observed, round nodes are the latent variables. Note how all pairs of latent variables are connected, and that each prediction y^* only depends on the corresponding f^*. Also note that the original distribution of the training data x_n, y_n, f_n is not altered by the addition of test points x_n^*, y_n^*, f_n^*. The marginalization property says that jointly Gaussian random vectors $\mathbf{x}$ and $\mathbf{y}$ with $\begin{bmatrix} x \\ y \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}, \begin{bmatrix} A & C \\ C^T & B \end{bmatrix}\right)$ lead to a marginal distribution of $\mathbf{x}$ of $\mathbf{x} \sim \mathcal{N}(\mu_x, A)$. The marginalization property explains why the addition of test points does not influence the training distribution, and allows inference using a finite amount of computation over our infinite dimensional Gaussian Process. Image courtesy (Rasmussen, 2006)

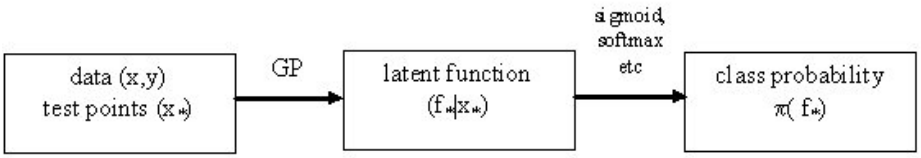


Figure 5.8: Gaussian Process Classification: The GP provides a latent function which is used as input to a transfer function which latches latent function values to a probability of being one of n classes.

Process regression is how the output data y relate to the underlying function inputs f. In the case of regression as we saw in equation 5.8 the two are related by a Gaussian noise process. In classification however, the output data is now discrete and represents *labels* which we have given our data, say $y = 0$ for *grass*, $y = 1$ for *bitumen*, $y = 2$ for *dirt path* and $y = 3$ for *scrub*. We can extend the *squashing* approach to multiple classes by positioning the GP as a *latent function* which determines the input to the transfer function $\pi(y)$. Figure 5.8 illustrates this approach.

We assume the class labels are independent random variables, which gives rise to

a factorial likelihood

$$p(\mathbf{y}|\mathbf{f}) = \prod_{i=1}^{n} p(y_i|f_i). \tag{5.20}$$

Binary classification using Gaussian Processes is performed in two steps. Using Bayes' rule we can compute the posterior of the latent function:

$$p(\mathbf{f}|X,\mathbf{y}) = \frac{p(\mathbf{f}|X)p(\mathbf{y}|\mathbf{f})}{\int p(\mathbf{f}|X)p(\mathbf{y}|\mathbf{f})d\mathbf{f}} \overset{i.i.d}{=} \frac{\mathcal{N}(\mathbf{0},\mathbf{K})}{p(\mathbf{y}|X)} \prod_{i=1}^{N} p(y_i|f_i). \tag{5.21}$$

Note that unlike in regression, the likelihood term $p(\mathbf{y}|\mathbf{f})$ is not Gaussian and is instead a probit or logistic function as described above.

The joint prior over the training and test latent values is

$$p(\mathbf{f}_*,\mathbf{f}|X,\mathbf{y},X_*) \sim \mathcal{N}\left(\begin{bmatrix}\mathbf{f}\\ f_*\end{bmatrix}\middle|\mathbf{0},\begin{bmatrix}K(X,X) & K(X,X_*)^T\\ K(X,X_*) & K(X_*,X_*)\end{bmatrix}\right) \tag{5.22}$$

so that the conditional prior can be written

$$\begin{aligned} p(\mathbf{f}_*|\mathbf{f},X,\mathbf{y},X_*) \sim \mathcal{N}\big(&K(X,X_*)K(X,X)^{-1}\mathbf{f},\\ &K(X_*,X_*) - K(X,X_*)K(X,X)^{-1}K(X,X_*)^T\big). \end{aligned} \tag{5.23}$$

From here we can obtain the class prediction of a new test point $\mathbf{x}_*$ in two steps:

$$p(f_*|X,\mathbf{y},\mathbf{x}_*) = \int p(f_*,\mathbf{f}|X,\mathbf{y},\mathbf{x}_*)d\mathbf{f} \tag{5.24}$$

$$= \int \underbrace{p(f_*|\mathbf{f},X,\mathbf{y},\mathbf{x}_*)}_{(5.23)}\underbrace{p(\mathbf{f}|X,\mathbf{y})}_{(5.21)}d\mathbf{f} \tag{5.25}$$

and then use this distribution to produce

$$p(y_*|X,\mathbf{y},\mathbf{x}_*) = \int p(\mathbf{y}_*|\mathbf{f}_*)\underbrace{p(\mathbf{f}_*|X,\mathbf{y},\mathbf{x}_*)}_{(5.25)}d\mathbf{f}_*. \tag{5.26}$$

The non-Gaussian likelihood of equations 5.21 and 5.25 makes equation 5.25 analytically intractable. Likewise, for certain transfer functions equation 5.26 is also intractable. We get around this problem by either using analytic approximations of these integrals (through techniques such as Laplace approximation or expectation maximization) or solutions based on Monte Carlo sampling.

Multi-class classification requires two major changes to the binary approach. In the multi-class case our n training points $\{x_1, ...x_n\}$ are now related to a vector y of length $C \times n$:

$$\{y_{i1}, y_{i2}, ...y_{iC}, ..., y_{n1}, y_{n2}, ..., y_{nC}\}$$

with entries of 1 for the class which corresponds to the label of point $\mathbf{x_i}$ and 0 for the remaining $C - 1$ entries for that point.

Again, we assume the existence of an underlying latent function $f(x)$

$$\{f_{i1}, f_{i2}, ...f_{iC}, ..., f_{n1}, f_{n2}, ..., f_{nC}\}$$

over which we place a GP prior with form $p(\mathbf{f}|X) = \mathcal{N}(\mathbf{0}, K)$. K is a block diagonal matrix as the C latent processes are uncorrelated: within each sub matrix K_C we represent the correlations within that class C only. So our first major change over the binary case is to make K much bigger, now size $C_n \times C_n$.

The second major change is required because our single dimensional transfer functions such as probit and logit are insufficient to handle the 'multi-level squashing' required in multi-class classification. Instead we typically make use of the *softmax* function:

$$p(y_i^c|\mathbf{f}_i) = \frac{\exp(f_i^c)}{\sum_{c'} \exp(f_i^{c'})}. \tag{5.27}$$

Equations (5.25) and (5.26) still hold for the multi-class case. As with binary classification, a number of approximation techniques exist for performing multi-class Gaussian process classification. They vary in their choice of likelihood function (e.g.

softmax, multinomial probit) and in their approach to handling the analytically intractable integrals of equations (5.25) and (5.26).

5.4.3 Training the GP

In the previous sections we have described how Gaussian Processes can be used to do both regression and classification. However in our description so far we have assumed that sensible priors for the mean and covariance of the Gaussian Process are available. Typically though, we have only vague information on the shape of our prior. While not completely free-form, Gaussian Processes provide what is commonly called a non-parametric approach to regression and classification. We need specify only the family of covariance function and we leave it up to the training data to 'speak' as to the form of the *hyperparameters* of that particular covariance function family.

As an example, the oft-used squared-exponential covariance of equation 5.7 has 3 hyperparameters $\boldsymbol{\theta} = \{\sigma_n, \sigma_f, l\}$.

We can use the training data to learn what these parameters should be by maximizing the log marginal likelihood of the training data given the inputs and the hyperparameters $p(\mathbf{y}|X, \boldsymbol{\theta})$. The marginal likelihood is

$$p(\mathbf{y}|X, \boldsymbol{\theta}) = \int p(\mathbf{y}|\mathbf{f}, X, \boldsymbol{\theta}) p(\mathbf{f}|X, \boldsymbol{\theta}) d\mathbf{f} \tag{5.28}$$

whereby we marginalize over the function values $\mathbf{f}$. The prior $p(\mathbf{f}|X, \boldsymbol{\theta})$ is a Gaussian distribution $p(\mathbf{f}|X, \boldsymbol{\theta}) \sim \mathcal{N}(\mathbf{0}, K)$ and the likelihood is a factorized Gaussian $p(\mathbf{y}|\mathbf{f}, X, \boldsymbol{\theta}) \sim \mathcal{N}(\mathbf{f}, \sigma_n^2 I)$. As the product of two Gaussians is a Gaussian, once we perform the integration we are left with

$$\log p(\mathbf{y}|X, \boldsymbol{\theta}) = -\frac{1}{2}\mathbf{y}^T (K + \sigma_n^2 I)^{-1} \mathbf{y} - \frac{1}{2} \log |K + \sigma_n^2 I| - \frac{n}{2} \log 2\pi. \tag{5.29}$$

Maximizing the log marginal likelihood using any multivariate optimization algo-

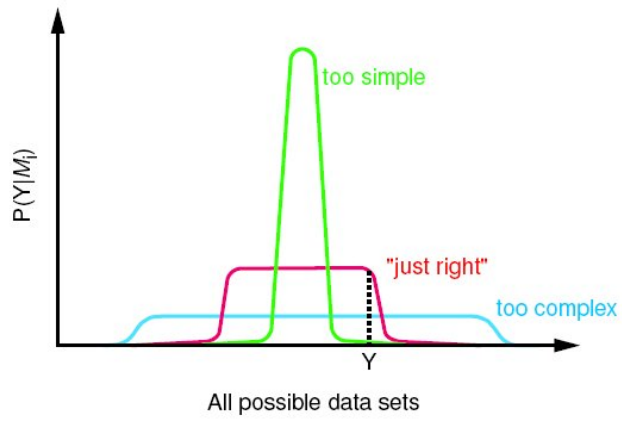


Figure 5.9: Image from Rasmussen (2004). An illustration of how, for a particular dataset denoted by y and indicated by the dotted line, the marginal likelihood will prefer a model of intermediate complexity over simple and complex alternatives.

rithm (such as the conjugate gradient method) leads to a suitable choice for $\boldsymbol{\theta}$.

Note that the log marginal likelihood expression of equation 5.29 contains three terms. The first is a negative quadratic and is the only term that relies on the output values $\mathbf{y}$. It acts to ensure that the predictions of the log marginal likelihood method closely fit the data. The second term $-\frac{1}{2}\log|K + \sigma_n^2 I|$ measures (and penalizes) the complexity of the model. Between them, these two terms implement an automatic trade-off between data fit and complexity in the GP model which is illustrated in Figure 5.9.

Of course, maximizing the log marginal likelihood in this way is subject to the pitfalls of numerical optimization and care must be taken against bad local minima. It is possible (although often intractable) to maximize over the full Bayesian posterior $p(\boldsymbol{\theta}|\mathbf{y}, X)$ and a full treatment of this is provided in Chapter 5 of Rasmussen and Williams (2006).

5.4.4 Computational Complexity and Sparse Methods

The problem with Gaussian Processes is that in order to predict a new value we are required to invert a matrix, see equation (5.17) and (5.23). This operation is order

$O(N^3)$ in the number of training points and becomes unwieldy for more than 1000 or so data points. This high cost makes standard GPs unsuitable for large data sets, and as such a number of *sparse* approaches to Gaussian process modeling have arisen to combat this challenge.

Typically, these sparse methods seek to reduce the number of training points by selecting $M \ll N$ points which are still *informative* enough (as measured by entropy) to adequately represent the training set. This reduces training cost to $O(NM^2)$ and prediction to $O(M^2)$.

In the next section we make use of the Informative Vector Machine (IVM) technique introduced by Lawrence (Lawrence et al., 2004) for Gaussian Process regression. IVM operates by selecting a key subset $d < N$ of the N original data points to provide a sparse representation of the data set, in much the same way the support vectors provide the bulk of the information about the location of the discriminator in the SVM technique. The selection of these points is motivated by the entropy (information content) of the posterior process.

Two sets of points are maintained, I holds the 'active points', currently selected, and J holds those not currently included in the critical set. The data points in I are greedily selected according to the entropy change

$$\Delta H_{in} = \frac{1}{2} \log(1 - \upsilon_{in} \varsigma_{i-1,n}) \tag{5.30}$$

where $\varsigma_{i-1,n}$ is the nth diagonal element from Σ_{i-1}, the approximation of the posterior covariance to date.

The user decides on the size of the set d, so a rolling set of informative points is maintained.

We use the Variational Bayes MATLAB toolkit for classification. As the name suggests, it uses a variational Bayes approximation to the non-Gaussian likelihood term - as opposed to the Markov Chain Monte Carlo (MCMC) or Laplace approxi-

mations found elsewhere. Rather than employ separate one-vs-all discriminators, it represents the joint likelihood over K classes, and uses IVM to approximate that likelihood in a sparse fashion. The 'squashing function' is the multinomial probit model, a generalization of the sigmoidal probit function to more than one class.

5.5 Results

In this section we present results of the probabilistic cost map framework. To illustrate the capabilities of the framework we have used a mixture of simulated data and overhead images drawn from Google Maps. Ideally, we would like to have run the framework on a real robot and shown the robot successfully following the paths it generated. There are a number of reasons why this was not done.

Firstly, the main contribution of this work lies in the ability to address uncertainty in terrain knowledge in planning - and it is not necessary to translate it onto an operational platform to prove its utility in accomplishing this. In moving to a real platform, the classification algorithm would simply accept data from a different sensor (such as a camera or a laser scanner) rather than a manufactured or overhead image. The terrain maps, which are reliant (at least initially) on local knowledge and a human estimate of the cost to the robot of driving over that sort of terrain, would remain the same.

Furthermore, the framework is most useful in the case of long range planning outdoors - because it is here that costs within a single terrain type are more likely to vary - and the framework explicitly accounts for this spatial variance in cost. We did not have access to the sort of autonomous vehicle capable of doing these long range cross-country traverses.

Finally, a brief survey of the path planning literature will reveal that using simulation to prove the validity of an idea is somewhat ubiquitous. Constructing simulated,

yet realistic, datasets which highlight the uses of the framework as we have done here is in-line with the existing literature and much more cost-effective than waiting for the 'perfect example' to present itself when operating in the real world.

That said, there is definitely scope to adapt this to onboard planning and this will be discussed at the conclusion of the chapter. In the meantime we discuss the results obtained from simulation here.

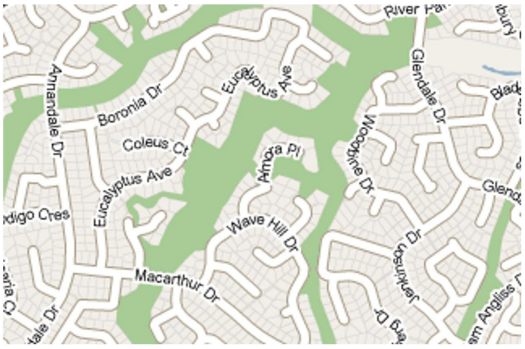
5.5.1 Responding to Variable Costs Within One Terrain Type

5.5.1.1 Synthetic Dataset I

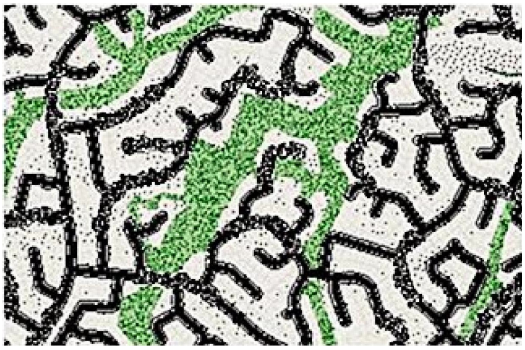
The following example shows our framework operating on a simulated data set. The image has a resolution of 15m per pixel, and each pixel becomes a grid cell for our planner. To make the dataset 'realistic' the image was constructed using a terrain map from Google Maps (Figure 5.10(a)), the text and property boundaries were removed and some noise was added by hand to provide some variation in the colour of each class. The resulting image (Figure 5.10(b)) was labelled with 3 classes, and the classifier was then trained using 50 points each for the *road*, *obstacle* and *scrub land* classes with input features corresponding to red, green and blue channels in the image.

In Figures 5.10(c) - 5.10(e) we see the results of the classification stage. Note how we get reasonably strong probabilities (above 0.6) in the *obstacle* and *road* classes, yet a muted response for the *scrub* class. Looking at the original image we see the greatest variation in colour amongst the *scrub* class, where there are various shades of green (N.B. These results are best understood in colour) and the light and dark extent of the colour range represented leaves pixels of this class more open to mis-classification than *road* or *obstacle* pixels.

Figures 5.10(f) - 5.10(h) show the spatial cost maps for each terrain type masked by the terrain classification. The cost maps for the *obstacle* and *scrub* classes are



(a) The basis for the original image, taken from Google Maps



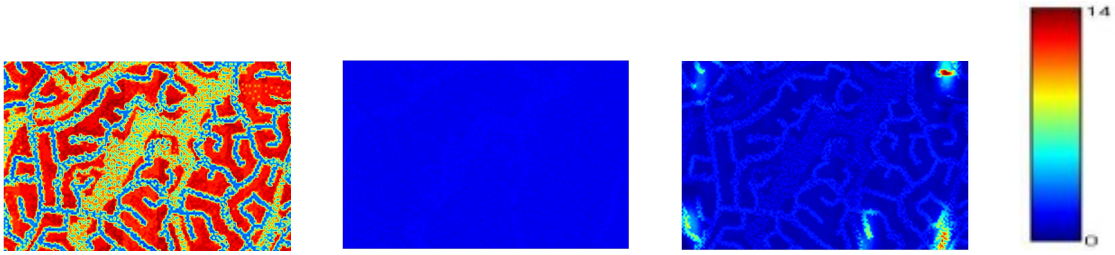
(b) The original image: A simplified map of a residential area with black corresponding to roads, white to rooftops and green to scrub.

Construction of the Urban Dataset.



(c) *Obstacles* (d) *Scrub* (e) *Road*

The results of classifying the image into 3 classes: *obstacle*, *scrub* and *road*. Note the relatively good discrimination between high and low confidence regions obtained for both the *obstacle* and *road* classes, and compare this with the *scrub* class, where confidence in actual regions of *scrub* (evident here in (b)) barely reaches 0.4. The *scrub* class has the highest variation in colour (see the mottled green in 5.10(b)) and as we train and classify on colour information only, this leads to poorest results for this class.

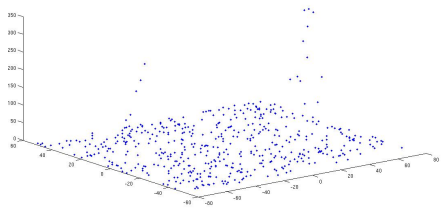


(f) *Obstacle* Costmap: instance 1 (g) *Scrub* Costmap: instance 1 (h) *Road* Costmap: instance 1

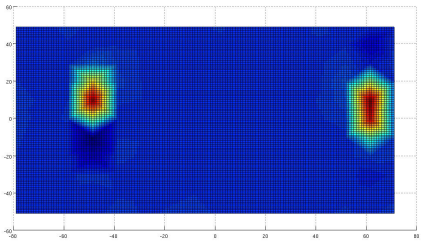
The cost maps for each of the 3 terrain types used for the first iteration of the planner. To enhance understanding we have masked the cost maps with the probability of class membership. Note that the *obstacle* class is generally high cost everywhere, while the *road* class is almost uniformly low cost except for localized high cost regions which we have deliberately added around the edge of the image.

Figure 5.10: continued over page

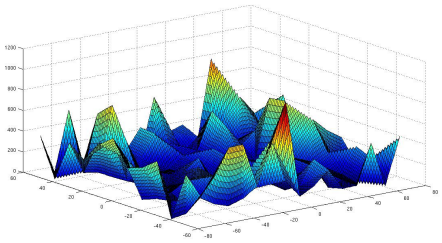
almost uniform over the map, but we have added some local high cost areas to the *road* class to simulate areas of congestion, visible in Figure 5.10(h). The blue lines in



(i) *Road* costmap instance 2: Training Data

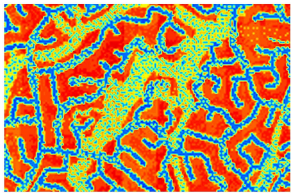


(j) *Road* costmap instance 2: Predicted Mean

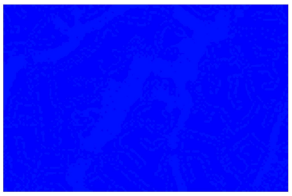


(k) *Road* costmap instance 2: Predicted Variance

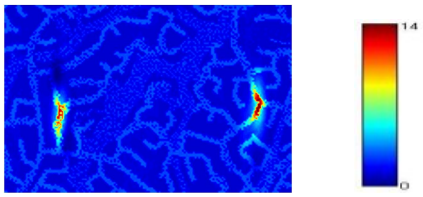
The training data and resulting posterior mean and variance for the *road* costmap instance 2; resulting from the evaluation of the spatial GP.



(l) *Obstacle* costmap: instance 2



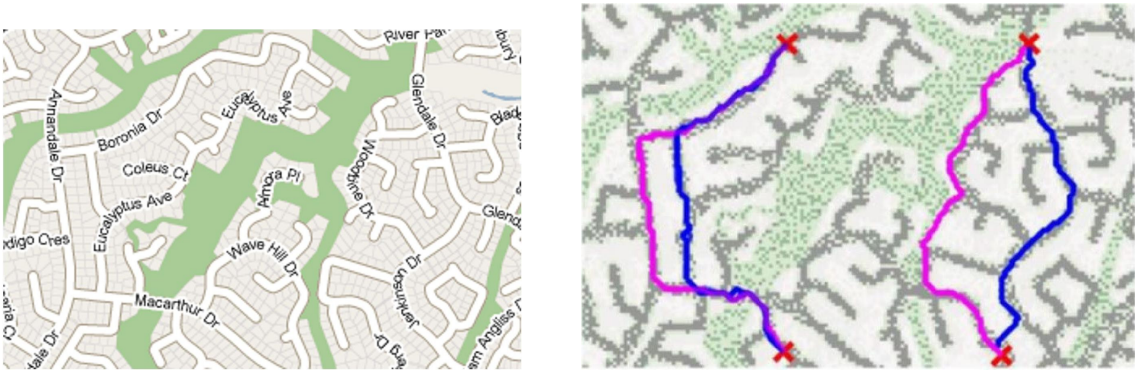
(m) *Scrub* costmap: instance 2



(n) *Road* costmap: instance 2

The cost maps for each of the 3 terrain types used for the second iteration of the planner. Note the high cost regions added to the *road* costmap right where the path of instance 1 passed through.

Figure 5.10: continued ...



(o) Original Image

(p) Resultant Paths

Resultant Paths, two different searches using two different instances of the costmap. The blue line represents the original path planned using the costmap of Figures 5.10(f) - 5.10(h). We then simulated congestion on the road by adding a high cost region to the *road* costmap in Figure 5.10(n). Note that the magenta path resulting from instance 2 now veers around the high cost regions added to the map, and that by comparing with the original image (a) the magenta path in the right hand side of the image actually prefers to traverse *scrub* rather than *road* for part of the route. This shows the framework allows us to modify locally the cost of one terrain type, and that the planner responds by avoiding the high cost area.

Figure 5.10: Synthetic Data Set I. This figure shows that planner successfully responds to the local variation in terrain cost which we added to the *road* class. Initially, the *road* terrain cost map contained a few localized areas of high cost around the border of the map. The blue path shown in (m) resulted from A* searches between the two sets of start and goal nodes denoted by crosses on the map. We then added a high cost areas to the road terrain cost map, see (k), to simulate congestion on the initial blue paths. In both search cases the planner successfully plans around the congested areas, shown by the magenta path.

Figure 5.10(p) show two different paths planned using this particular costmap.

Figures 5.10(i) - 5.10(k) show how the training data is manually specified for the second instance of the *road* costmap in this dataset. The spatial GP is evaluated from the training data of Figure 5.10(i) and this leads to the posterior mean and variance of the *road* costmap in Figures 5.10(j) and 5.10(k) respectively.

Figures 5.10(l) - 5.10(n) show the convolution of spatial cost map and class probability for a different congestion situation. The difference is most visible in Figure 5.10(n) where we see two very high cost regions in the *road* costmap covering territory which the blue path of the previous instance passed through. Running the planner over this second costmap configuration produces the magenta paths shown in Figure 5.10(p), so we see that by introducing some local spatial variation in the *road* class only we have been able to influence the path chosen by the planner.

In summary, the results in Figure 5.10 are what we expect. The classification stage provides good, accurate discrimination between the 3 class types and this means that the alterations we make to the individual terrain cost maps are directly translated to the overall costmap. These results show the framework works to specification, and is capable of reflecting local variation into the cost of a single terrain type accurately in the overall cost map and hence allows the planner to take localized anomalies in cost (in this case road congestion) into account.

5.5.1.2 The Tempe Dataset

The Tempe Dataset satellite image was obtained from Google Maps. The aerial image is centered at Latitude 33.413847 North, Longitude -111.918526 West, a suburban area of Tempe, Arizona. The image size is 320 pixels by 240 pixels, taken at zoom level 16 (Google Maps divides the world up into tiles - at zoom level 0 there is one tile representing the earth and each succeeding zoom level doubles the precision in both the horizontal and vertical directions out to a maximum zoom level of 21). The original image is shown in Figure 5.11(a).

The results that follow show again how the framework can be used to plan paths that respond to variations in cost within a terrain type. Using the same principles as the previous case, we add show the effect of adding an area of congestion to the *road* costmap. However this time we illustrate this capability using an aerial map.

The results of the classification stage are shown in Figures 5.11(b) - 5.11(d).

The first costmap for the *road* terrain type had form shown in Figures 5.11(e) - 5.11(g).

The corresponding costmaps for each terrain class, masked by probability of class membership, are shown in Figures 5.11(h) - 5.11(j). The resultant costmap to be fed to the planner from this instance is shown in Figure 5.11(k).

The second costmap used for the *road* terrain type has the form of Figure 5.11(l) - 5.11(n).

Its corresponding costmaps by class type are illustrated in Figures 5.11(o) - 5.11(q). Node the added high cost regions (light colours correspond to higher costs) in the total costmap of Figure 5.11(r).

The result of running the planner over the two different costmaps is shown for two different queries in Figures 5.11(s) and 5.11(t). Note how the cyan coloured path now veers around the high cost regions added in the second costmap instance.

Note that while both Figures 5.11(s) and 5.11(t) illustrate that the ability to

account for local variation in terrain cost can be carried over to an aerial image, they both highlight a point of fragility in our framework. We have annotated both images to point out sections of the path which veer off through backyards rather than sticking to the road. While this type of path may be feasible for some robots, it was not what we were aiming for here and we deliberately weighted the relative terrain costs of *scrub* and *road* to preference *road* overwhelmingly.

The reason this error occurs stems from the classifier. Because we are classifying on colour alone, we are unable to differentiate (highly non-traversable) backyards from (possibly traversable) open areas of parkland, meaning that both are classified as *scrub*. The cost we have assigned to *scrub* means that the planner will search approximately 10 times as far along the *road* as it will through *scrub* to find a path. However, in the highlighted section of the map the shortcut through the backyard still proves a cheaper option under this cost scheme.

A simple remedy would be to scale up the cost associated with the *scrub* class. However this means we do away with the perfectly viable option of sometimes traversing open parkland. A more sophisticated solution would be to take into account the neighbourhood of the cell in classifying, and use the proximity to obstacles to discriminate between backyards and open parkland. We elected not to pursue this avenue as the focus in this work is on creating and utilizing probabilistic costmaps, rather than creating robust classifiers.

A further remedy, which would be possible once we implemented this system online on a real robot, would be to use the framework to update locally the cost of terrain. So in the cases illustrated in Figure 5.11(s) and 5.11(t), the path would still direct the robot towards the back yard shortcut, but on arrival it would realize that the *scrub* terrain in this area is locally very high cost and could update the *scrub* terrain cost map to reflect this. Once this was done, the effect of the spatial regressor would be to automatically propagate this high cost locally and the robot could replan a new

path around the road.

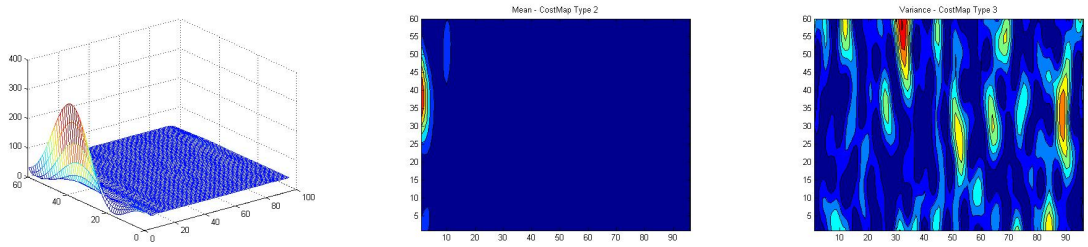


(a) The Original Image, taken from Google Maps



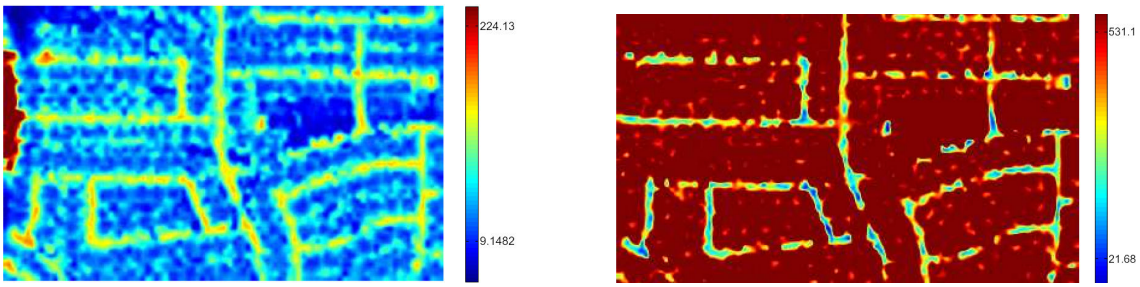
(b) *Road* (c) *Scrub* (d) *Obstacle*

The Tempe Dataset: Results of Classification Stage

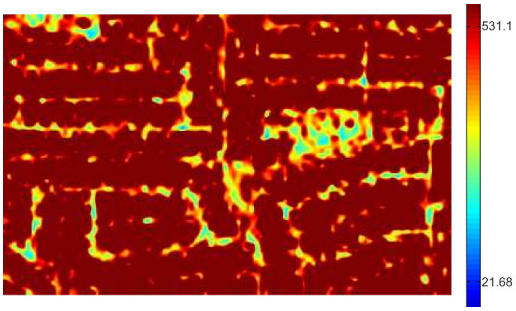


(e) 3D Plot of the Mean (f) Contour Plot of the Mean (g) Contour Plot of the Variance

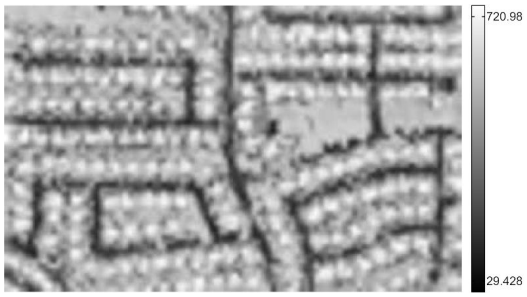
The costmap for the *road* terrain type - instance 1 of planning with spatial variation in terrain Cost, Tempe Dataset



(h) *Road* costmap (i) *Scrub* costmap

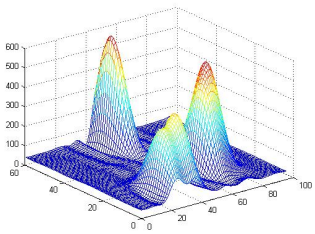


(j) *Obstacle* costmap

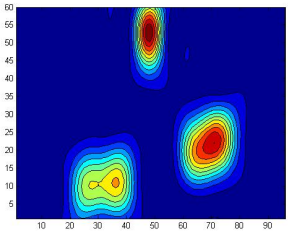


(k) Total costmap

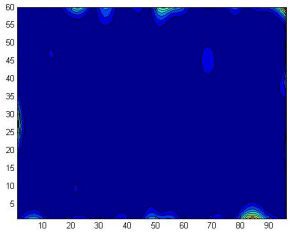
The Tempe Dataset, individual terrain type costmaps masked by probability of class membership, instance 1



(l) 3D plot of the mean

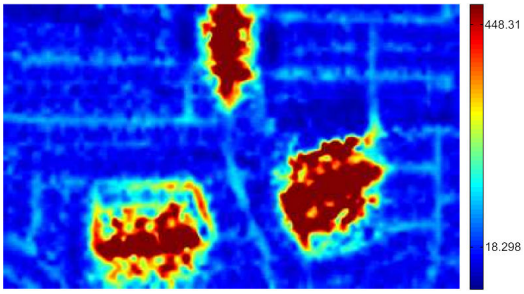


(m) Contour plot of the mean

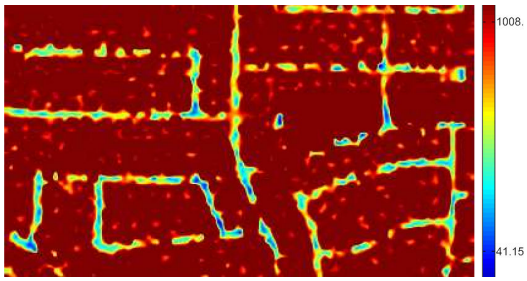


(n) Contour plot of the variance

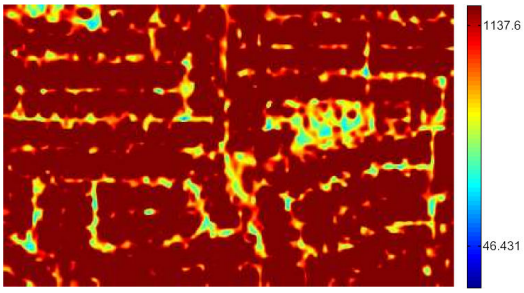
The costmap for the *road* terrain type - instance 2 of planning with spatial variation in terrain cost, Tempe Dataset



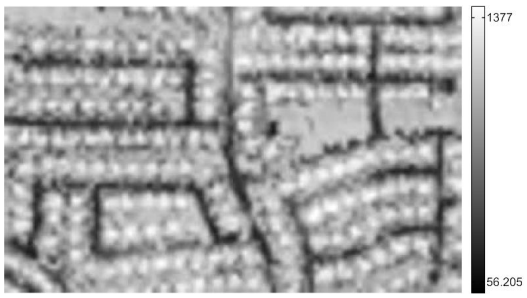
(o) *Road* costmap



(p) *Scrub* costmap



(q) *Obstacle* costmap

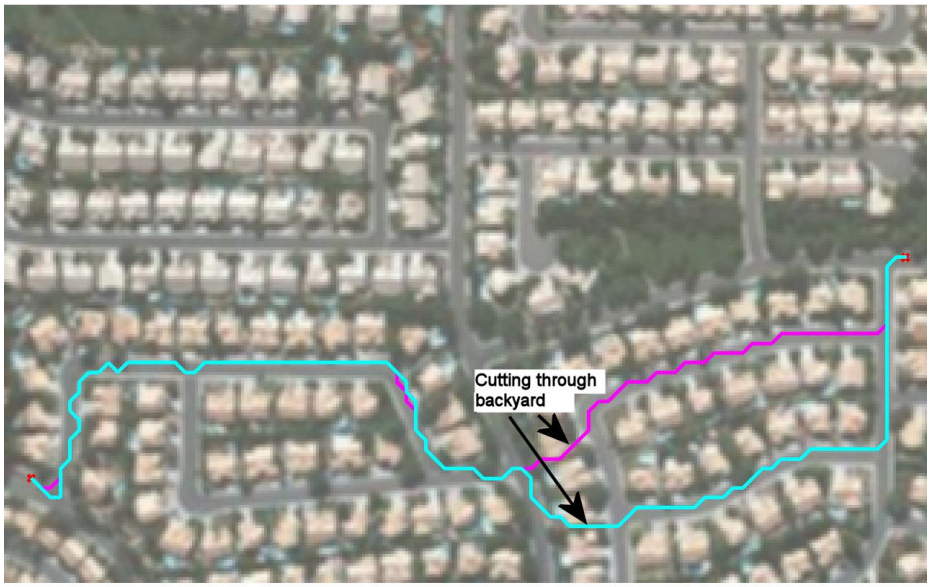


(r) Total costmap

The Tempe Dataset, individual terrain type costmaps masked by probability of class membership, instance 2



(s) Planning with spatial variation in terrain cost for the *road* terrain class. The magenta path is the result of planning over the costmap of Figure 5.11(k). The cyan path is planned around the high cost regions added in Figures 5.11(o) and 5.11(r).



(t) Planning with spatial variation in terrain cost for the *road* terrain class. A further example using the Tempe Dataset. The magenta path is the result of planning over the costmap of Figure 5.11(k). The cyan path is planned around the high cost regions added in Figures 5.11(o) and 5.11(r).

Figure 5.11: Tempe Data Set. The figures illustrate that responding to local variations in terrain cost successfully carries over to an implementation on an aerial image. However, the restrictive nature of our classifier and the lack of any localized terrain cost data for the *scrub* class means that we fail to discriminate adequately between non-traversable backyards and open parkland here. With the relative weighting of the *road* and *scrub* classes meaning that the planner searches only approximately 10 times as far along the *road* as it does across *scrub*, the shortcut through the backyard is preferred over the much longer *road* route in both (s) and (t).

5.5.2 Generating a Distribution of Paths

The second application of the probabilistic costmap framework is to sample the pdf cost function in order to generate a distribution of paths between points. We demonstrate this operating over 4 separate datasets - a further synthetic dataset (Figure 5.12), an urban area (Figure 5.13), an inner city area with parkland (Figure 5.14), and a desert (Figure 5.15).

The method used to construct the probabilistic costmaps is identical to that in the previous section. Again, we limited ourselves to 3 terrain classes and trained the classifier using approximately 50 training points per class. In this section we are demonstrating the utility of sampling from the costmap, so only one instance of the costmap is used for each example. For simplicity, the variance of each grid cell for each of the 3 classes in all datasets is set to a random fraction of the cell's mean. The cost maps were approximately uniform and scaled appropriately so that *road* was preferred over *scrub/parkland* which was in turn preferable to *obstacle*.

In Figures 5.12 to 5.15 we see the results of running 200 different A* searches over the map, resampling the cost function each time. All paths have been plotted using an alpha-blended overlay. In each case we see a thickened *most-likely* path, together with other *feasible* paths that could result given that many of the grid cells on the most-likely path have uncertain classifications.

5.5.2.1 Synthetic Data Set II

Like the first synthetic dataset, this dataset was also constructed using a terrain map from Google Maps (Figure 5.12(a)). The resulting synthesized image has been designed to represent 3 class types, *road*, *parkland* and *obstacles*. This particular area was chosen for its rectangular nature which means there should be multiple equal length paths between points chosen on opposite sides of rectangles found in the map - ideal for illustrating a multi-modal distribution of paths between two points.

In Figures 5.12(c) - 5.12(e) the results of the classification stage are shown. Figures 5.12(f) - 5.12(h) show the costmaps by class for each of the 3 classes, while Figures 5.12(i) and 5.12(j) show the resulting costmap mean and variance respectively.

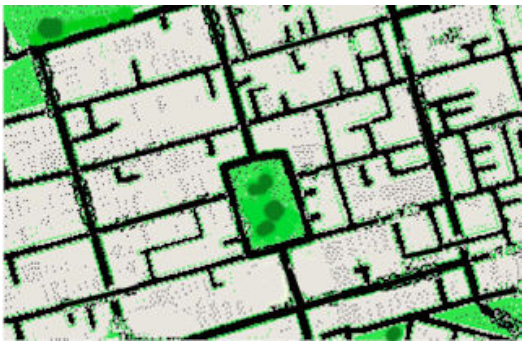
Figures 5.12(k) - 5.12(n) show that when we sample from this costmap and plan paths between the red and blue crosses marked on the images, we indeed get more than one configuration of the shortest path. In fact, there are 4 different shortest paths evident from the 200 samples.

Figure 5.12(o) provides an α-blended overlay of all 4 path modalities. The darker regions will be traversed with greater probability. Figure 5.12(p) provides a histogram showing the frequency of occurrence of each of the 4 path modes. While the magenta path is the most probable, in a significant portion of instances sampled from this costmap the yellow path would actually be shorter.

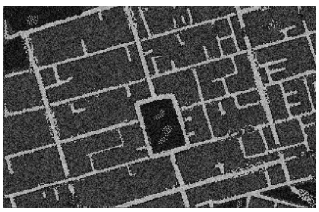
The results we see here conform to expectation. Our synthetic environment is a city grid with a square of parkland in the middle. With the naked eye it is easy to pick out that there are a number of similar-length paths which navigate between the blue and red squares of Figures 5.12(k) to 5.12(o). If we were planning deterministically over this image we would return only the magenta path as the shortest path. We refer to this path as the 'most-likely' path, and it is the same path that would be returned if we planned over the costmap using only the mean value of each grid cell. However when we sample from the probabilistic cost map, each grid cell sample results in some perturbation (the magnitude is governed by the variance of the grid cell) on the mean value. With a number of similar length paths to choose from here, it is natural to expect that every now and then the combined result of these perturbations over the whole map means that the 'most-likely' shortest path is not chosen.



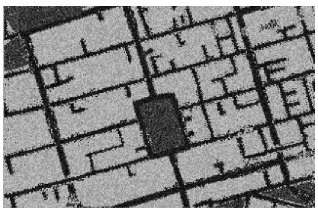
(a) The basis image, taken from Google Maps



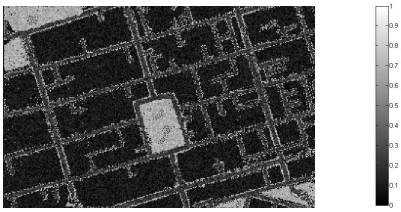
(b) The synthesized image



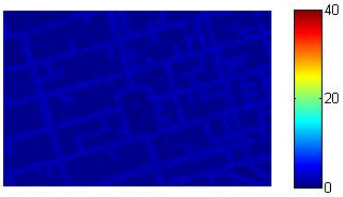
(c) *Road* class



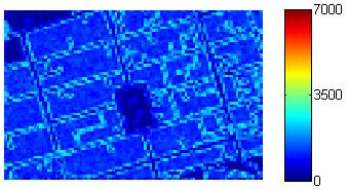
(d) *Obstacle* class



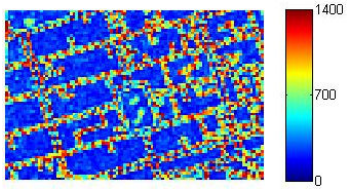
(e) *Parkland* class



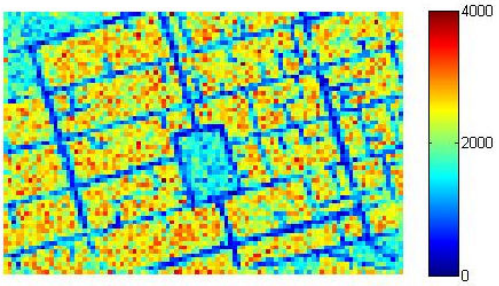
(f) *Road* costmap - Mean



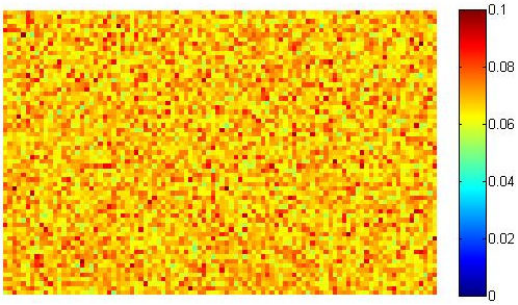
(g) *Obstacle* costmap - Mean



(h) *Parkland* costmap - Mean



(i) Total costmap - Mean



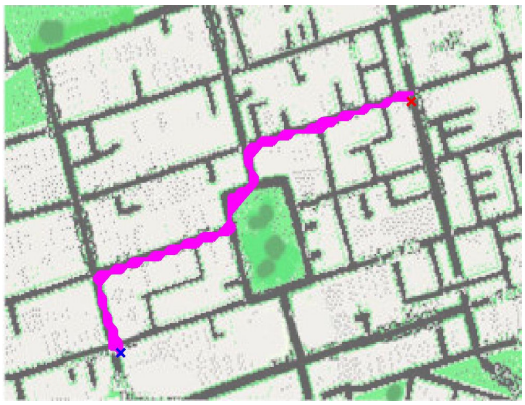
(j) Total costmap - Variance (as a fraction of cell mean)



(k) Cyan Paths - Across the bottom



(l) Red Paths - Around the square



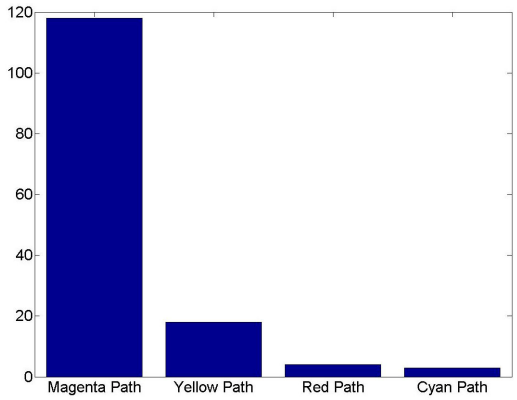
(m) Magenta Paths - Top Diagonal



(n) Yellow Paths - Bottom Diagonal



(o) The resultant distribution of paths- an α-blending of all 4 path types



(p) Bar graph showing the distribution of paths

Figure 5.12: Synthetic Dataset II. Sampling from a probabilistic costmap produces a distribution of possible paths between any two points in the map. Here the magenta path is the 'most-likely' path and is the one that would result should we pass only the mean value of grid cells to the planner. In sampling from the costmap, at each cell we add some perturbation to the mean. The cumulative effect of those perturbations over the map means that on occasion, paths other than the most-likely path are actually shortest.

5.5.2.2 Tempe Data Set

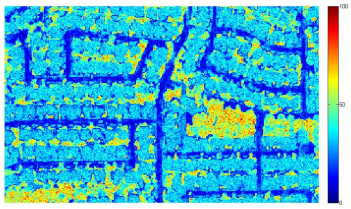
Here we generate samples from a costmap generated using the Tempe Dataset introduced in Section 5.5.1.2. As the results of classification remain the same we choose not to duplicate Figures 5.11(b) - 5.11(d) here. Each class is combined with a uniform costmap (where the mean is scaled according to the relative cost of *road*, *scrub* and *obstacle's* terrain, and the variance is a random fraction of the mean of each cell capped at 10% of the mean - done to reflect the sort of uncertainty we expect to see in real world data). The resultant costmap mean is shown in Figure 5.13(b) and the variance is depicted in 5.13(c).

Figures 5.13(d) - 5.13(h) show that 5 main path modalities result from sampling from the resultant costmap and planning paths between the red and blue crosses marked on the image. Figure 5.13(i) provides an alpha-blended image of the 5 paths, illustrating that the bottom path - corresponding to the magenta path of Figure 5.13(d) - is the most likely path. This is confirmed by the histogram of Figure 5.13(j) which shows the magenta path is nearly 3 times as likely to be the shortest path than any of the other 4 path modalities.

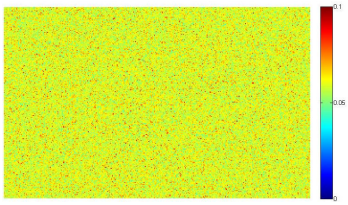
Notice that none of these path modalities adheres completely to the *road*. As discussed in Section 5.5.1.2, if desired this could be resolved by a more powerful classifier and/or a scaling up of the relative cost of the *scrub* terrain class.



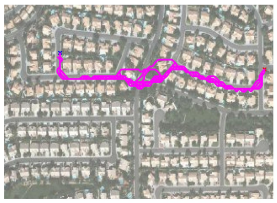
(a) Original Image



(b) Total costmap - Mean



(c) Total costmap - Variance



(d) Magenta Path - the bottom path



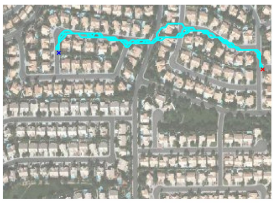
(e) Yellow Path - the cross path



(f) Blue Path - bump shaped path



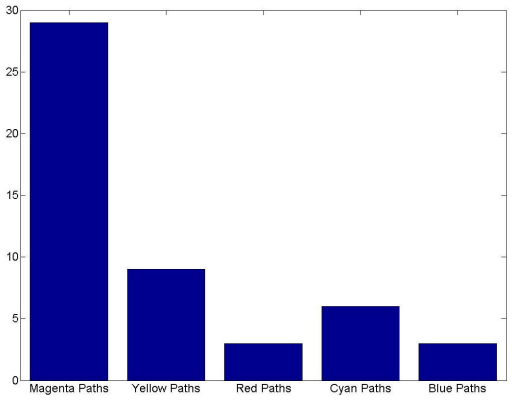
(g) Red Path - reverse cross direction



(h) Cyan Path - the top path



(i) Overall Path Distribution - an alpha blending of all 5 path types



(j) The distribution of paths

Figure 5.13: The Tempe Dataset. This shows that we can generate a distribution of paths over aerial images. Note again that as in Figure 5.11 the quality of our results here are limited by the our inability to distinguish between open parkland and backyards and none of the 5 path modalities adheres strictly to the roadways.

5.5.2.3 Lisbon Data Set

The Lisbon Dataset was also obtained from Google Maps (Figure 5.14(a)). It is a 320×240 pixel image taken at zoom level 16, centered at latitude 38.773009 North, longitude -9.118418 West. It surveys part of suburban Lisbon notable for its curved roadways, greenery and red roofs.

Figures 5.14(b) - 5.14(d) show the results of the classification stage. By combining these results with scaled, uniform costmaps with random variances we obtain the resultant costmap with mean depicted in Figure 5.14(f) and variance in Figure 5.14(g).

Figures 5.14(h) - 5.14(l) depict the 5 principle path modalities that result from sampling the resultant costmap and planning paths between the red and blue crosses marked on the image. Figure 5.14(m) provides an alpha blended overlay which shows that the magenta paths of Figure 5.14(j) represent the most-likely path. However, as Figure 5.14(n) shows, the other 4 path modalities occur as the shortest path with significant probability.

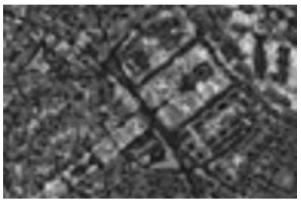
Note that in the Lisbon Dataset, all 5 path modalities adhere to the *road* terrain type. This shows that even with crude classification and minimal terrain classes good discrimination between drivable and non-drivable terrain can be achieved. We achieve better results here than in with the Tempe dataset in Figure 5.11 primarily because this image is heavily populated with road ways and the planner never has to search very far to find a better path along a road than through parkland. The good performance of the framework on this image also owes something to the ease of separability of our three classes (*road, scrub* and *obstacle*) on colour alone. The red roofs in this image mean we have limited confusion between roads and buildings, or trees and buildings; more so than in the Tempe Dataset where dark roads and overhanging trees lead to substantial confusion between these two classes.



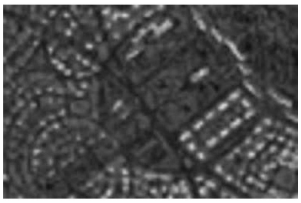
(a) The original image, courtesy of Google Maps



(b) *Road* class

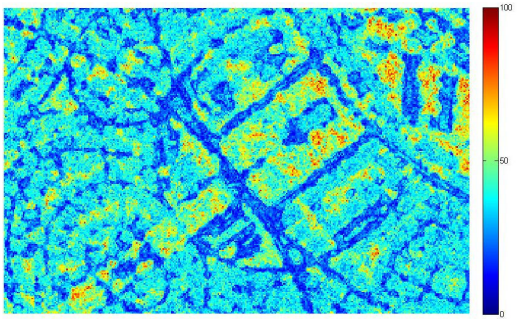


(c) *Scrub* class

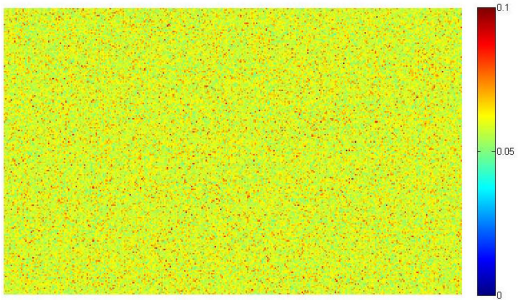


(d) *Obstacle* class





(f) Total costmap - Mean



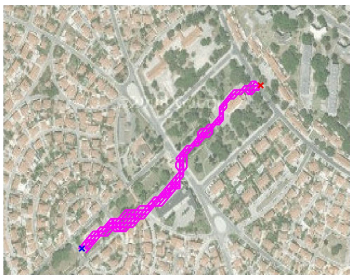
(g) Total costmap - Variance as a percentage of the cell mean



(h) Blue Path - the bottom path



(i) Cyan Path - the top road



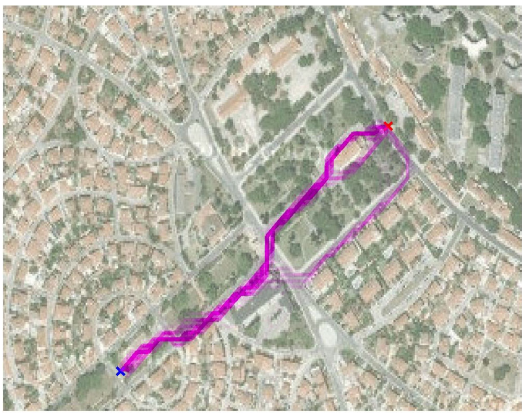
(j) Magenta Path - the road most travelled



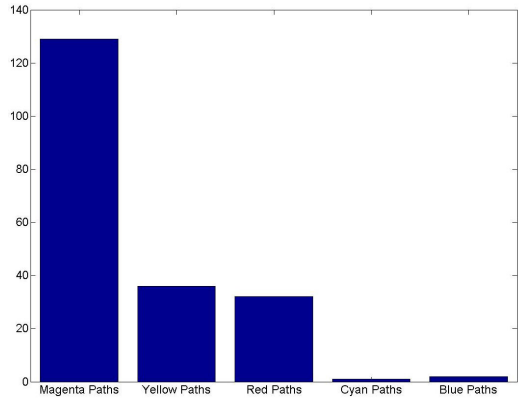
(k) Red Path



(l) Yellow Path



(m) Overall Path Distribution - an alpha blend of the 5 path types



(n) A histogram showing the distribution of paths on the Lisbon Dataset

Figure 5.14: The Lisbon Dataset. Five path modalities are detected in planning between the tow points indicated. High colour separability of the three classes combined with a surplus of roads in the area means the framework performs exceptionally well on this dataset. All 5 path modalities adhere completely to road ways.

5.5.2.4 Whyalla Data Set

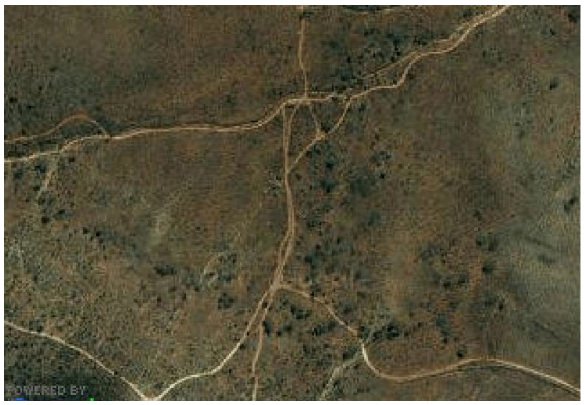
The Whyalla Dataset (Figure 5.15(a)) is centered on latitude -32.998171 South and longditude 137.543098 East. It is a 320×240 image taken at zoom level 16. The terrain encompasses 3 classes: red, sandy *desert* and *scrub*, and it is criss-crossed by dirt *roads*.

Figures 5.15(b) - 5.15(d) show the results of the classification stage. The resultant costmap mean (Figure 5.15(e)) and variance (Figure 5.15(f)) are again the product of multiplying each cell's class membership probability with the cost of that terrain class at the cell and summing over all 3 terrain classes. Again, suitably scaled uniform costmaps were used for each terrain type.

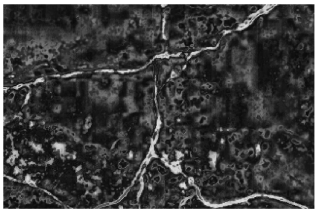
Figures 5.15(g) - 5.15(i) illustrate 7 different path modalities that occur when sampling from the costmap and planning over the resulting costmap between the two crosses denoted on the figures. In Figure 5.15(g) the paths take the fork to the left at the first junction when travelling from the blue cross towards the red. In Figure 5.15(h), the cyan paths take the middle fork at this junction, while a solitary green path takes the right fork. The paths of Figure 5.15(i) show a number of cross-country paths through the scrub.

Figure 5.15(j) shows the alpha-blended overlay of the paths from which we have distilled the 7 path modalities. Figure 5.15(k) is a histogram showing the frequency with which each of the modalities occur.

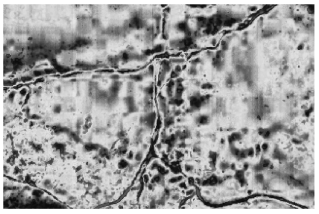
While the most-likely path in this case is the cyan path of Figure 5.15(h), the cross-country paths of Figure 5.15(i) are also highly likely to be the result of a shortest path search on our probabilistic costmap. Note in Figures 5.15(b) - 5.15(d) we see less obvious distinction between class types as a result of the classification stage than is evident in the Tempe or Lisbon datasets. In Figure 5.15(b) particularly, notice that at the edges of the image the classifier does quite well at recognizing the road, but performs poorly on the area we plan over in the centre of the image. The overall



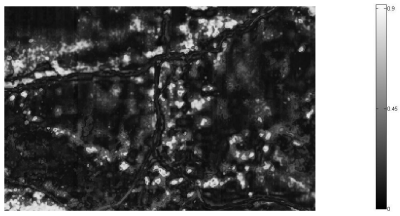
(a) Whyalla Dataset Original Image, imaged obtained from Google Maps



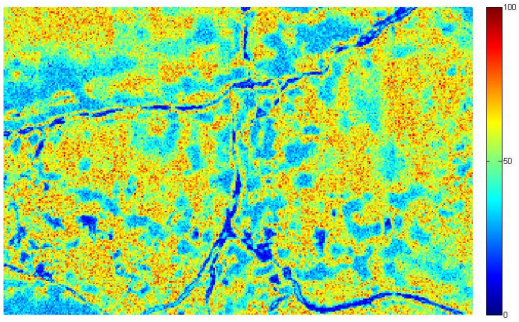
(b) *Road* class



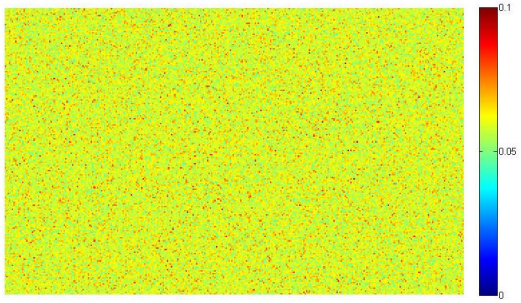
(c) *Desert* class



(d) *Scrub* class



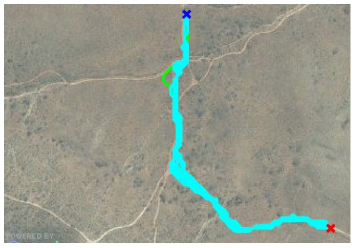
(e) Total costmap - Mean



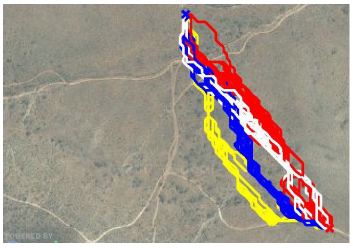
(f) Total costmap - Variance



(g) Magenta Path - take the left fork

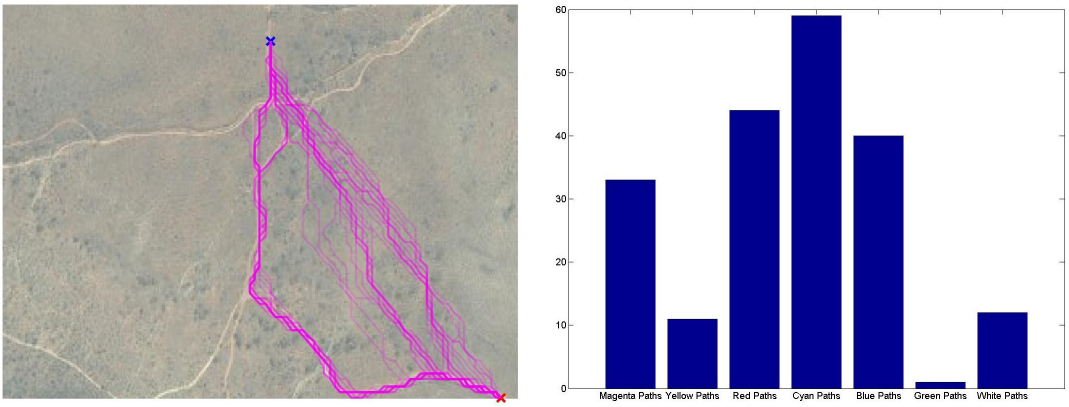


(h) Cyan Path - take the right fork



(i) Cross Country Paths

redness of the image means that in classifying on colour alone we are prone to high levels of confusion between the open *desert* and the slightly more yellow *road*. Green trees and *scrub* are easier to detect! The investigation of available sensors which would aid in the discrimination between classes is a matter for future work.



(j) Overall Path Distribution - an alpha blend of the 7 path types

(k) A histogram showing the distribution of paths on the Whyalla Dataset

Figure 5.15: Whyalla Dataset. A failure to adequately discriminate between the *desert* and *road* classes using only colour inputs to the classifier results in a high level of confusion between the two classes. This means we see a high proportion of the paths between the marked locations (both on the road) take a cross-country route across the desert. The cyan path of (h) that adheres completely to the road way is the most-likely path, demonstrating that the framework has some robustness to poor classification results.

5.5.2.5 Classifier Performance

To aid discussion of the results, a summary of the GP classifier's performance is presented. In Table 5.1 the percentage accuracy obtained for hold-out set of the training data is shown. Note that the poorest performance is observed on the Tempe and Whyalla datasets. This relatively poor performance was observed and discussed in the results of Sections 5.5.2.2 and 5.5.2.4 where the paths obtained did not always adhere to the least expensive terrain type.

	GP Overall Accuracy
Synthetic 1	98.67
Synthetic 2	100.00
Tempe	95.60
Lisbon	98.90
Whyalla	94.73

Table 5.1: Percentage accuracy of the multi-class Gaussian Process classifier when the classifier is applied to a hold-out set of the training data.

5.6 Chapter Summary

This chapter presented an approach to probabilistic cost map construction and path planning. Using two separate Gaussian process techniques we were able to classify locations in an aerial image probabilistically into one of n_C classes. Independently of classification we produce a cost map for each of the n_C terrain classes using a sparse Gaussian process regressor. The benefits of this approach are multifarious: we allow the cost of a terrain type to vary spatially, this reduces the classification burden as we can abstract away variations within broader terrain types into the independent spatial cost map and hence require fewer classes to represent terrain. We are also able to incorporate vague local knowledge (like *'avoid highways'*) into the spatial cost maps without having to know the exact location of every instance of that terrain type in the affected region. The Gaussian process spatial regressor produces a cost and

variance at each location in the map for each terrain type, this is combined with the probability of class membership at that point in a global probabilistic cost map. We can sample from the cost map to produce a distribution of probable paths between two points, as well as use the mean value of the cost function to find the most-likely path.

In this chapter we demonstrated how the resulting costmaps can be used to both respond to variable costs within one terrain type, and to generate a distribution of paths from the overall costmap. In a number of cases we noted that the results we obtained were negatively influenced by inconclusive results of the classification stage and that overall performance could be improved upon by using a more powerful classification technique. Whilst we obtained decent results using only colour red, green and blue pixel intensities for classification into three classes, we would expect better results using more inputs (such as hyper spectral data which would allow us to better discriminate between terrain such as vegetation or water). In turn, better paths could also be obtained by using more classes. This should not introduce much extra computational burden as the Gaussian process multi-class classification technique we employed scales linearly with the number of classes. However, pursuing this avenue was regarded as beyond the scope of this thesis which has focused on the validity of producing probabilistic costmaps. Scaling up the number of classes used was considered undesirable, as even with 3 classes and a limited amount of training data the classification procedure using the Variational Bayes MATLAB toolkit required the order of minutes to run on a standard desktop PC.

Although we chose to demonstrate the framework by planning over aerial images, it could be applied to other domains (such as on a mobile robot) simply by altering the input feature set used for classification. An obvious extension to this work would be to field the system on a real robot and use the onboard sensors as a feedback mechanism to update the probability of terrain class membership for cells in the costmap as they

are traversed. On a real robot it would also be possible to use mechanisms such as vibration sensors or computer vision tools to learn the spatial costmaps for each terrain type as the robot traverses the environment.

On a similar improvement related vein, we have used standard squared-exponential covariance functions in both the GPC and the GPR. Future work could encompass experimenting with different covariance functions. One would expect the use of non-stationary covariance functions in modeling the spatially varying cost map to produce better results, as these may better capture the abrupt local variations in terrain cost that we are interested in modeling.

In the next chapter we show how the probabilistic costmaps can be used to shape the paths planned by our chosen shortest path planner. We can use the information contained in our probabilistic costmap to construct informed heuristics to guide the planner in its initial journeys. Depending on how much risk we would like to take on in potentially sacrificing the shortest path for a longer, but still good, path - we are able to reduce the time spent searching for a path. Similarly, we are able to restrict the search to returning paths that only traverse well-known areas of the map.

Chapter 6

Risky Heuristics - Efficient Path Planning on Probabilistic Costmaps

In the previous chapter we demonstrated how Probabilistic Costmaps can be obtained from overhead imagery. We proved the framework to be useful in allowing the cost of a terrain type to vary spatially over the map - reducing the classification burden; and in generating a distribution of likely viable paths rather than just a single shortest path. In the context of the A*/D* family of search algorithms, the previous chapter can be thought of as making the input to the planner *probabilistic*. In this chapter we modify the algorithm itself to better respond to planning with uncertainty by focusing on the heuristic that guides the search. Through representing the heuristic as a probability distribution over cost we can speed up and direct the search by characterizing the risk we are prepared to take in gaining search efficiency while sacrificing optimal path length.

In the text that follows we discuss how a probabilistic heuristic can be constructed and show the performance gains that can be achieved over normal heuristic search methods. An existing algorithm, R^*_δ search (Pearl and Kim, 1982; Pearl, 1984) is applied for the first time to the problem of path planning over costmaps. This existing work is an extension of A* search, and provides performance guarantees when supplied with a heuristic that represents a probability distribution over the cost between a node

and the goal. The contribution in this Chapter is the computation of that probabilistic heuristic for the robot path planning domain (Section 6.3), and the novel application of the R^*_δ search to this area (Section 6.5).

6.1 Motivation

Typical approaches to path planning for field robots involve simplifying the robot's environment into a discretized costmap, where the cost of each grid cell is a scalar value proportional to the estimated cost of traversing the terrain thought to lie at that location. In the absence of perfect sensing and classification, this cost is actually an uncertainty distribution over the terrain associated with a particular cell. In Chapter 5 we argued that the typical 'scalar' approach to costmap creation throws away useful information, and presented a framework for producing probabilistic cost maps from aerial imagery.

In this chapter we capitalize on the probabilistic nature of these costmaps to 'speed up' path planning by providing guaranteed bounds on the sub optimality of paths produced when we trade the accuracy of the results for the speed of finding a potential path. We do this by making the heuristic that controls the search probabilistic. To recap Chapter 2, the A* and D* family of algorithms, commonly used in field robotics, both belong to the heuristic search family. The heuristic is used to estimate how far a particular node is away from the goal (in A*, in D* the direction of search is reversed so the heuristic estimates the distance to the start) and is used in conjunction with the known distance from the start to estimate the shortest path from start to goal through that node. This total distance is used to prioritize the node against all other nodes in the search in terms of its likely contribution in finding the shortest path and determines which of the nodes under consideration is chosen for expansion next. By focusing the search towards the goal, heuristic searches find optimal paths more

efficiently than comparable algorithms such as Dijkstra's search.

The more accurate the heuristic is, the less time we spend searching. It is particularly desirable when applying A* search to large graphs (such as those created by an overhead map) to restrict the point to point search to examining only relevant areas of the input graph. Recently a number of techniques have evolved which focus on preprocessing the graph to obtain better heuristics. The ALT (so named because of its use of A*, Landmark and the Triangle inequality) algorithm (Goldberg and Harrelson, 2005; Goldberg, 2007) precomputes distances between every cell in the map to and from a small set of *landmarks* distributed throughout the map. These precomputed distances are used in conjunction with the triangle inequality to provide a lower bound on the heuristic distance for arbitrary node-goal pairs during an A* search.

The LPI (Landmark Pathfinding between Intersections) algorithm (Grant and Mould, 2008) also uses landmarks to precompute heuristics, but solution paths are restricted to follow shortest paths stored between landmarks. Hierarchical Terrain representation for Approximately shortest Paths (HTAP) (Mould and Horsch, 2004) works by precomputing a hierarchy of abstracted graphs. At each level of the hierarchy a path is found between start and goal and this is used to constrain subsequent higher-resolution searches.

In this work we make use of precomputation to obtain accurate heuristics. However, in our case this precomputation of path lengths involves the addition and subtraction of Gaussian cell costs, which leads to a distribution which approximates the one we are interested in. This approximation governs the distance between the node and the goal.

More often than not in robotic path planning we are interested in finding a 'good' path rather than the shortest possible path. *Anytime* algorithms recognize this and a suite of solutions (Likhachev et al., 2003, 2005) have been developed to provide

good trajectories to the robot quickly and improve on them if time allows. These algorithms work by relaxing the admissibility criterion - which normally requires that the heuristic estimate of the goal distance *always* be less than or equal to the true distance. Figure 6.1 shows the quandary that the admissibility criterion introduces into our probabilistic planning domain. Let $p(f_n)$ be the probability density function over the cost of traversing a path from start to goal through node n. Here, $p(f_{n_1})$ has the lower mean, but it is quite possible that the actual path cost $p(f_{n_2})$ through node n_2 is somewhat lower due to the longer tail of its distribution. Intuitively we would opt to expand n_1 as the shape of the density functions renders $f(n_1) > f(n_2)$ unlikely. However, an admissible heuristic would force us to expand n_2 as the longer tail of this distribution means that to act otherwise would risk overestimating the cost to the goal. We propose to solve this quandary by using the heuristic to quantify the level of risk we wish to take that the paths returned by the planner will be suboptimal.

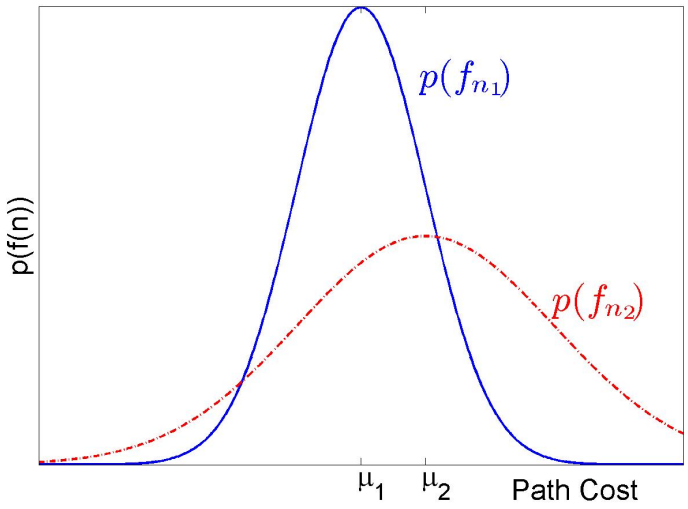


Figure 6.1: Which node to expand next? Node n_1 with cost function f_1 promises a lower mean, but f_2 indicates n_2 has significant probability of offering a lower cost than n_1.

Pearl and Kim (1982); Pearl (1984) developed the R^*_δ algorithm to deal with cases in which the user wishes to invoke probability distributions in the admissibility criterion. The premise is simple - if we know the probability distribution governing the heuristic, we know how often, on average, it will overestimate the distance between

the node and the goal and therefore we know how often it will overestimate the shortest path. The algorithm allows the specification of a level, δ which bounds the risk of returning a suboptimal path.

In this chapter we show how the R^*_δ algorithm can be applied to probabilistic costmaps. We use ALT to provide a good base estimate of the probability distribution governing the heuristic and learn a scaling parameter to improve this estimate on the fly. The contribution of this work is the novel application of ALT over a probabilistic graph (Section 6.3), and the application of R^*_δ to a new domain - path planning on costmaps (Section 6.5). The results show that for minimal precomputation we can obtain a 73% improvement in search efficiency and quantify the risk of the path being longer than the optimal path. Together with the work in Chapter 5, this chapter provides an end-to-end framework for incorporating uncertainty into path planning for mobile robots.

6.2 The Heuristic Tradeoff

Many modern planning algorithms (Koenig and Likhachev, 2002a; Ferguson and Stentz, 2005b; Stentz, 1994a) build on the idea of A* search (Hart et al., 1968). A* is a best-first graph search algorithm which finds the shortest path between a start and a goal node. Central to the A* algorithm and its variants is a distance-plus-cost heuristic which determines the order in which nodes of the graph being searched are visited. This heuristic itself has two parts, one is the cost of travelling from the starting node to the current node n under examination, usually denoted $g(n)$. The second part, denoted $h(n)$ is a heuristic estimate of the distance from n to the goal. The sum $f(n) = g(n) + h(n)$ determines the priority of the node n on the OPEN queue of nodes awaiting expansion. Refer to Section 2.3 for a description of the OPEN queue's role in the search.

Typically in the A* paradigm the heuristic must be consistent (monotonic) and admissible, meaning it never overestimates the cost of reaching the goal. It is the admissibility condition that guarantees that A* will find an optimal solution path if it exists. However, it has the unfortunate by-product of frequently leading the search to spend large amounts of time deliberating between roughly equal solution paths and does not give us the option of terminating the search with an acceptable but not optimal solution path.

If we have perfect knowledge of the problem space and can tailor the heuristic to make use of this knowledge, then $h = h_{optimal}$ and the fewest nodes are expanded. This is rarely achievable and so alternative means of generating solutions quickly are needed.

Note that the g and h components of the heuristic exhibit two competing tendencies. In one extreme, when $h = 0$ for every node, A* becomes a purely breadth-first search. At the other, with $g = 0$, decisions are based completely on the estimated proximity to the goal. While this may work well for a trusted heuristic function, often one is not available and by ignoring g we can be led into continuously trying to explore high cost fruitless paths which promise (but in reality cannot deliver) short paths to the goal.

Early work by Pohl (1970) and Harris (1973) set out to strike an optimal balance between these two tendencies by weighting the two components differently to induce the desired level of conservative / reckless search. Pearl (1984) proved the useful property that if the heuristic is consistent and the heuristic values are multiplied by a factor $(1 + \epsilon)$, the cost of the resulting solution is guaranteed to be within $(1 + \epsilon)$ times the cost of an optimal solution C_O.

$$C(t) \leq C_O(1 + \epsilon) \tag{6.1}$$

It is this idea that forms the basis for *Anytime* search algorithms (Likhachev et al., 2003, 2005; Hansen and Zhou, 2007) which generate initial solutions with large ϵ-values and continually rerun the search in an incremental fashion to give increasingly better results as time progresses.

The idea of placing risk bounds on heuristics was also introduced by Pearl and Kim (1982); Pearl (1984) who sought to invoke likelihood considerations into the admissibility guarantee. The resulting R^*_δ algorithm is a variant of A* that relaxes Harris' ε-admissibility condition even further, to allow the use of a precise estimator that may occasionally overestimate h^* by more than ε, such as is the case when the arc costs of a graph are known to be drawn from a probability distribution. R^*_δ provides a framework for bounding the suboptimality for different probability distributions which we build upon in this work.

In Pearl and Kim (1982) the performance of R^*_δ was demonstrated on an instance of the Travelling Salesman Problem with statistics generated from multiple different distance matrices. (A distance matrix gives the distance between the different cities that a salesman needs to travel between on a tour that visits each city only once. Each instance of the Travelling Salesman problem has a unique distance matrix). Pearl used regression techniques and the results of previous searches to fit parameters to a best fit model $\hat{h}_{opt}$, a heuristic that accurately estimates the future cost of the search but occasionally overestimates this distance. The true minimum arc cost h is assumed to be normally distributed with mean $\hat{h}_{opt}$ and variance σ^2 estimated from the statistics.

Obviously, the computation of heuristics for a Travelling Salesman search bears little relevance to problem domains encountered by a mobile robot. What we wish to demonstrate in this paper is the effectiveness of risk bounded search in searching terrain that may be traversed by a mobile robot. In Chapter 5 we demonstrated how probabilistic maps can be obtained from aerial imagery and probabilistic cost

functions for the different terrain classes contained in the map. In the work that follows we will demonstrate how these probabilistic maps can be used to construct probabilistic heuristics for use with R^*_δ search.

6.3 Precomputing a Probabilistic Heuristic

The first stage of our risk-bounded search strategy is to obtain an input graph where the arc costs are probability distributions. The nature of the problem domain is such that negative arc costs are not permitted. The reason for this is two fold - with negative edge costs the optimality guarantees of A* and its variants are voided. In addition, we are making the assumption that it will always cost something for a robot to traverse terrain.

We assume we have a set of arc costs obtained using the approach in Chapter 5, where the cost of a grid cell is obtained by multiplying the probability of its class membership with the probability density associated with the cost of that class in that particular area. We then use ALT to precompute probabilistic heuristics. Consider the landmark L in Figure 6.2: if $d(\cdot)$ defines the distance *to* L, then by the triangle inequality:

$$d(u) - d(v) \leq dist(u, v), \tag{6.2}$$

else if $d(\cdot)$ defines the distance *from* L, we have

$$d(v) - d(u) \leq dist(v, u). \tag{6.3}$$

The largest lower bound over all landmarks is used to select the heuristic which will guide the A* search.

The mean of the Gaussian distributions is used in conjunction with Dijkstra's search to precompute distances to and from every point in the map to the selection of

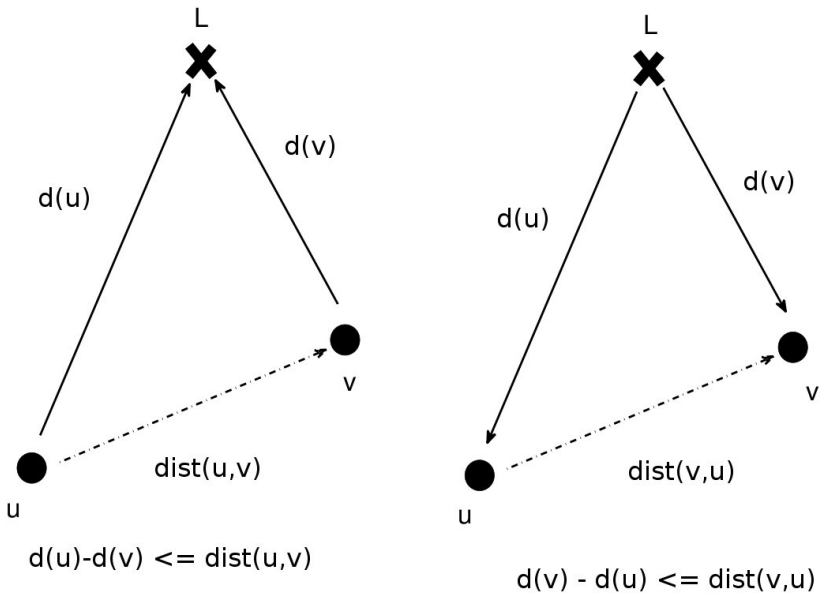


Figure 6.2: Landmarks: The triangle inequality provides an upper bound on the distances between nodes in the graph.

landmarks. The variances over the path so calculated are also summed. Neighbouring cell costs are added together leading to a cost over the entire path from node n to the landmark (L_k) of K landmarks:

$$\hat{L}_{k\in K}(n) = \mathcal{N}\left(\sum_{i\in path} u_i, \sum_{i\in path} \sigma_i^2\right). \tag{6.4}$$

We use this estimate of the distance between node n and the landmark k in conjunction with equations (6.2) and (6.3) to estimate the distance from the node to the goal. The 'most accurate' lower bound is provided by taking the maximum heuristic estimate over the K landmarks we are evaluating.

$$\hat{h}_{ALT}(n) = \max_{k\in K}\left(\hat{L_k}(n) - \hat{L_k}(g), \hat{L_k}(g) - \hat{L_k}(n)\right) \tag{6.5}$$

Note that this distribution is an approximation to the distribution we are actually interested in, over the distance between the node and the goal. As Figure 6.3

illustrates, depending on where the landmark is located in relation to the node and the goal influences how accurate the landmark's estimation of the goal distance is.

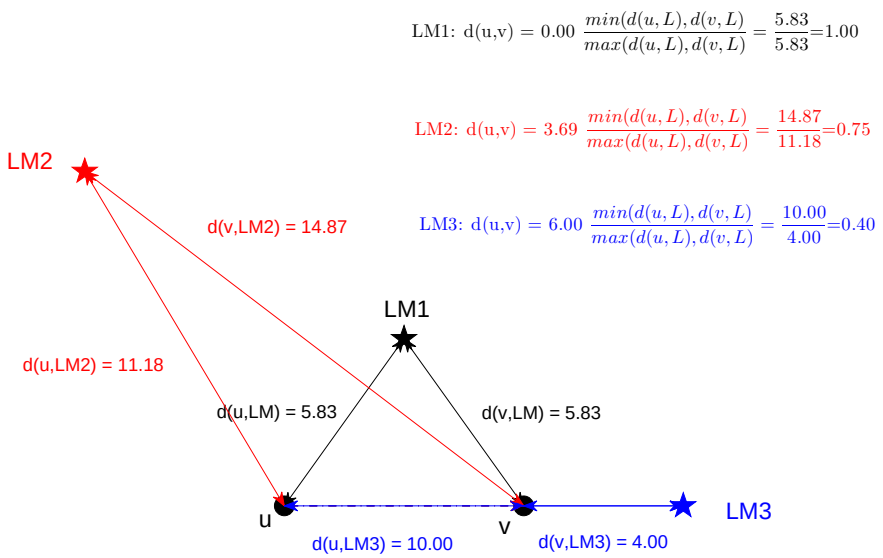


Figure 6.3: Three different landmarks produce widely varying estimates for the true distance (6.00) between nodes u and v. While they all produce valid lower bounds, the correct distance is provided only by landmark $LM3$ which correspondingly exhibits the smallest ratio of min to max distances between u, v and the landmark. The problem of the accuracy of the lower bound being dictated by the relative position of the node, goal and landmark is addressed by pushing this ratio through a sigmoidal function to 'scale up' estimates such as those provided by $LM1$ in this instance, while leaving correct estimates (such as $LM3$) untouched.

We account for this by scaling the variance of the heuristic estimate by a factor φ, which is dependent on the ratio of the lengths of the sides of the triangle used to compute the estimate.

$$c = \frac{\min\left[d(u), d(v)\right]}{\max\left[d(u), d(v)\right]} \tag{6.6}$$

$$\varphi = \frac{1}{1 + e^{-c}} + 1 \tag{6.7}$$

In order to guarantee the probabilistic bounds detailed later in this section, we also need to correct for the consistent overestimation of the mean by the ALT heuristic. This is because the ALT heuristic always produces an estimate of the node-goal cost *via a landmark*. It is highly unlikely that this is ever the shortest distance between

node and goal. We introduce small scaling parameters for both the mean (τ) and the variance (into which we incorporate φ) and learn these parameters by conducting multiple searches and comparing the results to optimal paths found by A* search using an admissible (euclidean) heuristic.

$$\hat{h} = \mathcal{N}(\mu_{\hat{h}_{ALT}} - \tau, \beta(\sigma^2_{\hat{h}_{ALT}})) \tag{6.8}$$

Learning the parameters is done once per environment. In our experience the searches converge to a 'good' estimate within approximately 15 searches.

6.4 Proof of Tradeoff Risks

In analyzing the risk we adapt the analysis provided in Pearl and Kim (1982).

If we take $h(n)$ to be the minimum arc cost from node n to the goal, we know that equation 6.8 has provided an estimate of the minimum arc cost in the form of a probability density function $\hat{h}(n)$. We can choose to interpret this as the likelihood of h given our precomputed estimates $\hat{h}$, and denote the likelihood by $p(h \mid \hat{h})$. In the course of carrying out an $A*$ search we observe $\hat{g}(n)$, which is the best known approximation to $g(n)$ - the minimum cost of navigating from the start s to node n. Note that this is a scalar value. Knowledge of $\hat{g}$ induces a conditional density function on $f^{\dagger}(n)$, the cost of a path from the start to goal via n.

$$f^{\dagger}(n) = \hat{g}(n) + h(n) \tag{6.9}$$

This leads to a probability distribution over $f^{\dagger}$:

$$p(f^{\dagger} \mid \hat{g}, \hat{h}) = \hat{g} + p(h \mid \hat{h}). \tag{6.10}$$

Every optimal solution path must be a continuation of a path $T(n)$ which passes through some node currently contained on the OPEN list. Therefore we can say that OPEN always contains a/some node/s for which $f^\dagger(n) = f(n) = C_{opt}$, where C_{opt} denotes the optimal shortest path length.

Termination conditions for a normal, scalar-driven A^* search are straightforward. The search continues until the goal node has the lowest f-value of any node in the OPEN list. Now that our OPEN list is ordered by probability distributions, the decision as to when to terminate the search becomes more complicated.

Figure 6.4 illustrates the dilemma. Suppose the best path found so far has a cost C, shown by the straight vertical line on the graph. The node n at the top of the OPEN list has an f-value given by the Gaussian distribution shown. Expanding n shows some promise of coming up with a path shorter than C, but on average n would produce a longer path as the mean of the distribution $f^\dagger$ is greater than C. The decision as to whether to explore n or not thus involves evaluating the *risk* of terminating the search at cost C.

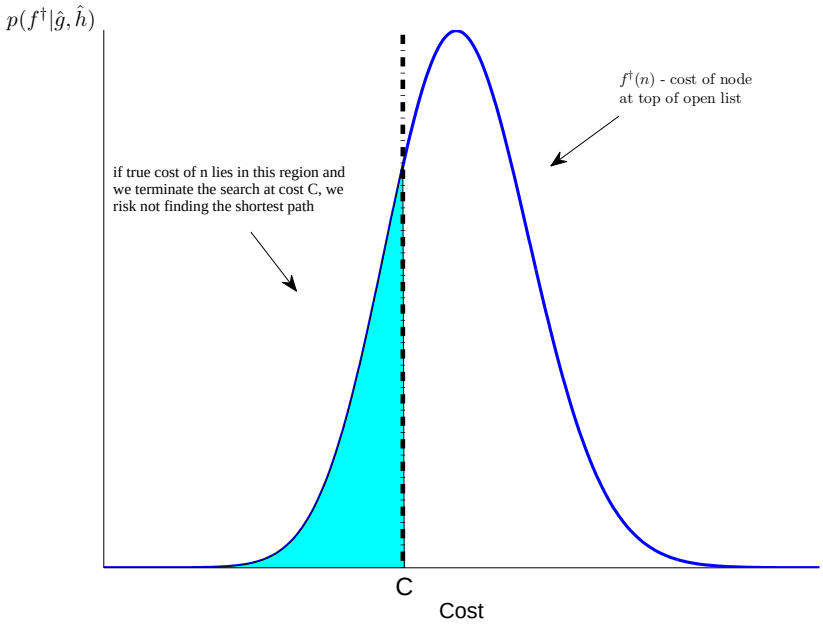


Figure 6.4: Under R^*_δ search, the decision to terminate the search becomes a matter of weighing up the risk of leaving unexplored nodes on the OPEN queue.

Under the R^*_δ paradigm, the risk of missing further cost reduction is characterized by a Risk Function $R(C)$ which depends on both C and $p(f^\dagger \mid \hat{g}, \hat{h})$. It is a non-decreasing function of C. In Pearl and Kim (1982), three types of Risk Functions are described:

1. *Worst Case Risk*

$$R_{WC}(C) \triangleq \max_{f:p(f^\dagger|\hat{h},\hat{g})>0} (C - f^\dagger) = C - \hat{g} - h_a \tag{6.11}$$

   where h_a is the lowest imaginable value of $p(h|\hat{h})$, the lowest h with positive density. As we are dealing with Gaussian distributions, this h_a will always be 0 and as such will add nothing to the heuristic search over a Dijkstra search. We did not examine the Worst Case Risk in our studies.

2. *Probability of Suboptimal Termination*

$$R_{ST}(C) \triangleq p(C - f^\dagger > 0) = \int_{y=-\infty}^{C} p(f^\dagger \mid \hat{h}, \hat{g}) dy \tag{6.12}$$

3. *Expected Risk*

$$R_{ER}(C) = \int_{f^\dagger=-\infty}^{C} (C - f^\dagger) p(f^\dagger \mid \hat{h}, \hat{g}) dy \tag{6.13}$$

The form of the Risk Functionals as a function of cost is depicted in Figure 6.5.

An R^*_δ search imposes the requirement that the underlying $A*$ search will continue until no node on the OPEN list has a risk associated with it that is greater than some level δ. This introduces the notion of *δ-risk admissibility*, guaranteeing that the search always terminates at a cost C such that $R(C) \leq \delta$ for all nodes left on OPEN.

This means that instead of using f-values to order the OPEN list, we use a *threshold cost function* $C_\delta(n)$ which is given by the solution to the equation

$$R(C) = \delta. \tag{6.14}$$

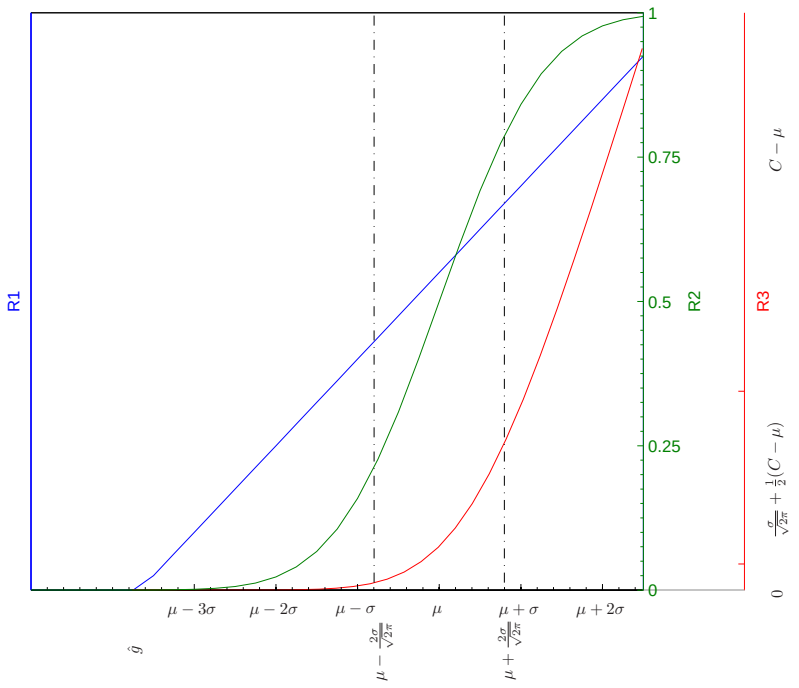


Figure 6.5: The worst case, suboptimal termination and expected risk functionals graphed as a function of cost. Note that 3 different scales are represented on the y-axis.

When using the *Probability of Suboptimal Termination* risk functional, we choose δ to be the probability of obtaining a suboptimal solution which we are prepared to accept. For instance, if we were prepared to risk obtaining a suboptimal solution 5 out of every 100 iterations of a particular problem instance we would set δ to 0.05. When using the *Expected Risk* functional, δ represents the additional cost of the path over that of the optimal path which we are prepared to accept. As equation (6.19) will show, it is often preferable to express this as a percentage of optimal path length.

Once we have chosen our risk functional and the level of risk we are prepared to accept, we need to translate this into a *threshold cost function* value $C_\delta(n)$ - that can be computed from $\hat{g}$ and $p(h \mid \hat{h})$ to be used in ordering the OPEN list in our A*-like search.

For the $R_{ST}(C)$ risk functional, it makes sense to use the properties of the Gaussian

distribution and write δ as an expression of the distance from the mean.

$$C_\delta(n) = \begin{cases} \mu & \delta = 0.5 \\ \mu - \sigma & \delta = 0.159 \\ \mu - 2 * \sigma & \delta = 0.023 \end{cases} \tag{6.15}$$

Evaluating the $R_{ER}(C)$ functional for a Gaussian leads to the following piecewise-linear approximation:

$$R_{ER}(C) = \begin{cases} 0 & C < \mu - \frac{2\sigma}{\sqrt{2\pi}} \\ \frac{\sigma}{\sqrt{2*pi}} + \frac{1}{2}(C - \mu) & \mu - \frac{2\sigma}{\sqrt{2\pi}} < C < \mu + \frac{2\sigma}{\sqrt{2\pi}} \\ C - \mu & \mu + \frac{2\sigma}{\sqrt{2\pi}} < C \end{cases} \tag{6.16}$$

which can be solved for values of $C_\delta(n)$

$$C_\delta = \begin{cases} \mu + 2\left(\delta - \frac{\sigma}{\sqrt{2\pi}}\right) & 0 < \delta < \frac{2\sigma}{\sqrt{2\pi}} \\ \mu + \delta & \frac{2\sigma}{\sqrt{2\pi}} < \delta \end{cases} \tag{6.17}$$

As alluded to earlier it is more useful to express the risk as a percentage of the solution cost

$$\delta = \frac{R(C')}{C'} \tag{6.18}$$

which leads to the following expression for C'_δ

$$C'_\delta = \begin{cases} \frac{\frac{2\sigma}{\sqrt{2\pi}} - \mu}{(2\delta - 1)} & 0 < \delta < \frac{\frac{2\sigma}{\sqrt{2\pi}} - \mu}{2\left(\frac{2\sigma}{\sqrt{2\pi}} + \mu\right)} + \frac{1}{2} \\ \frac{\mu}{(1-\delta)} & \frac{\frac{2\sigma}{\sqrt{2\pi}} - \mu}{2\left(\frac{2\sigma}{\sqrt{2\pi}} + \mu\right)} + \frac{1}{2} < \delta. \end{cases} \tag{6.19}$$

6.5 Results

We implemented the risk heuristic searching algorithms in C++ and ran them on a standard PC with 4G RAM and a dual core 1.8GHz processor. Landmarks were generated using the *planar landmark selection* method (Goldberg and Harrelson, 2005; Goldberg and Werneck, 2009). This method stems from the observation that placing a landmark behind the destination tends to generate good results in geometric graphs.

6.5.1 Obtaining an approximation to the true heuristic

This first set of results justifies the choice of heuristic approximation described in Equation 6.8. The performance of four means of approximating the heuristic ($\hat{h}$) are compared, they are:

- the unmodified ALT distance estimate,
- the ALT distance estimate with the variance adjusted by φ to account for landmark accuracy using Equation 6.7,
- the ALT distance estimate with the variance adjusted by $\beta = c_\sigma * \varphi$ as per Equations 6.8 and 6.21, and
- the ALT distance estimate with the mean adjusted by τ and variance adjusted by β according to Equations 6.8, 6.20 and 6.21.

Each of the four heuristics were then used in R^*_δ searches using three different estimators: $h = \mu_{\hat{h}}$, $h = \mu_{\hat{h}} - \sigma_{\hat{h}}$ and $h = \mu_{\hat{h}} - 2\sigma_{\hat{h}}$. The best performing heuristic (and therefore that which provides is the best estimate of the true distance between nodes and goal) is judged to be that which closely approximates the results we expect to see under each estimator given that the heuristic represents a gaussian distribution over the distance between a node and goal. Namely:

- 97.7% of solutions return the optimal path length under the $h = \mu_{\hat{h}} - 2\sigma_{\hat{h}}$ estimator,
- 84.1% of solutions return the optimal path length under the $h = \mu_{\hat{h}} - \sigma_{\hat{h}}$ estimator,
- 50.0% of solutions return the optimal path length under the $h = \mu_{\hat{h}}$ estimator.

The graph in Figure 6.6 compares the results obtained over a 65×65 grid graph with the various approximations to the heuristic. Eight landmarks were used in precomputing the search. Each data point in the graph is the average result obtained over repeated searches between the same start and goal node after sampling 50 distinct instances from the probabilistic cost map distribution. The 30 data points represent 30 different start/goal combinations. The three graphs show the results of searching using the mean minus 2 standard deviations (where 97.7% of searches should return the optimal path), the mean minus one standard deviation (84.1% should be optimal) and the mean (50% optimal). The graphs clearly show that the method of scaling both the mean and variance of the heuristic as per Equation 6.8 produces a distribution nearly identical to the true distance between the node under consideration and the goal - because with this heuristic alone we obtain the expected performance in terms of the number of optimal solutions for all three estimators.

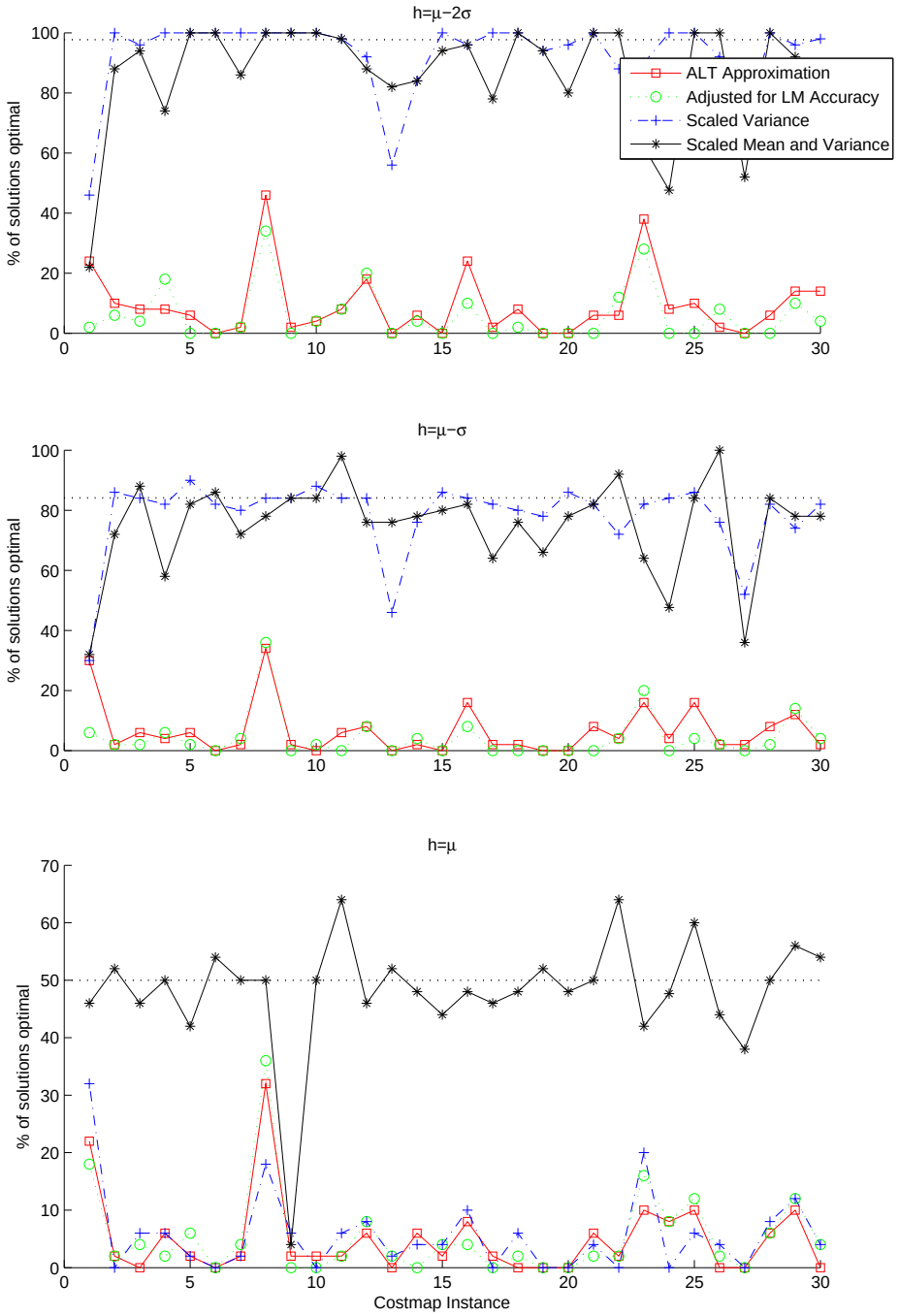


Figure 6.6: Approximating the heuristic distribution. The dotted horizontal lines show expected % accuracy for each of the $h = \mu_{\hat{h}} - 2\sigma_{\hat{h}}$, $h = \mu_{\hat{h}} - \sigma_{\hat{h}}$ and $h = \mu_{\hat{h}}$ bounds. The graphs show that scaling the heuristic estimate obtained via ALT produces many-fold improved results over the raw estimate. The bottom graph illustrates the importance of shifting the mean to account for consistent (but slight) overestimation by ALT.

6.5.2 R^*_δ Search Results with Initial Heuristic Approximation

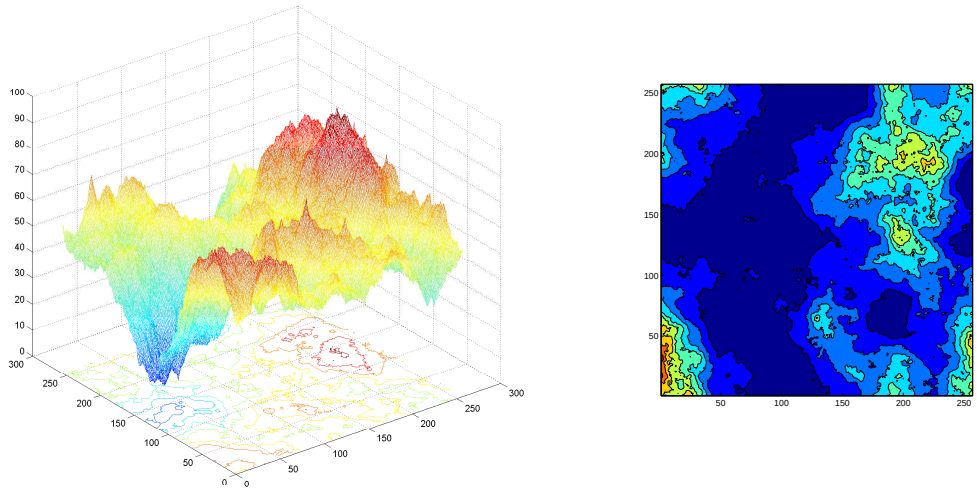
To test the performance of the R^*_δ algorithm itself we generated a 257×257 random fractal terrain map (Figure 6.7) for both the mean and variance of the cells. This was done to ensure that there would be distinctive areas of high and low variance in the map - rather than randomly scattered throughout - and that this would reflect real world situations where certain regions would be better known than others. We placed 16 landmarks using planar landmark selection.

Here we compare the results of our risk based searches:

- Suboptimal Termination Risk Functional with
  - $h = \mu_{\hat{h}}$, called ST0
  - $h = \mu_{\hat{h}} - \sigma_{\hat{h}}$, called ST1 and
  - $h = \mu_{\hat{h}} - 2\sigma_{\hat{h}}$, called ST2.
- Expected Termination Risk Functional with
  - $\delta = 5\%$ in excess of optimal path length, named ER5
  - $\delta = 10\%$ in excess of optimal path length, named ER10
  - $\delta = 15\%$ in excess of optimal path length, named ER15
  - $\delta = 20\%$ in excess of optimal path length, named ER20
  - $\delta = 30\%$ in excess of optimal path length, named ER30
  - $\delta = 40\%$ in excess of optimal path length, named ER40

against that of a standard A* search guided by a euclidean distance heuristic.

Precomputing the landmarks took an average of 2.58 seconds per landmark - this involved running a Dijkstra search in both forward and reverse directions over the graph, with the edge lengths set to Gaussian values. By way of comparison, A*



(a) Mean Value of Terrain (b) Variance of Terrain

Figure 6.7: The random fractal terrain map used for the experiment

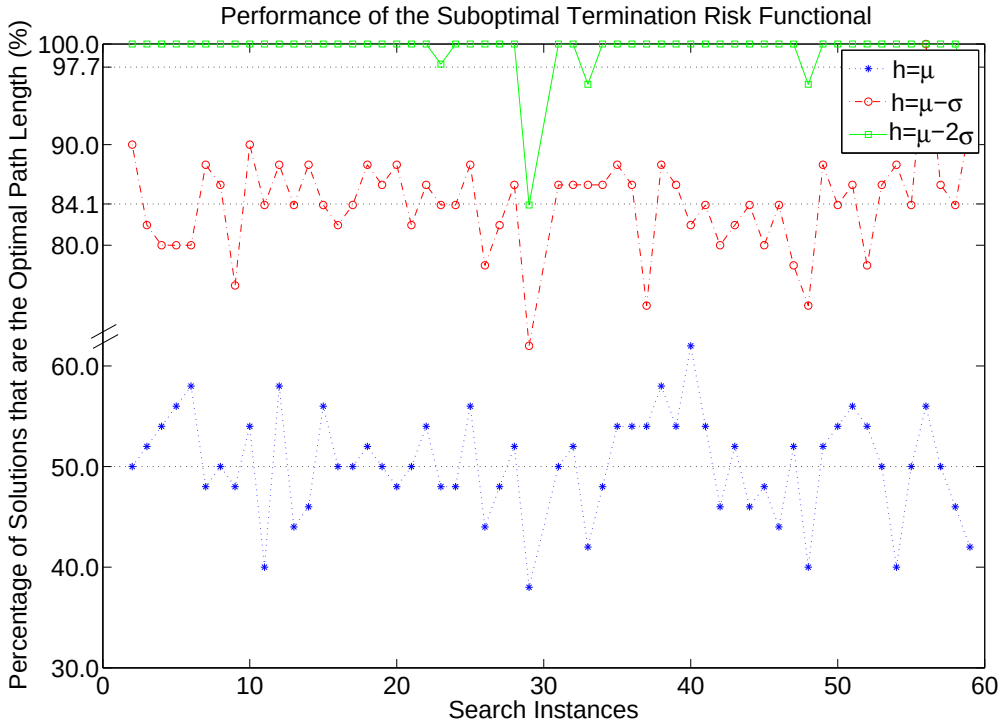
searches with euclidean distance over the graph took an average of 2.16 seconds. So precomputation took approximately 19 times as long as one search.

We performed 60 different start to goal searches, and at each iteration generated 50 different samples from the probabilistic cost map to test the operation.

6.5.2.1 Suboptimal Termination Risk Functional

Figure 6.8(a) graphs the results of the R^*_δ search over the 60 instances with varying levels of suboptimal risk allowed. Overall, as the table of Figure 6.8(b) illustrates, the ST2 heuristic achieved 96% accuracy; ST1 achieved 81.2% and ST0 achieved 50.2%, close to the expected values of 97.7%, 84.1% and 50.0% respectively. Figure 6.9 illustrates the efficiency savings of these heuristic values - on average they require 95.4%, 97.3% and 27.43% of the search effort respectively and produce normalized path lengths of $1.0002, 1.0006$ and 1.002 multiples of that of the A* search. Due to the probabilistic nature of the risk based search, ST1 performs worse than ST0. However, our results show that using the ST0 search heuristic guarantees the shortest path distance 50% of the time and expands only 27% of the cells that an equivalent

A* search would do - an efficiency saving of 73%.



(a) Percentage of sample searches that returned the optimal path length for each of the 60 start-goal queries.

	Suboptimal Termination Heuristic		
	ST0 μ	**ST1** $\mu - \sigma$	**ST2** $\mu - 2\sigma$
Expected % Optimal Solutions	50.0	84.1	97.7
Experimental % Optimal Solutions	50.2	81.2	96.0
Average Path Length Excess (%)	0.0159	0.0616	0.0197
Average Normalized Search Effort (%)	27.4	97.3	95.4

Key:

Average Path Length Excess is measured as % in excess of the optimal path length

(b) Performance of R^*_δ search with varying levels of suboptimal risk. These aggregate results show that in 60 different start-to-goal searches over the one map, our R^*_δ search achieved close to the expected levels of optimal solutions. This demonstrates that our heuristic approximation computed using ALT search and scaled according to equation 6.8 is a very good approximation to the exact heuristic for this problem.

Figure 6.8: Performance of R^*_δ with suboptimal termination

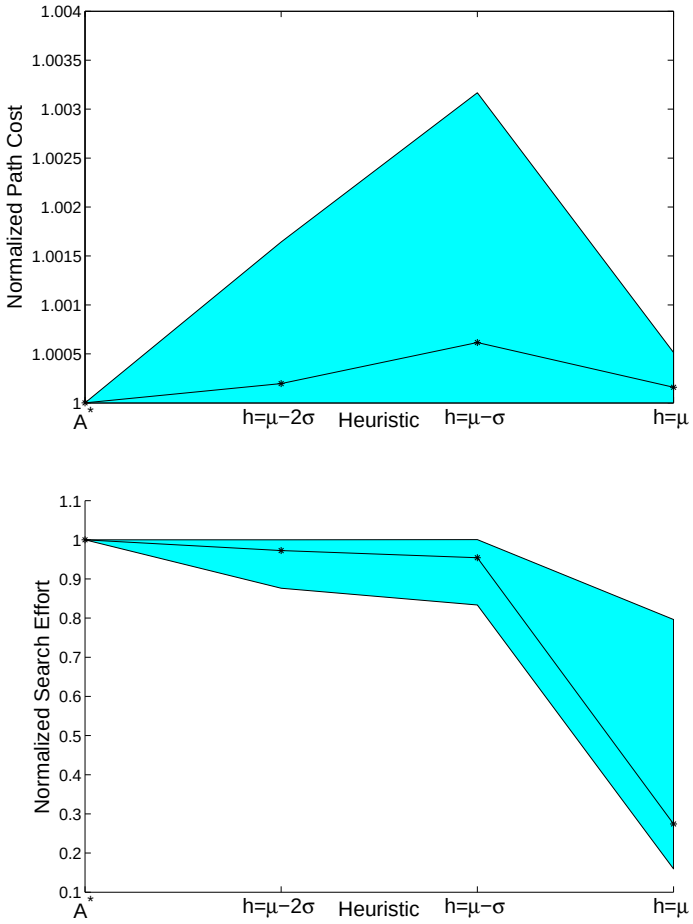


Figure 6.9: The results of running R_δ search with three different risk functional levels as per Equation (6.15). The top plot is the normalized path cost with the solid line representing the mean of the normalized path cost and the filled section denoting the spread of values obtained within one standard deviation of the mean. The bottom plot is the normalized search effort in terms of nodes expanded. The solid line denotes the average search effort, the filled section denotes one standard deviation from the mean search effort.

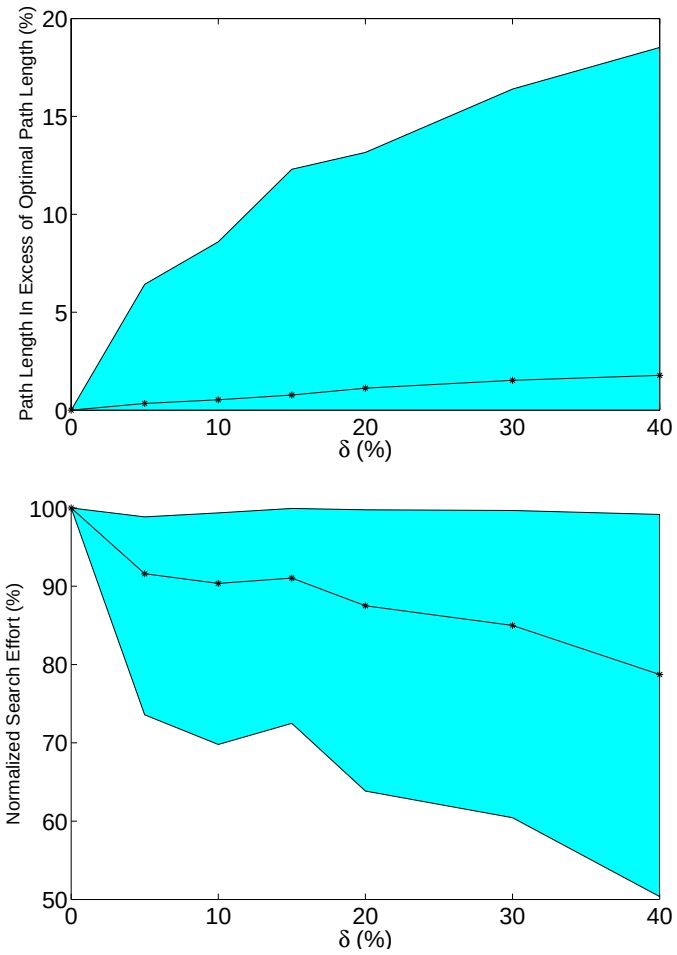
6.5.2.2 Expected Risk Functional

Figure 6.10 shows the results of running 60 iterations of the search with varying levels of expected risk allowed. Here, δ is used to bound the percentage over which the path length returned by R^*_δ exceeds the optimal path length. The graph shows that even with δ set to 40%, on average the path length is only 1.77% in excess of that achieved by A* with an admissible heuristic (the baseline), and at worst it is 18.5% longer. However it does provide an efficiency saving, it only expands 78.7% of the nodes of

the baseline.

Overall though, the *average* performance of this heuristic is disappointing. While it performs within the specification - the worst case performance of the heuristics never exceed the stipulated bounds, it does not perform anywhere near as well as the suboptimal termination heuristic with $h = \mu$. As table 6.8(b) shows, the ST0 heuristic on average exceeds the optimal path length by only 0.02%, yet requires only 27.4% of the search effort of the baseline A* euclidean search. In comparison, the most efficient Expected Risk heuristic (ER40) allows for a worst case error of 40%, but on average returns paths that are 1.77% longer than optimal at a search effort 78.7% of that of the baseline. So for a worse performance than the ST0 heuristic, the best Expected Risk functional requires almost 3 times the search effort.

It is possible that this particular costmap does not exhibit enough variance in cell values to make the Expected Risk functional an efficient option. An indication of this is the failure of the worst case performance of the high risk heuristics (i.e. 30%, 40%) to approach their stipulated limits. The suitability of this risk functional to the type of costmaps generated in robotic path planning is a matter for future consideration.



(a) The results of running R^*_δ with 6 levels of expected risk

	Expected Risk Heuristic (δ(%))					
	ER5 5%	**ER10 10%**	**ER15 15%**	**ER20 20%**	**ER30 30%**	**ER40 40%**
Average Path Length (%)	0.34	0.53	0.77	1.12	1.52	1.77
Worst Case Path Length (%)	4.43	8.59	12.30	13.16	16.40	18.52
Average Search Effort (%)	91.60	90.36	91.04	87.51	85.00	78.72

Key:

Average Path Length as measured as % in excess of the optimal path length
Worst Case Path Length as measured as % in excess of the optimal path length
Average Normalized Search Effort is measured % of that required by A* with euclidean heuristic

(b) Performance of R^*_δ search with varying levels of expected risk.

Figure 6.10: R^*_δ with Expected Risk Heuristic

6.5.3 Learning the Scaling Parameters

Before we progress it is important to revisit the scaling parameters used to aid in approximating the true underlying heuristic distribution. In equation 6.8 we stated that these scaling parameters are used to shift the mean and scale the variance of the ALT heuristic towards a distribution which better approximates the true heuristic distribution.

Restating equation 6.8 here,

$$\hat{h} = \mathcal{N}(\mu_{\hat{h}_{ALT}} - \tau, \beta(\sigma^2_{\hat{h}_{ALT}}))$$

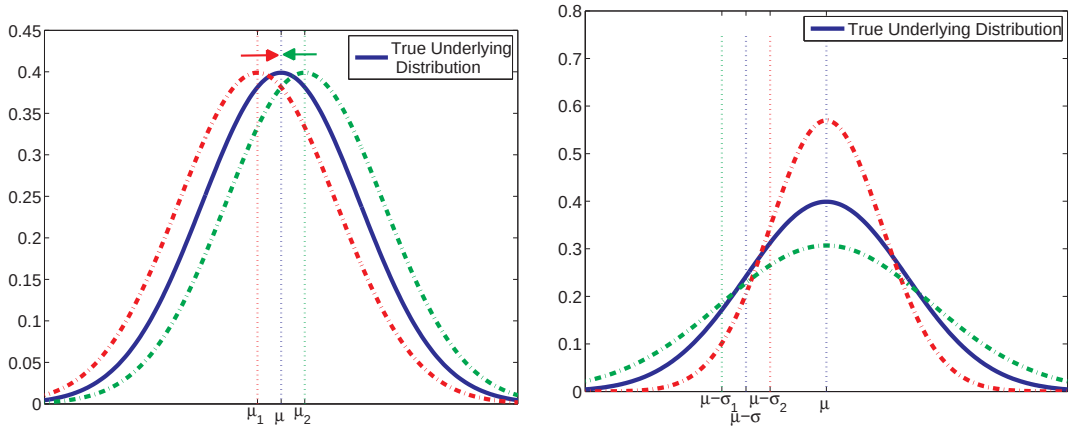
we add that the scaling parameter used to shift the mean is a constant fraction (c_μ) of the mean of the ALT heuristic $\mu_{\hat{h}}$.

$$\tau = c_\mu \mu_{\hat{h}_{ALT}} \tag{6.20}$$

The scaling parameter used to scale the variance is composed of a constant c_σ multiplied by the corrective factor φ described in Section 6.3 which accounts for the influence the orientation of the landmark, node under exploration and goal has on the accuracy of the heuristic estimate.

$$\beta = c_\sigma \varphi \tag{6.21}$$

Initially, we set $c_\mu = 0.1$ and $c_\sigma = 1$. We then ran multiple searches using both the ST0 ($h = \mu_{\hat{h}}$) and ST1 $h = \mu_{\hat{h}} - \sigma_{\hat{h}}$ *inadmissible* suboptimal termination heuristics together with an A* search with an *admissible* euclidean distance heuristic to obtain the true shortest path for comparison purposes. We kept statistics of the percentage of cases the two suboptimal termination heuristics returned the shortest path. If ST0 started to exceed the expected result of 50% correct solutions, we scaled down c_μ by



(a) Scaling the Mean. The unknown, underlying distribution is represented with a solid blue line. If the approximated mean is too low (red distribution) then more than 50% of the true distibution will lie above the mean and the R^*_δ search will return more than the expected fraction of optimal solutions. This means that in correcting for our approximation we have subtracted too much from the mean and need to reduce τ of equation 6.20 by scaling down c_μ. If we start to see less than the expected percentage of optimal solutions then the opposite is true, our approximated distribution resembles the green distribution, and we need to scale up c_μ.

(b) Scaling the Variance. If the variance is too great (green distribution) then more than 84.1% of the true distribution (blue solid line) will lie above our estimated $\mu-\sigma$ bound (green vertical dotted line). We will see the R^*_δ search return more than the expected level of optimal solutions using the $h = \mu - \sigma$ and $h = \mu - 2\sigma$ heuristics. To remedy this, we scale down c_σ. If the converse is true and our estimated distribution takes the form of the red distribution, then we have less of the true underlying distribution lying above the $h = \mu - \sigma$ bound than expected. We will see less optimal solutions than we expect, and need to remedy this by scaling up c_σ.

Figure 6.11: Adjusting the scaling parameters based on the evidence of the true underlying distribution.

10%, if it trended under 50% correct responses then we scaled up c_μ by 10%. Likewise, if ST1 trended above 84.1% we would scale down c_σ by 10%, and conversely if it was below that level c_σ would be scaled up by 10%. Figure 6.11 illustrates the principle.

Note that we *never* had ground truth values for the actual heuristic values to compare to. A grid of 257×257 cells has 66049 cells and a significant portion of them are examined during the course of the search, and the heuristic is evaluated for each node examined. Finding a ground truth value would mean computing a complete search (such as A* with euclidean heuristic) from *each node* considered in the course of the search to the goal - a computationally expensive process! Instead, our method of learning relied on the overall outcome of the search to crudely scale up or down

the scaling parameters.

In the results detailed in Figures 6.8(a), 6.9, 6.10 and Table 6.8(b) above, the process of learning the scaling parameters continued throughout all 50 samples for all 60 search instances. Note that this does not constitute overfitting, as we *never* had access to the ground truth data. However, this method of taking a gross average and adjusting the scaling parameters accordingly is reliant on conducting an admissible search at every instant to compare the result to. While the results of Figures 6.8(a), 6.9, 6.10 and Table 6.8(b) show that it is possible to accurately approximate the underlying heuristic, this method has little practical value!

Our next move was to investigate learning good 'average' average parameter values for a given map using a training set of queries, and testing those values on a hold-out set of different queries. Figure 6.12 graphs the values of the scaling parameters obtained over the 60 queries used in Section 6.5.2. We chose to use the median of these values for the hold out set, because the mean, especially in the case of the c_μ parameter, would be overly-influenced by the few large outlier data points in Figure 6.12. The median value is plotted as the dotted line.

We used these median values as the scaling parameters for searches over a hold-out set of 120 further queries. Figure 6.13(a) and Table 6.13(b) show that the approximation to the true underlying heuristic is reasonably accurate. Close to expected results are obtained with the ST0 heuristic, but more than the expected number of optimal results are obtained with the ST1 and ST2 heuristics. This means that the c_μ parameter value is close to optimal, but the c_σ value obtained from the median values of the training set is too high.

Figure 6.13 shows the normalized path costs and search effort obtained for each of the 3 suboptimal termination heuristics, together with that of the normalizer - the A* search. Note here, that as in Figure 6.9, both the average and worst case Normalized Path Costs are *worse* for the ST1 heuristic than they are when using the

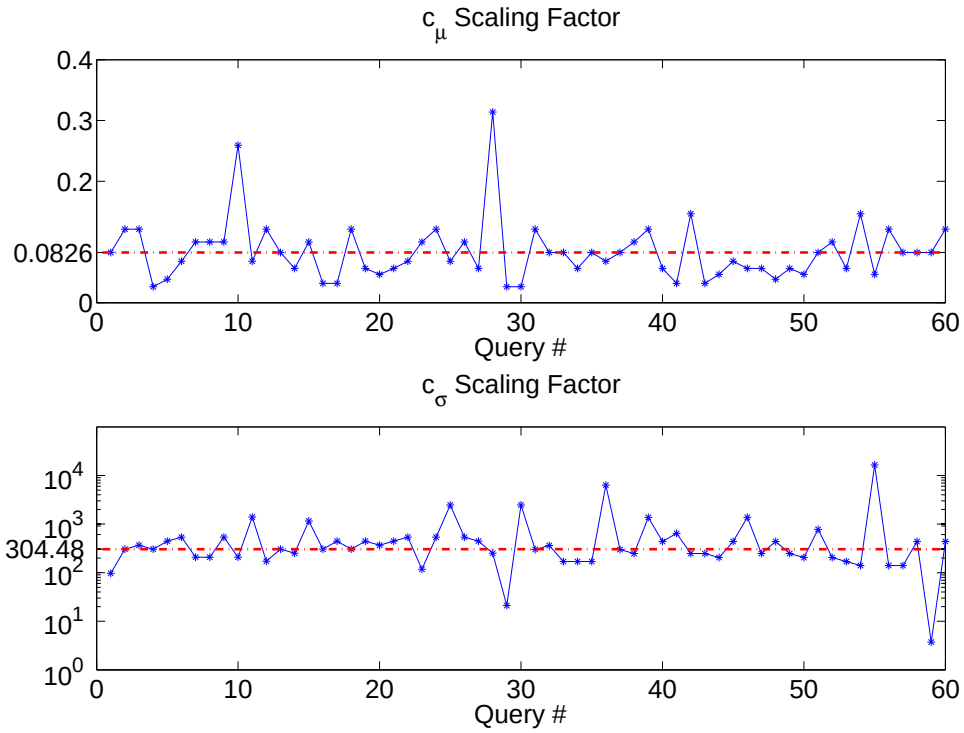
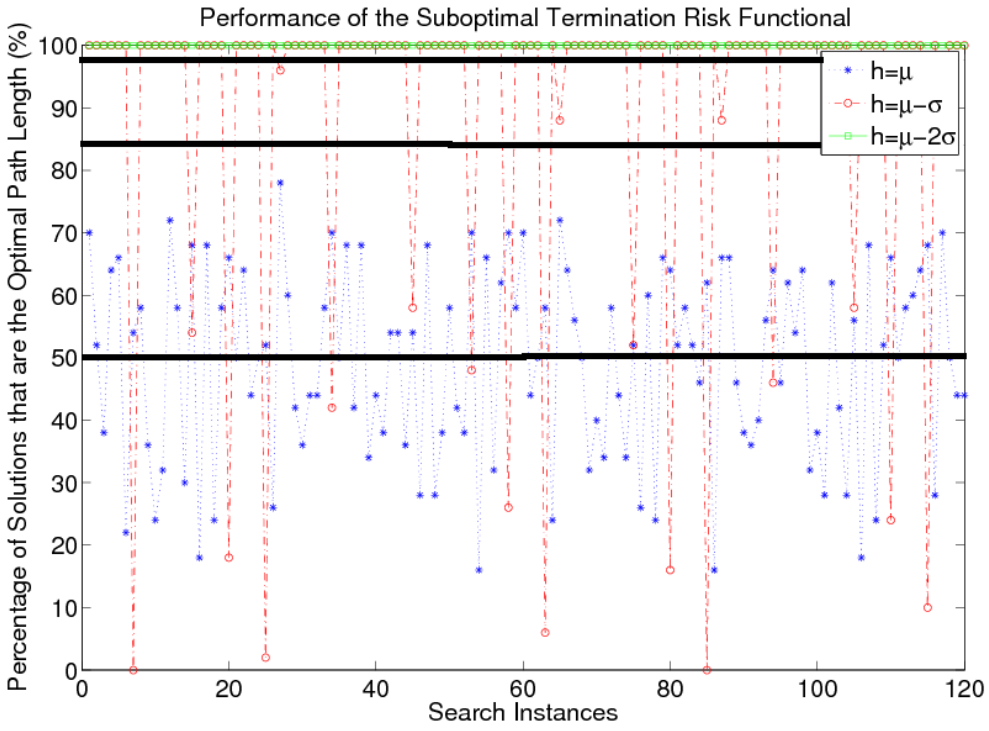


Figure 6.12: Scaling parameters obtained from the training data which was composed of queries used in Section 6.5.2.

ST0 heuristic. This is a somewhat counter-intuitive result - while both heuristics are inadmissible and therefore cannot guarantee a shortest, the values returned by the ST1 ($h = \mu_{\hat{h}} - \sigma_{\hat{h}}$) heuristic are always lower than that of ST0 ($h = \mu_{\hat{h}}$) for any given node which intuitively means that paths returned by ST1 should be shorter than those of ST0. As this has been shown empirically here to not be the case, we thought this phenomenon warranted further investigation.

6.5.4 Performance of the ST1 Heuristic

A salient observation can be made by comparing the performance of the ST1 risk functional in Figure 6.8(a) (used as the training set) with that of the hold-out set in Figure 6.13(a). While both perform approximately to expectation overall (achieving 81.2% and 90.28% against an expected number of 84.1% optimal solutions), notice that in Figure 6.8(a) the performance for all queries is clustered around the 84.1%



(a) Performance of R^*_δ with suboptimal termination, using constant scaling parameters.

	Suboptimal Termination Heuristic		
	$\boldsymbol{\mu}$	$\boldsymbol{\mu - \sigma}$	$\boldsymbol{\mu - 2\sigma}$
Expected % Optimal Solutions	50	84.1	97.7
Experimental % Optimal Solutions	49.11	90.28	100
Average Path Length Excess (%)	0.0067	0.0110	0
Average Normalized Search Effort (%)	26.14	98.69	98.76

Key:

Average Path Length Excess is measured as % in excess of the optimal path length

(b) Performance of R^*_δ search with varying levels of suboptimal risk over a hold-out set of 120 queries using scaling parameters that were the median result of those queries whose results are shown in Table 6.8(b). In comparison, using constant scaling parameters achieves the expected results for the ST0 ($h = \mu_{\hat{h}}$) heuristic, but performs too well on the ST1 and ST2 heuristics which suggests the c_μ scaling parameter estimate is about right, while c_σ is too high.

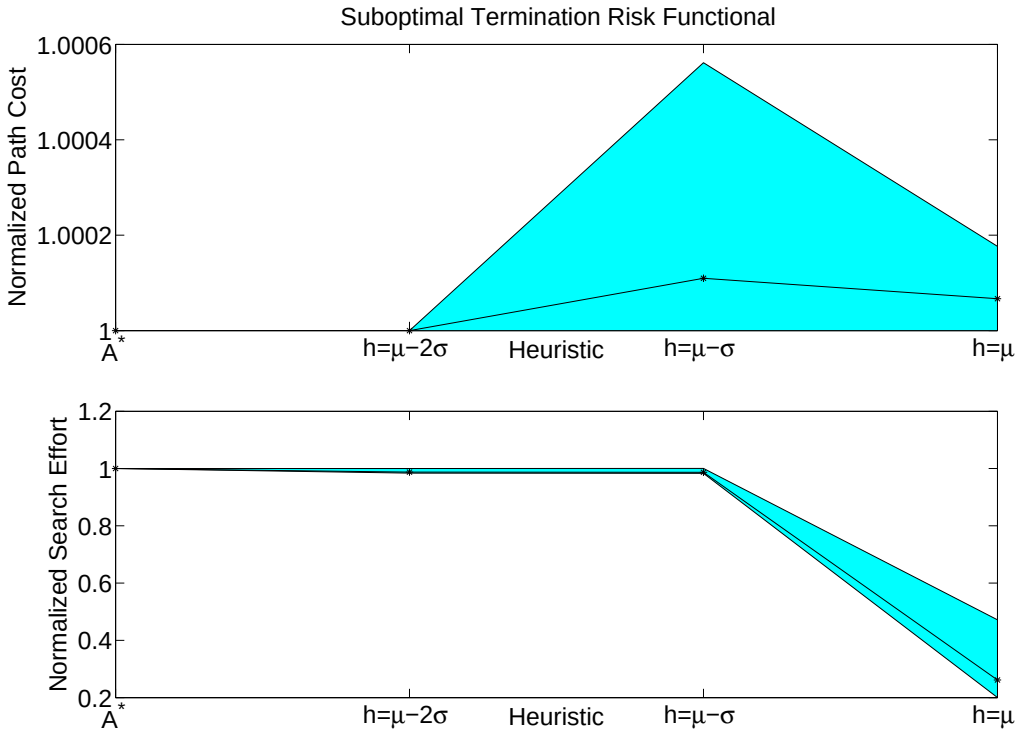
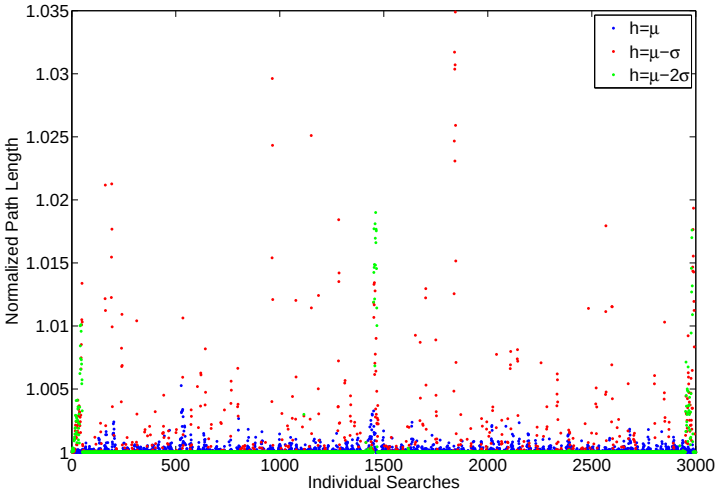


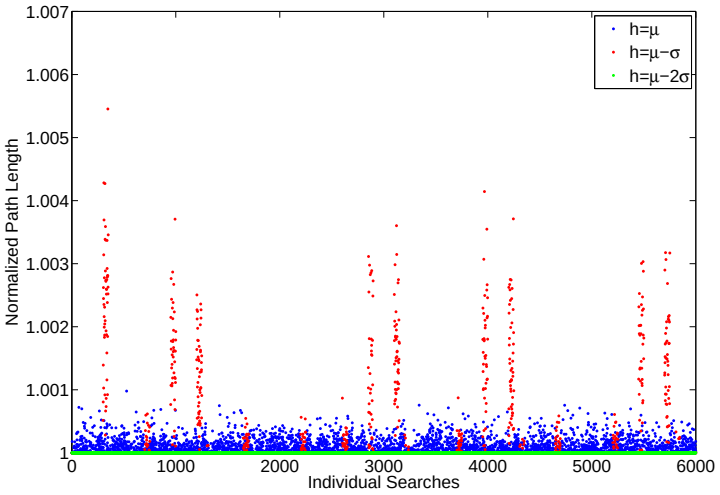
Figure 6.13: The results of running R_δ search with three different risk functional levels as per Equation (6.15), this time with constant scaling parameters c_μ and c_σ. The top plot is the normalized path cost with the solid line representing the mean of the normalized path cost and the filled section denoting the spread of values obtained within one standard deviation of the mean. The bottom plot is the normalized search effort in terms of nodes expanded. The solid line denotes the average search effort, the filled section denotes one standard deviation from the mean search effort.

mark while in 6.13(a) the spread of percentage optimal solutions is much greater amongst the 60 queries - some return close to 0% correct out of the 50 samples, many return 100% correct. This lead us to examine the normalized path lengths for all queries (3000 for the training set, 6000 for the hold-out set) for the two cases in Figure 6.14.

There are two interesting points to note from Figure 6.14. The first, evident in both Figures 6.14(a) and 6.14(b), is that when the ST1 heuristic gets the path wrong, it *really* gets it wrong. To illustrate this further, we plotted histograms of the normalized path lengths for both the ST0 and ST1 heuristics for both the training and hold-out sets in Figure 6.15. This figure shows that almost all path lengths for the ST0 case lie within 1.0009 times that of the optimal path length. By contrast,



(a) Constant Updating of Scaling Parameters



(b) Scaling Parameters Held Constant

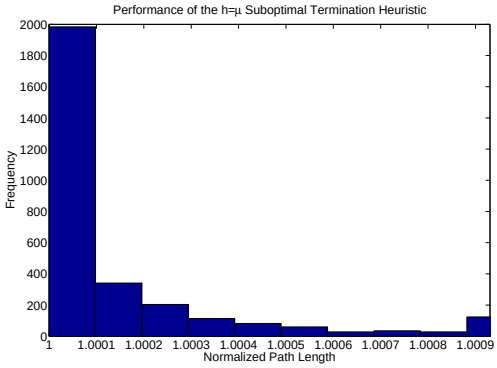
Figure 6.14: Normalized Path Lengths for all queries for both the training data set (a) and the hold-out data set (b). Note the 'streaky' appearance of the data for the $h = \mu - \sigma$ heuristic in (b) which denotes entire queries (corresponding clusters of 50 datapoints adjacent to each other on the x-axis) for which the R^*_δ algorithm with this heuristic performs badly. This 'streakiness' is not apparent in (a), where the parameters are constantly being tweaked, so that a poor performance at the start of 50 samples from a query can be remedied over the course of sampling from that query by adjusting the mean and variance scaling accordingly.

in the ST1 case there is a considerable portion of the paths beyond 1.001 times the optimal path length!

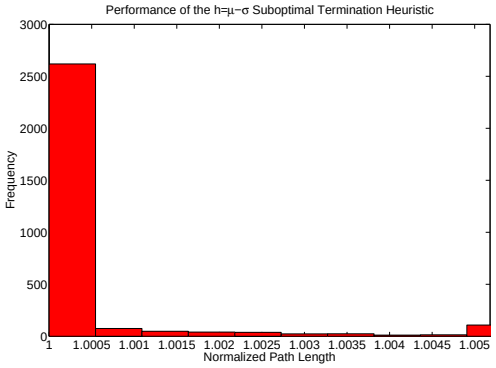
The second interesting point from Figure 6.14 is the comparative 'streakiness' of the data for the ST1 heuristic in (b) compared with (a). The 'streaky' appearance of the data in (b) denotes entire queries (corresponding clusters of 50 datapoints adjacent to each other on the x-axis) for which the R^*_δ algorithm with this heuristic performs badly. This 'streakiness' is not apparent in (a), where the parameters are constantly being tweaked, so that a poor performance at the start of 50 samples from a query can be remedied over the course of sampling from that query by adjusting the mean and variance scaling accordingly.

Taking these factors into account leads us to the conclusion that there are simply pathologically bad cases for which ST1 performs extremely badly. There is nothing wrong with our implementation - in these pathologically bad cases the ST1 heuristic *always* returns a smaller approximation to the underlying heuristic than the ST0 risk heuristic - intuitively this should lead to a shorter path! The correctness of our approach is further evidenced by the fact that the ST2 heuristic consistently returns 100% of queries equal to the optimal path for these pathological cases. In experiments using the Travelling Salesman Problem, Pearl and Kim (1982) does not report such an anomaly with the ST1 ($h = \mu - \sigma$) heuristic, which suggests this may be an interesting artifact of Probabilistic Costmaps and the ordering of the R^*_δ priority queue worthy of future investigation.

While the normalized path length of the ST1 estimator seems somewhat anomalous, its behaviour is not inconsistent with the stated aims of our risk-based search, which is that it should return the *optimal* solution 84.1% of the time. Both the training data and hold-out set results confirm this.

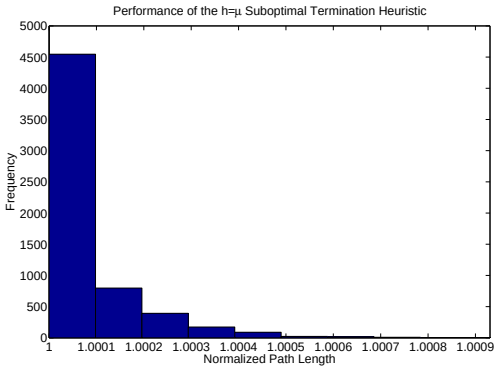


(a) ST0 ($h = \mu_{\hat{h}}$), Training Data

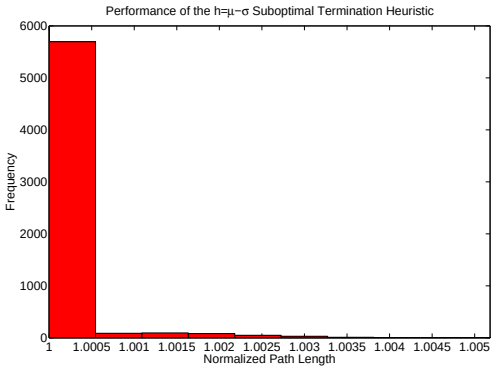


(b) ST1 ($h = \mu_{\hat{h}} - \sigma_{\hat{h}}$), Training Data

Histograms of the Normalized Path Lengths returned by the ST0 and ST1 heuristic estimators over the entire training data set.



(c) ST0, Hold-Out Data



(d) ST1, Hold-Out Data

Histograms of the Normalized Path Lengths returned by the ST0 and ST1 heuristics over the entire hold-out data set.

Figure 6.15: Normalized Path Lengths of the ST0 and ST1 heuristics. Note the comparative spread of the distributions between the two heuristics - ST1 ($h = \mu - \sigma$) returns more optimal solutions than ST0 ($h = \mu$) but its outliers are far further from 1.0 than is the case with suboptimal solutions returned by ST0. In comparing the performance in (a) and (b) (with constant tweaking of the parameters) with the constant scaling factor performance in (c) and (d), note the lack of outliers in the hold-out set results - yet this does not translate to any performance improvement in terms of optimal solutions or normalized path cost.

6.5.5 Ten Costmap Dataset

We further validated the operation of R^*_δ using our heuristic approximation technique on a set of 10 random terrain maps. The terrain maps are detailed in Appendix C. The costmaps are all 257×257 cell fractal terrain maps. We constructed a training data set of 50 separate queries to learn suitable parameters for c_μ and c_σ and then tested each costmap using a hold-out set of 50 new queries. The results are shown in Table 6.1.

This table confirms our conclusions drawn from the original, single costmap. Again, it is the ST0 ($h = \mu_{\hat{h}}$) suboptimal termination heuristic which is the standout performer. On average, it requires only 21.8% of the equivalent euclidean distance heuristic search effort. It is far more efficient than any other other heuristics we have considered here. The next best, the Expected Risk functional which allows up to 30% exceedence of optimal path length, achieves only a 6.7% increase in search efficiency over a standard A* search with euclidean distance heuristic. It is debatable, and may depend on the size of the graph considered, that the cost of precomputation makes the use of any heuristic other than the ST0 here worthwhile.

Note also that the ST1 and ST2 estimators both returned higher than expected levels of optimal solutions (95.35% versus 84.1% and 100% versus 97.7% respectively). When an inadmissible search returns a less-than-optimal solution it usually does so because it has terminated earlier than it would otherwise have if using an admissible heuristic. As these values are too high, we are missing the full efficiency gains of using inadmissible heuristics. What the values also show is that our estimate of the variance seems to be consistently too large, meaning that more than 84.1% of the true distribution lies above our estimated ST1 mark. This is evidenced in the table - out of the 10 costmaps only 2 report less than 84.1% average optimal solutions for the ST1 heuristic, and none report below 97.7% for the ST2 heuristic. In contrast, the median value obtained over the training data for the mean scaling parameter seems

to work well - 4 of the costmaps returned greater than the expected level of 50% optimal solutions and 6 reported levels below this for an average of 46.3% optimality. Obviously, the method of choosing the scaling parameters remains an area for further investigation.

6.6 Chapter Summary

Here we have presented an approach that combines the ALT algorithm and risk-based heuristics to guide path planning over probabilistic costmaps. The major contribution of this chapter is to make the heuristic of the A*/D* family of searches probabilistic - to take into account the fact that our knowledge of the terrain we are planning over is uncertain. Combined with the work in Chapter 5, we now have an end-to-end system for constructing probabilistic costmaps and planning probabilistically over them.

We began by describing the implementation of a precomputed heuristic by instituting a probabilistic variant of the A*, Landmark and Triangle (ALT) technique. We then implemented 2 families of risk heuristics - the Expected Risk functional bounds the length of the path over that of the optimal solution which we are prepared to accept. The Suboptimal Termination functional bounds the percentage of solutions we are prepared to accept that do not return the optimal path length, it has no bearing on the size of the error in those suboptimal paths it returns. These heuristics when used in conjunction with the A* search constitute an R^*_δ search (Pearl and Kim, 1982) (Pearl, 1984); an A* search variant wherein the heuristic may violate the admissibility condition, but where the risk of missing the opportunity for further cost reduction is bounded by some amount δ.

Of these, we found that the Suboptimal Termination risk functional ST0 ($h = \mu_{\hat{h}}$) works extremely well. It reduces the search effort of a standard A* search with euclidean heuristic by 75% and is guaranteed to return the optimal path length 50%

of the time. Empirically we showed that even when it does not return the optimal path, the solutions returned tend to lie within 0.02% of the optimal solution.

Future work could focus on learning a better approximation from the precomputed heuristic we learn through ALT to the unknown, underlying distance between nodes and the goal. At the moment we are using a linear scaling technique whose parameters are based on the median of those learned from a training set. While this appears to work well for predicting the mean of the underlying distribution, its performance in learning the variance is not as accurate. This affects the efficiency of the R^*_δ search as overestimating the variance leads the search to be overly conservative and examine more nodes.

Another area for future investigation is to examine the placement of the landmarks. In this work we used a planar landmark selection method which scatters landmarks around the boundary of the the map. While this works well in normal scalar grid maps, it would be worthwhile investigating the effect of landmark placement in the probabilistic domain. For example, landmark placement could be dictated by the variance in the probabilistic cost map, and it would be worthwhile verifying whether placing landmarks in areas of high or low uncertainty produces better results.

	Heuristic														
	ER 10		**ER 20**		**ER 30**		**ST0**			**ST1**			**ST2**		
	PL	**nSE**	**PL**	**nSE**	**PL**	**nSE**	**%Opt**	**PL**	**nSE**	**%Opt**	**PL**	**nSE**	**%Opt**	**PL**	**nSE**
Costmap 1	0.32	98.54	0.47	98.47	0.68	98.32	54.48	0.01	23.59	100.00	0.00	98.72	100.00	0.00	98.72
Costmap 2	0.00	98.81	0.01	98.78	0.02	98.74	52.32	0.01	21.37	100.00	0.00	98.88	100.00	0.00	98.88
Costmap 3	0.03	98.70	0.08	98.70	0.17	98.73	45.80	0.01	23.12	98.20	0.01	98.83	100.00	0.00	98.83
Costmap 4	0.36	98.64	0.47	98.65	0.64	98.72	55.52	0.01	26.98	100.00	0.00	98.58	100.00	0.00	98.58
Costmap 5	0.12	91.56	0.53	87.49	1.49	78.70	38.80	0.03	14.82	81.08	0.00	97.37	100.00	0.00	98.91
Costmap 6	0.00	98.68	0.00	98.67	0.01	98.64	45.60	0.02	21.61	100.00	0.00	98.69	100.00	0.00	98.69
Costmap 7	0.40	87.55	0.49	85.08	0.92	80.86	31.72	0.02	21.14	88.16	0.06	94.83	100.00	0.00	98.84
Costmap 8	0.16	98.49	0.21	98.37	0.32	98.12	60.48	0.01	23.81	100.00	0.00	98.76	100.00	0.00	98.79
Costmap 9	0.22	93.43	0.40	91.70	1.23	86.13	38.28	0.03	19.96	95.64	0.03	97.53	100.00	0.00	98.94
Costmap 10	0.91	98.24	1.30	97.65	2.53	96.29	49.32	0.02	24.64	71.80	0.34	98.65	100.00	0.00	98.75
Average	**0.18**	**96.07**	**0.29**	**95.12**	**0.56**	**93.24**	**46.32**	**0.02**	**21.80**	**95.35**	**0.01**	**97.99**	**100.00**	**0.00**	**98.78**

Key:

ER 10 - Expected Risk Heuristic allowing for 10% exceedence of optimal path length
ER 20 - Expected Risk Heuristic allowing for 20% exceedence of optimal path length
ER 30 - Expected Risk Heuristic allowing for 30% exceedence of optimal path length
ST0 - Suboptimal Termination with $h = \mu_{\hat{h}}$ heuristic
ST1 - Suboptimal Termination with $h = \mu_{\hat{h}} - \sigma_{\hat{h}}$ heuristic
ST2 - Suboptimal Termination with $h = \mu_{\hat{h}} - 2\sigma_{\hat{h}}$ heuristic
PL - % in excess of optimal Path Length
nSE - normalized Search Effort, % of equivalent search with euclidean heuristic
%Opt - percentage of solutions returned equal to shortest path length

Table 6.1: Performance over the 10 costmap dataset

Chapter 7

Conclusions

We conclude the thesis with a summary of the material covered in the preceding chapters, highlighting the main contributions of the thesis. Finally, we discuss directions for future work stemming from the results presented within the thesis.

7.1 Summary

The aim throughout this thesis has been to develop techniques to allow a robot to explore an unknown environment in a manner that is robust to deficiencies in its current understanding of the world. We have presented approaches to constructing an accurate representation of the world; to planning paths over it; and to deciding which region to explore next, all of which take into account the uncertainty in its current world view. In so doing, we have covered the three central tenets of autonomous robotic exploration - the capability to explore, the capabilitiy to plan paths, and the capability to navigate.

The introduction in Chapter 1 outlined the problem and discussed our motivation in addressing it. In Chapter 2, we reviewed previous approaches to tackling the problems of path planning and cost map creation. We positioned our own choice of planning algorithms for later development in the thesis - the A*/D* family of

algorithms - amongst the literature and discussed why they remain the dominant choice for field robotic path planning. We then discussed the means of creating the costmaps required by this family of path planners, before discussing in depth the operation of the A*, D* and Field D* algorithms.

In Chapter 3 we presented our own implementations of the D* and Field D* algorithms compatible with the MOOS operating system, done to enable these algorithms to run on the robots operated by the Mobile Robotics Group (MRG). The implementation comprised a library containing the algorithms, as well as support processes to process navigation and costmap data and generate motion commands. We also described our specific implementation of the Occupancy Grid Builder algorithm which together with our Costmap Builder forms the Costmap Library. Support processes were written to combine the output of laser scanners and stereo cameras into occupancy grids/costmaps subsequently published for consumption by the planner, and also to display the costmaps in a Graphical interface.

Chapter 4 presented an approach to enabling the robot to decide where to go next. We began by providing background to the exploration problem, positioning our choice of exploration algorithm, the Gap Navigation Tree, amongst the literature and providing arguments in support of its use. The Gap Navigation Tree is a lightweight topological exploration algorithm which explores by looking for 'gaps' (perceptual discontinuities commonly associated with corners and doorways in an indoor environment). In Tovar et al. (2003, 2004, 2007) the algorithm's original authors describe its reliance on an abstract *gap sensor*. We provided an implementation of a robust gap sensor using a laser scanner and metric mapping system which improves the real-world performance of the algorithm. Our approach was demonstrated using a robot simulator and simulated world which interfaced with our sensor software and the GNT algorithm. We demonstrated the capability of our gap sensor in detecting and tracking the gaps as they morphed with the robot's movements. However, the

definition of a gap is highly dependent on the structure of the environment that the robot is operating in. Failing to estimate that structure correctly can cause the gap sensor to fail to detect critical gap events crucial to maintaining the structure of the tree which drives the exploration process. Ultimately we concluded that the Gap Navigation Tree is too fragile a technique to be implemented on an operational robot.

In Chapter 5 we presented a method of creating probabilistic costmaps to facilitate planning with uncertainty. Our technique used overhead imagery to create costmaps which (a) take into explicit account uncertainty in terrain classification; and (b) allow the cost of a particular terrain (say scrub land) to vary with location in the map. Neither of these approaches are common in the literature, yet both reflect valid real-world considerations. The image is first classified using a multi-class Gaussian Process Classifier which provides probabilities of class membership at each location in the image. The probability of class membership at a particular grid location is then combined with a terrain cost evaluated at that location using a spatial Gaussian process. The resulting cost function is, in turn, passed to a planner. This allows both the uncertainty in terrain classification and spatial variations in terrain costs to be incorporated into the planned path. Because the cost of traversing a grid cell is now a probability density rather than a single scalar value, we can produce not only the *most-likely* shortest path between points on the map, but also sample from the cost map to produce a distribution of paths between the points. We showed results over 2 simulated and 3 real-world datasets, wherein paths were shown to vary in response to local variations in terrain cost; and valid distributions of the different possible paths between any two points in the map were obtained.

Chapter 6 built on the work of Chapter 5 to produce a planning algorithm capable of generating plans over uncertain maps quickly. Our approach combined the ALT (A* search, Landmarks and the Triangle inequality) algorithm and risk-based heuristics of the R^*_δ search to guide search over probabilistic cost maps. We used the probabilistic

cost map concept from Chapter 5 to precompute heuristics for searches such as A* and D* using landmarks as per the ALT technique. The resulting heuristics are probability distributions. We can speed up and direct search by characterizing the risk we are prepared to take in gaining search efficiency while sacrificing optimal path length, and modifying the heuristic accordingly. We showed results which evidence that our ALT-based technique must provide a good approximation to the true distribution of the (unknown) distance it estimates; and that demonstrate efficiency increases in excess of 70% over normal heuristic search methods.

7.2 Discussion and Conclusions

The bulk of the contribution of this work is in the incorporation of the treatment of uncertainty in the planning domain. We have shown how uncertainty measures can be incorporated into standard grid-based planning techniques through the construction of probabilistic cost maps. This method provides an alternative, faster means of planning probabilistically than the alternative Partially Observable Markov Decision Processes (POMDPs), predominant in the literature. Furthermore, the availability of a-priori data such as overhead imagery allows the precalculation of probabilistic heuristics, which have been shown to introduce efficiency savings of up to 72% over the standard euclidean distance heuristic.

At present this work, detailed in Chapters 5 and 6, sits somewhat at odds with the work in Chapter 4 which deals with exploration. In the approach to incorporating uncertainty into planning, we examined outdoor environments and used a grid-based representation of the world. In performing exploration using the Gap Navigation Tree, we superimposed a technique most suited to indoor environments and utilized a feature-based map on top of an existing metric navigation system. Whilst these two techniques are not incompatible, a limitation of the work presented here is that it

does not accomplish the consistent framework between planning and exploring under uncertainty that we set out to achieve.

A consistent framework would utilize the information contained in the probabilistic cost maps of Chapter 5 to drive exploration. O'Callaghan et al. (O'Callaghan et al., 2009) in their work on Occupancy Grid Maps produced using Gaussian Processes, point out that the variance plots produced as a by-product of the mapping process highlight unexplored regions and could potentially be used to optimize a robot's search plan. Likewise, the variance maps produced here as part of costmap construction in 5 could be used for this purpose, and a simple frontier-based approach to drive the robot to areas of greatest uncertainty would be both easy to implement and consistent with the framework.

A further limitation of the work in this thesis is that it has not been validated online on a robot. While this would not affect the contribution of the work in Chapter 6 (the a-priori data available to precompute heuristics would not change), it would bring interesting challenges to the construction and updating of the costmaps described in Chapter 5 as the a-priori data available from the likes of overhead imagery would need to be fused with the data acquired by the robot's sensors as the robot moves through the environment.

7.3 Future Work

Naturally, there are many aspects of this thesis where there is scope for improvement and extension. While some have been discussed in the preceding chapters, we summarize by topic some of the major areas for future investigation here.

7.3.1 Probabilistic Cost Maps

In Chapter 5 we identified the primary cause of our costmaps producing invalid paths as being the inconclusive and confused results from the multi-class Gaussian process classifier. A first step towards rectifying this problem would be to incorporate more inputs to the classification problem. At present, the classifier is only trained on colour data. Evidently from our results, this is not always sufficient to discriminate correctly between highly traversable terrain like roadways and dark coloured shrubbery or dark coloured roofs of houses! Infra-red imagery could be used to help discern between vegetation and man-made structures (Manduchi et al., 2005; Poppinga et al., 2008). Aerial LIDAR (Light Detection and Ranging) data could be used to provide height information to separate out roads from buildings with dark roofs (Carlberg et al., 2009; Guo et al., to be published; Lodha et al., 2007).

Secondly, in our examples we restricted ourselves to classifying amongst 3 distinct classes. The speed of the VBGP Matlab toolkit (Girolami and Rogers, 2006) we used was the limiting factor in this choice. A faster, alternative implementation of a probabilistic multi-class classifier needs to be explored in order to extend the number of terrain classes we can consider. Numerous techniques promise fast multi-class classification (Chen et al., 2009; Galleske and Castellanos, 2003; Lin et al., 2009; Wu et al., 2004) and alternative approaches are in need of investigation. Note however, that by allowing the cost of the terrain to vary within one terrain type, we have reduced the number of classes we need to be able to identify. While traditionally 'thick shrubbery' and 'sparse shrubbery' may need to be classed separately due to their different traversability properties, in our paradigm they can be represented with a single class. We envisage only having to extend to around 5-6 'super type' classes.

Finally, we see much to be gained from implementing the probabilistic costmap creation technique *online* on a robot. Obviously, running the algorithm online means the coarse initial classification and cost estimates garnered from the overhead image

can be updated, leading to a higher quality map. The robot's onboard sensors obviously produce different information from that available a-priori, so this data could be fused with the a-priori overhead information to produce more accurate terrain classification results. Feedback from the robot (the presence of certain features in onboard stereo imagery, its velocity, vibration etc) can be used to measure traversal costs (Hadsell et al., 2007; Konolige et al., 2009) and could be used to update the terrain map for each terrain type.

7.3.2 Risky Heuristics

In Chapter 6 we used the probabilistic cost maps to provide a precomputed heuristic estimate of the distance between any two grid cells in the map. In the ten costmaps we examined, our results showed that we could consistently and accurately estimate the mean of a probability distribution governing this distance, but consistently overestimated the variance in every case. This made the algorithm less efficient than is thought possible, and a more precise distribution is sought. Further investigation needs to be done into the relationship between the scaling parameters we employed to fit our estimated distribution to the real-world distribution in the training stage. It needs to be ascertained whether the linear scaling we employed in equation 6.8 is sufficient to estimate the underlying real-world distribution or whether an alternative approximation needs to be constructed.

Two heuristics were examined. One, the Suboptimal Termination Risk Functional, governed the risk the user was prepared to accept that the path computed would be longer than the optimal path length. It put bounds on the percentage of paths returned that would be longer than optimal. The second heuristic, the Expected Risk Functional, bounded the amount the paths returned could exceed the optimal path length. In future, it would be worthwhile investigating the combination of these two heuristics, so that guarantees such as 'the probability that the path obtained is

within 10% of the optimal path exceeds 90%' can be made.

We also reported a disappointing performance of the Expected Termination heuristic, which bounds the maximum fraction of optimal path length by which paths produced can exceed the length of the optimal path. This was unfortunate, as being able to stipulate a path length it is an appealing property to have for a path planner for the mobile robot domain - more so than the suboptimal termination heuristic for which we obtained good results; but which simply bounds the percentage of searches which return the optimal path and makes no guarantees as to the path lengths returned. Interestingly, in the paper in which Pearl introduces both risk functionals (Pearl and Kim, 1982) he only reports results using the suboptimal termination heuristic. Whether this omission means that the Expected Termination heuristic also failed to deliver in Pearl's chosen application of the heuristic search to the Travelling Salesman problem warrants further investigation. We postulate that the costmaps we experimented on do not exhibit enough variance to fully exploit the benefits of this heuristic and in future work would build alternative representations to test this theory.

In our precomputation of the heuristic we used a simple planar landmark selection method which scattered landmarks around the border of the map. This method had been shown to work well for scalar maps (Goldberg and Harrelson, 2005). In the probabilistic domain we have the added dimension of the variance and it would be worthwhile exploring the effect of placing landmarks in areas of high variance versus low variance on the quality of the heuristic produced. In Goldberg and Werneck (2009) the search for an optimal set of landmarks to serve queries from all over the map is posed as a k-median clustering problem (Bradley et al., 1997; Jain and Dubes, 1988), which places a 'cost' on the 'service' (quality of the heuristic) provided to each start-goal pair and repeatedly substitutes landmarks from a larger pool C into a smaller pool k until a subset of C is found that minimizes the total cost to all queries. This technique would reveal whether high or low variance landmarks provide better

solutions.

7.3.3 Exploration

In Chapter 4 we ultimately concluded that the GNT algorithm is not suited to use on a real robot. An exploratory capability is crucial to our aim of developing autonomous robots, and in future work we would like to investigate suitable drivers for exploration that build on our work in creating probabilistic costmaps. The Information Gain techniques of Section 4.2.2.1 seem a suitable starting point, given the format of the costmaps. Recent results obtained by Chli and Davison (2009), using the mutual information between candidate measurement locations and existing feature locations to direct the search for features in new images in performing visual tracking, may be applicable to our problem; we would ultimately like to examine the link between candidate goal locations and the distribution of paths produced to that goal determine where to go next.

7.4 Concluding Remark

A treatment of uncertainty in path planning and exploration is often absent from present-day field robotics systems. However, to achieve true robot autonomy we need to take into account this critical information which provides a measure of the robot's knowledge of the surrounding environment and hence heavily influences the quality of exploration, path planning and navigation decisions it makes. This thesis set out to provide a framework for robotic path planning and exploring under uncertainty. We hope that our approach will serve as a building block for future work in this area, and a step towards achieving the ultimate aim of robots that can explore autonomously.

Appendix A

Optimized D*

This appendix contains the pseudocode for the optimized version of the D* Lite algorithm Koenig and Likhachev (2002a) Koenig and Likhachev (2002b) Koenig and Likhachev (2002d). It was implemented as part of the software suite described in Chapter 3.

```
CalculateKey(s)
1  return [min(g(s),rhs(s))+h(s_start,s) +k_m; min(g(s),rhs(s))];

Initialize()
2  U = ∅
3  k_m = 0;
4  for all s ∈ S
         do rhs(s) = g(s) = ∞
5  rhs(s_goal) = 0;
6  U.Insert(s_goal,[h(s_start, s_goal); 0]);

UpdateVertex(u)
7  if (g(u) ≠ rhs(u) AND u ∈ U)U.Update(u, CalculateKey(u));
8      else if (g(u) ≠ rhs(u) AND u ∉ U) U.Insert(u, CalculateKey(u));
9      else if (g(u) = rhs(u)) AND u ∈ U)U.Remove(u);
```

Figure A.1: D* Lite - Optimized Version, continued overleaf

```
ComputeShortestPath()
10  while (U.TopKey() < CalculateKey(s_start) OR rhs(s_start) > g(s_start))
11        do u =U.Top();
12           k_old = U.TopKey();
13           k_new = CalculateKey(u);
14           if (k_old < k_new)
15              then U.Update(u, k_new);
16              else  if (g(u) > rhs(u))
17                       g(u) = rhs(u);
18                       U.Remove(u);
19                       for all s ∈ Pred(u)
20                            do if (s ≠ s_goal)
                                    then rhs(s) = min(rhs(s), c(s, u) + g(u));
21                               UpdateVertex(s);
22              else
23                       g_old = g(u);
24                       g(u) = ∞;
25                       for all s ∈ Pred(u) ∪ u
26                            do if (rhs(s) = c(s, u) + g_old)
27                               then if (s ≠ s_goal)
                                          then rhs(s) = min_{s'∈Succ(s)}(c(s, s') + g(s'));
28                               UpdateVertex(s);

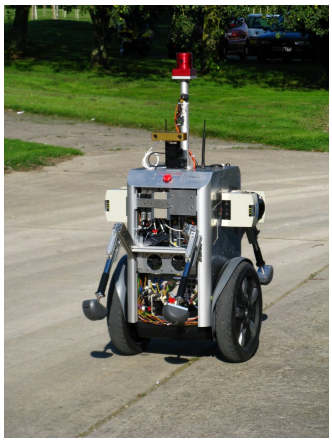
Main()
29  s_last = s_start;
30  Initialize();
31  ComputeShortestPath();
32  while (s_start ≠ s_goal)
          do ▷ if (rhs(s_start) = ∞) there is no known path
33           s_start = argmin_{s'∈Succ(s_start)}(c(s_start, s') + g(s'));
34           Move to s_start
35           Scan graph for changed edge costs;
36           if any edge costs changed
37              then k_m = k_m + h(s_last, s_start);
38                   s_last = s_start
39                   for all directed edges (u, v) with changed edge costs
40                        do c_old = c(u, v);
41                           Update the edge cost c(u, v);
42                           if (c_old > c(u, v));
43                              then if (u ≠ s_goal)
                                          then (rhs(u) = min(rhs(u), c(u, v) + g(v));
44                              else  if (rhs(u) = c_old + g(v))
45                                       if (u ≠ s_goal)
                                          then rhs(u) = min_{s'∈Succ(u)}(c(u, s') + g(s'));
46                           UpdateVertex(u);
47                   ComputeShortestPath();
```

Figure A.1: D* Lite - Optimized Version

Appendix B

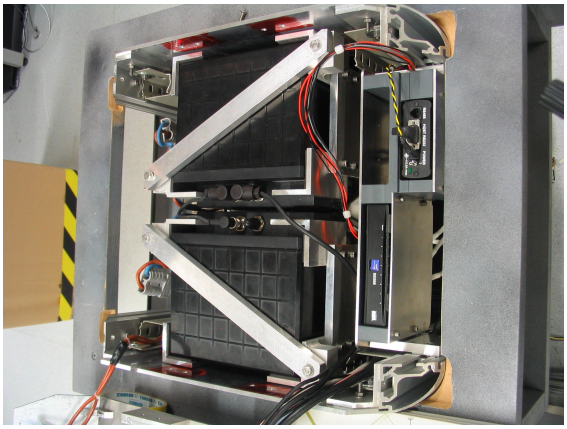
Hardware Systems

During the course of the thesis, MRG acquired a new robotic platform. Lisa is based on the Segway Robotic Mobility Platform and includes 2 SICK LMS 291 laser scanners, a Ladybug camera and a Bumblebee stereo camera.

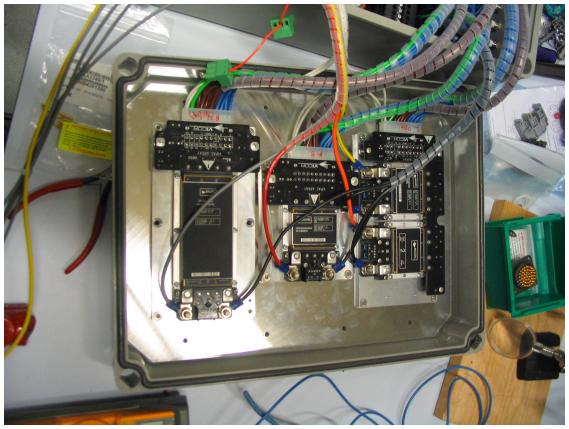


My involvement in the hardware design for Lisa's sensor payload was to design and procure batteries and dc-dc converters to power the various sensors. The weight restriction on Lisa's payload (approximately 150kgs) meant that special lightweight lithium ion batteries needed to be sourced. The power requirements of the various sensors needed to be calculated and the four operating voltages required by the different sensors (24V, 19V, 12V and 5V) needed to be accommodated.

I also designed the cut-out switch which acts as a remote-controlled emergency stop device for the robot. It was judged necessary to be able to be able to activate emergency stop from afar. The motion commands from the sensor platform to the RMP pass over CAN-bus, a two-wire interface. If this interface is broken for longer than 0.4 seconds then any motion commands controlling the velocity are immediately slewed to zero Segway (2007). The cut-out switch receives a Pulse Width Modulated signal from the servo output of the joystick remote control. This is fed through an integrator and then a comparator to determine whether the throttle on the joystick is ON or OFF. The CAN-bus is routed across a low-loss CMOS switch. If the joystick throttle is OFF, normal operation of the CAN bus continues. When the throttle is



(a) Lithium Batteries



(b) dc-dc converters

flicked to ON, the change in result of the integrator causes the control signal to be sent to the CMOS switch to throw the CAN bus open. After 0.4 seconds the robot will come to a halt.

Circuit schematics and PCB diagrams of the cut out switch are provided in Figure B.2 and Figure B.1 respectively.

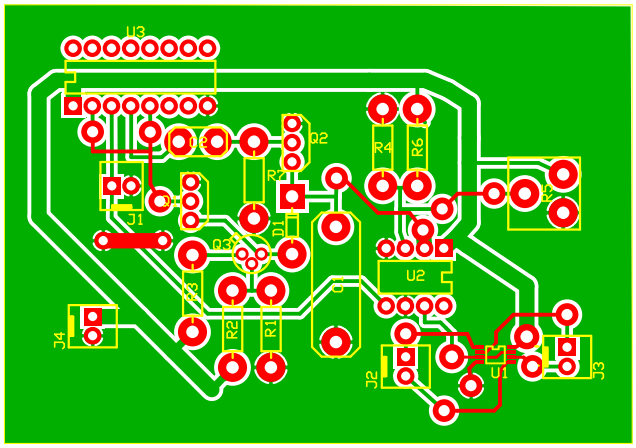


Figure B.1: Cut out switch PCB (actual size).

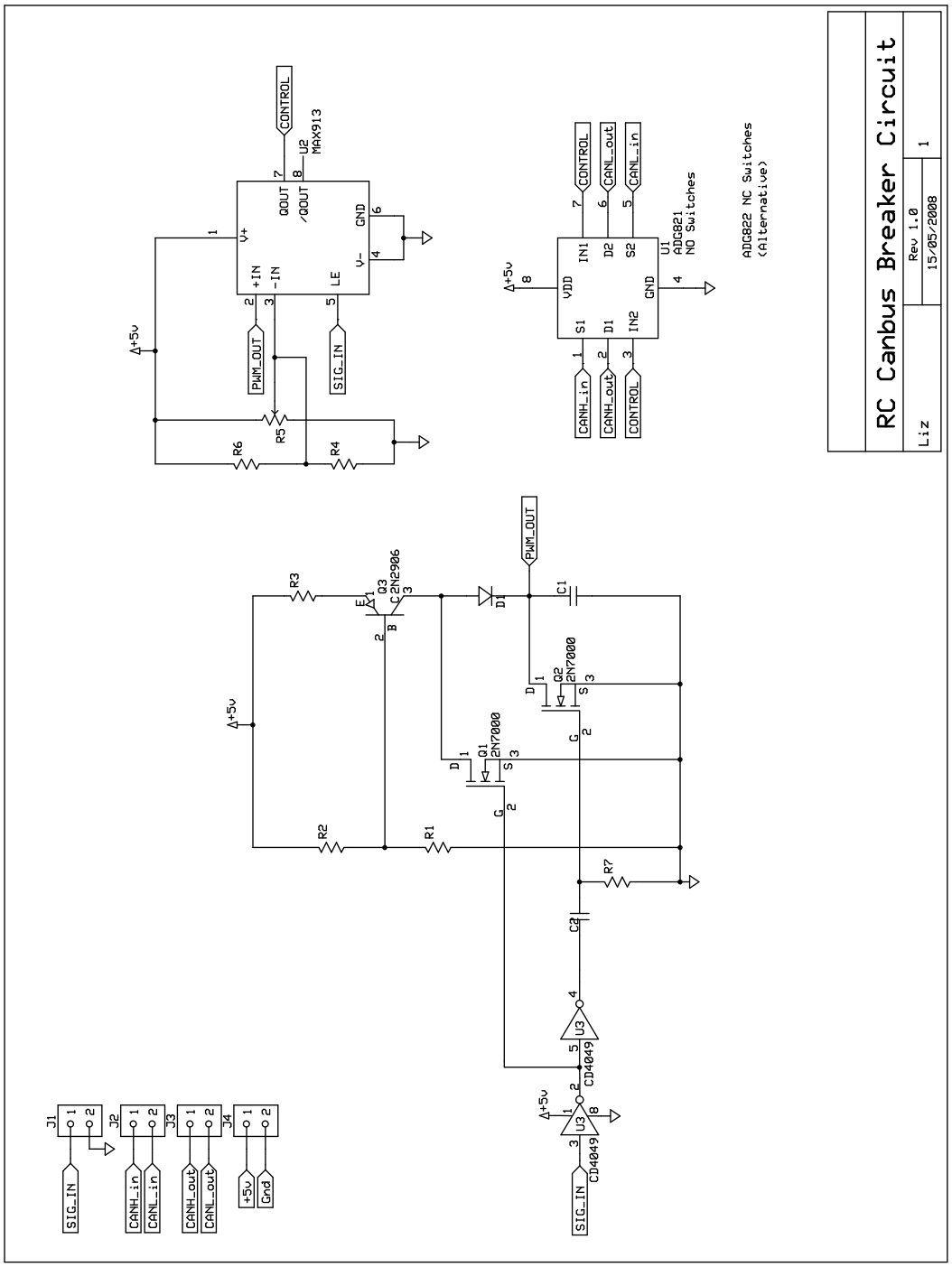
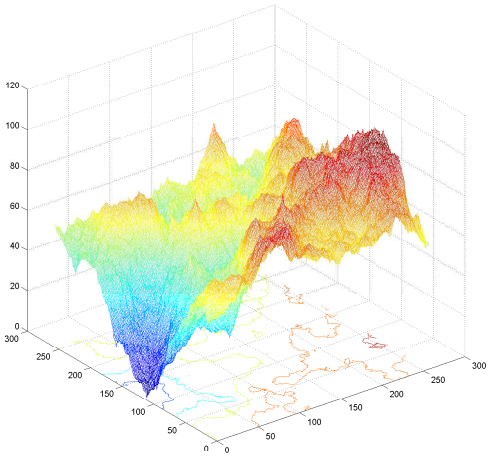


Figure B.2: Cut out switch schematic.

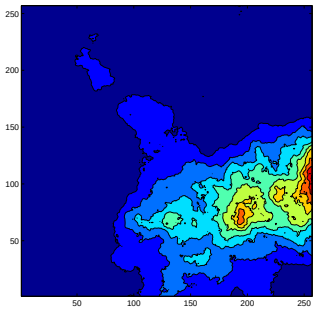
Appendix C

Random Costmaps

This Appendix contains the random costmaps used to generate the data in Chapter 6.5. The ten costmaps shown here are all 257×257 cell fractal terrain maps generated using the Diamond-square algorithm Fournier et al. (1982) Miller (1986) which is commonly used to generate realistic heightmaps for computer graphics applications.

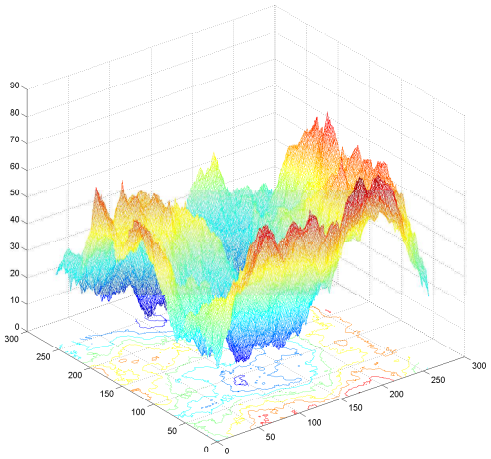


(a) Costmap 1 - Mean

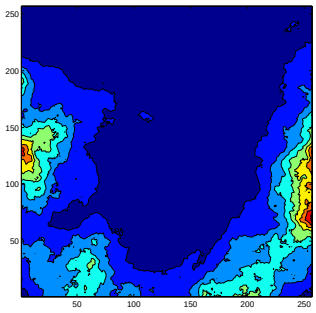


(b) Costmap 1 - Variance

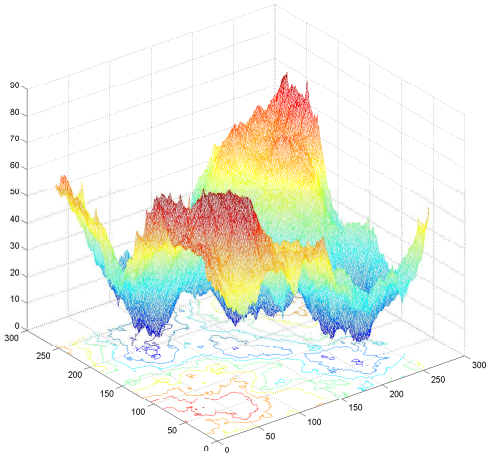
Figure C.1: Random Costmap Dataset, Costmap 1



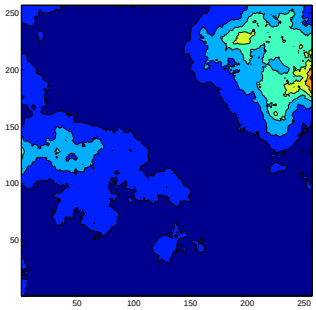
(c) Costmap 2 - Mean



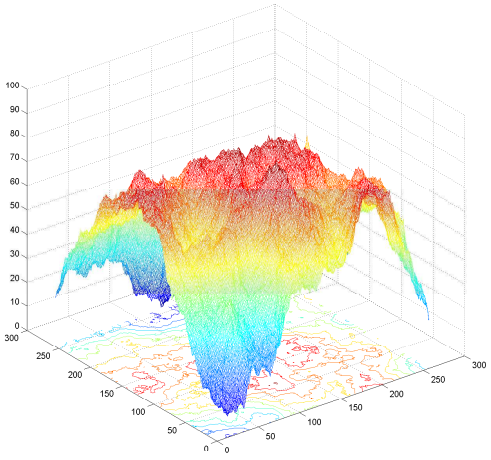
(d) Costmap 2 - Variance



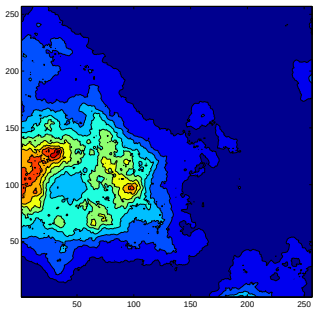
(e) Costmap 3 - Mean



(f) Costmap 3 - Variance

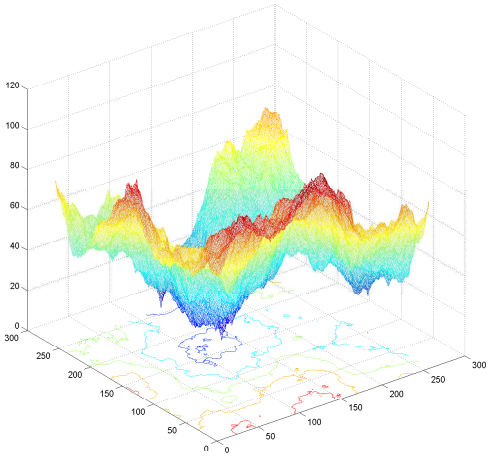


(g) Costmap 4 - Mean

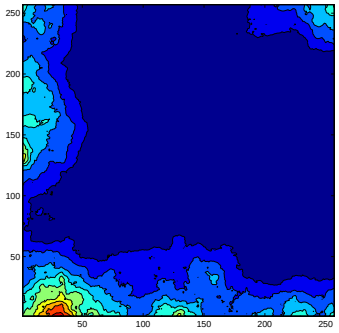


(h) Costmap 4 - Variance

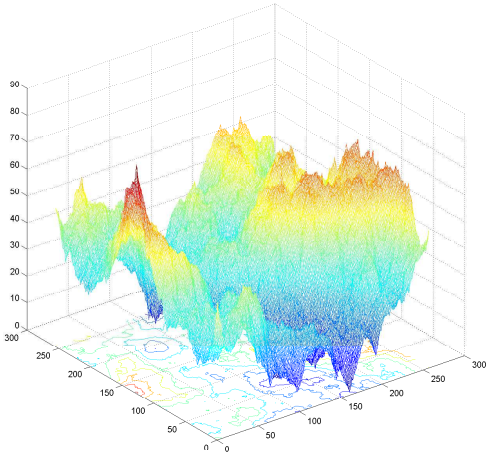
Figure C.1: Random Costmap Dataset, Costmaps 2 - 4



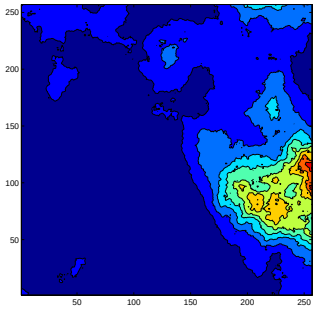
(i) Costmap 5 - Mean



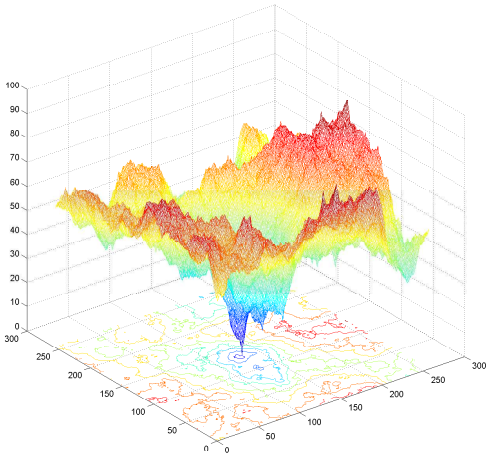
(j) Costmap 5 - Variance



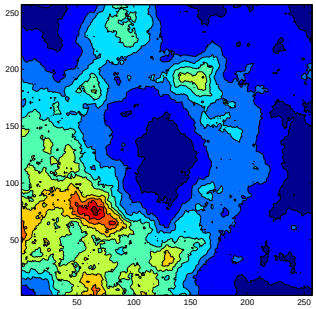
(k) Costmap 6 - Mean



(l) Costmap 6 - Variance

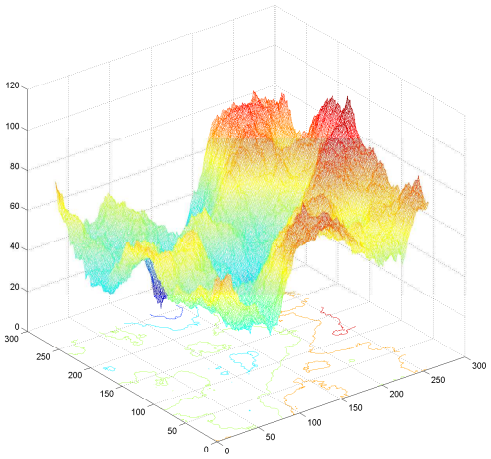


(m) Costmap 7 - Mean

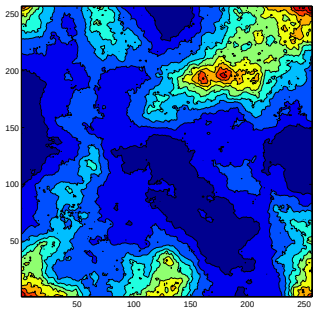


(n) Costmap 7 - Variance

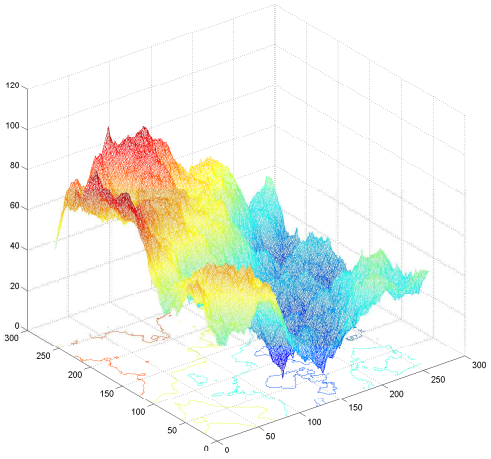
Figure C.1: Random Costmap Dataset, Costmaps 5 - 7



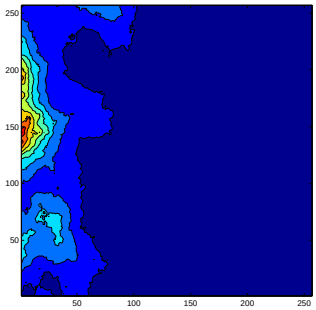
(o) Costmap 8 - Mean



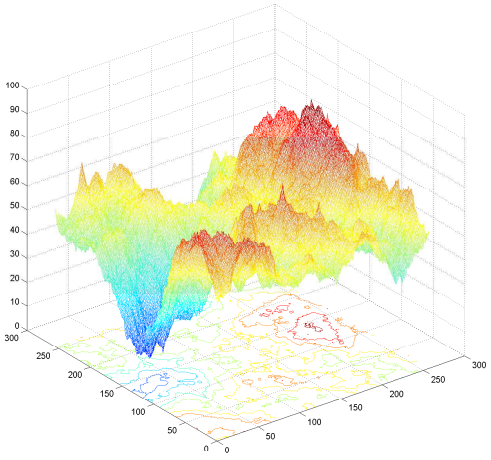
(p) Costmap 8 - Variance



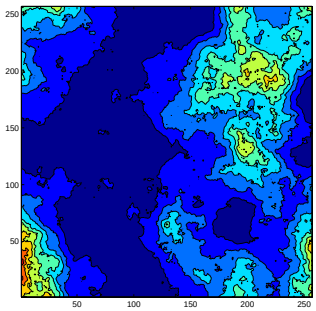
(q) Costmap 9 - Mean



(r) Costmap 9 - Variance



(s) Costmap 10 - Mean



(t) Costmap 10 - Variance

Figure C.1: Random Costmap Dataset, Costmaps 8 - 10

Bibliography

M. Aizerman, E. Braverman, and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.

R. C. Arkin and T. Balch. AuRA: Principles and practice in review. *Journal of Experimental and Theoretical Artificial Intelligence*, 9:175–189, 1997.

M. Bajracharya, A. Howard, L. H. Matthies, B. Tang, and M. Turmon. Autonomous off-road navigation with end-to-end learning for the LAGR program. *J. Field Robotics*, 26(1):3–25, 2009.

J. Bares, M. Hebert, T. Kanade, E. Krotkov, T. Mitchell, R. Simmons, and W. R. L. Whittaker. Ambler: An autonomous rover for planetary exploration. *IEEE Computer*, 22(1):18–26, June 1989.

J. Barraquand and J. Latombe. Robot motion planning: A distributed representation approach. *The International Journal of Robotics Research*, 10(6):628–649, 1991.

R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957. Republished 2003.

Boost. Boost C++ Libraries. `http://www.boost.org/`, 2010. Last accessed: 01/10/2010.

A. Botea, M. Muller, and J. Schaeffer. Near-optimal hierarchical pathfinding. *Journal of Game Development*, 1:7–28, 2004.

F. Bourgault, A. A. Makarenko, S. B. Williams, B. Grocholsky, and H. F. Durrant-Whyte. Information based adaptive robotic exploration. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 1, pages 540–545, Lausanne, Switzerland, September 2002.

D. Bradley, R. Unnikrishnan, and J. A. D. Bagnell. Vegetation detection for driving in complex environments. In *IEEE International Conference on Robotics and Automation*, pages 503–508, Rome, April 2007.

P. S. Bradley, O. L. Mangasarian, and W. N. Street. Clustering via concave minimization. In *Advances in Neural Information Processing Systems*, volume 9, pages 368–374. MIT Press, 1997.

R. A. Brooks. A robust layered control system for a mobile robot. *Robotics and Automation, IEEE Journal of*, 2(1):14–23, 1986.

B. Burns and O. Brock. Sampling-based motion planning with sensing uncertainty. In *IEEE Conference on Robotics and Automation*, pages 3313–3318, Rome, Italy,

April 2007.

J. Canny and M. Lin. An opportunistic global path planner. *Algorithmica*, 10:102–120, 1993.

M. Carlberg, P. Gao, G. Chen, and A. Zakhor. Classifying urban landscape in aerial LiDAR using 3D shape analysis. In *IEEE International Conference on Image Processing*, Cairo, Egypt, Feb. 2009.

J. Carsten, A. Rankin, D. Ferguson, and T. Stentz. Global planning on the mars exploration rovers: Software integration and surface testing. *Journal of Field Robotics*, 26:337–357, 2009.

B. Chazelle. Approximation and decomposition of shapes. In J. Schwartz and C. Yap, editors, *Advances in Robotics 1: Algorithmic and Geometric Aspects of Robotics*, pages 145–185. Lawrence Erlbaum Associates, 1987.

D. Chen, R. Szczerba, and J. Uhran. Planning conditional shortest paths through an unknown environment: a framed-quadtree approach. In *Intelligent Robots and Systems, IEEE/RSJ International Conference on*, volume 3, pages 3033–3038, Los Alamitos, CA, 1995.

J. Chen, C. Wang, and R. Wang. Adaptive binary tree for fast SVM multiclass classification. *Neurocomputing*, 72(13-15):3370 – 3375, 2009.

M. Chli and A. J. Davison. Active matching for visual tracking. *Robotics and Autonomous Systems*, 57(12):1173 – 1187, 2009.

C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, September 1995.

DARPA. DARPA Urban Challenge. `http://www.darpa.mil/grandchallenge/index.asp`, 2007. Last accessed: 01/10/2010.

R. Dechter and J. Pearl. Generalized best-first search strategies and the optimality of A*. *Journal of the Association for Computing Machinery (ACM)*, 32(3):505–536, July 1985.

E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematlk*, 1:269–271, 1959.

M. Dissanayake, P. Newman, S. Clark, H. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localization and map building (SLAM) problem. *Robotics and Automation, IEEE Transactions on*, 17(3):229 –241, jun. 2001.

D. Dolgov and S. Thrun. Detection of principal directions in unknown environments for autonomous navigation. In *Proceedings of the Robotics: Science and Systems IV (RSS-08)*, pages 73–80, Zurich, Switzerland, June 2008.

Y. Du, D. Hsu, H. Kurniawati, W. S. Lee, S. C. Ong, and S. W. Png. A POMDP approach to robot motion planning under uncertainty. In *Proceedings of the Int. Conf. on Automated Planning & Scheduling, Workshop on Solving Real-World POMDP Problems*, Toronto, May 2010.

A. Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation*, 3:249–265, June 1987.

A. Elfes. *Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation.* PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, 1989.

A. Ettlin and H. Bleuler. Randomised rough-terrain robot motion planning. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5798–5803, Beijing, China, October 2006a.

A. Ettlin and H. Bleuler. Rough-terrain robot motion planning based on obstacleness. In *Control, Automation, Robotics and Vision, 2006. ICARCV '06. 9th International Conference on*, pages 1–6, Singapore, December 2006b.

R. M. Eustice, H. Singh, and J. J. Leonard. Exactly sparse delayed-state filters for view-based SLAM. *Robotics and Automation, IEEE Transactions on*, 22(6): 1100–1114, Dec 2006.

H. Feder, J. Leonard, and C. Smith. Adaptive mobile robot navigation and mapping. *Int. Journal. Robotics Research*, 18(7):650–668, July 1999.

D. Ferguson and M. Likhachev. Efficiently using cost maps for planning complex maneuvers. In *In Proc. Int. Conf. on Robotics and Automation, Workshop on Planning with Cost Maps*, Pasadena, May 2008.

D. Ferguson and A. Stentz. The Delayed D* algorithm for efficient path replanning. In *Proc. IEEE International Conference on Robotics and Automation*, Barcelona, Spain, April 2005a.

D. Ferguson and A. T. Stentz. The Field D* algorithm for improved path planning and replanning in uniform and non-uniform cost environments. Technical Report CMU-RI-TR-05-19, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, July 2005b.

D. Ferguson and A. T. Stentz. Anytime RRTs. In *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '06)*, pages 5369 – 5375, Beijing, October 2006a.

D. Ferguson and A. T. Stentz. Using interpolation to improve path planning: The Field D* Algorithm. *Journal of Field Robotics*, 23(1):79–101, February 2006b.

D. I. Ferguson and A. Stentz. Multi-resolution Field D*. In *Proceedings of the International Conference on Intelligent Autonomous Systems (IAS)*, pages 65–74, Tokyo, March 2006c.

FLTK. Fast light toolkit. `http://www.fltk.org/`, 2010. Last Accessed: 01/10/2010.

D. A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach.* Prentice Hall, 2002. ISBN 0130851981.

A. Fournier, D. S. Fussell, and L. C. Carpenter. Computer rendering of stochastic models. *Commun. ACM*, 25(6):371–384, 1982.

Y. Freund and R. Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In P. Vitanyi, editor, *Computational Learning Theory*, volume 904 of *Lecture Notes in Computer Science*, pages 23–37. Springer Berlin / Heidelberg, 1995.

I. Galleske and J. Castellanos. A rotated kernel probabilistic neural network (RKPNN) for multi-class classification. In J. Mira and J. lvarez, editors, *Computational Methods in Neural Modeling*, volume 2686 of *Lecture Notes in Computer Science*, pages 1040–1040. Springer Berlin / Heidelberg, 2003.

E. Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Proceedings of the tenth national conference on Artificial intelligence*, AAAI'92, pages 809–815. AAAI Press, 1992.

B. Gerkey and M. Agrawal. Break on through: Tunnel-based exploration to learn about outdoor terrain. In *Proc. Int. Conf. on Robotics and Automation, Workshop on Planning with Cost Maps*, Pasadena, CA, May 2008.

B. P. Gerkey and K. Konolige. Planning and control in unstructured terrain. In *Proc. Int. Conf. on Robotics and Automation, Workshop on Planning with Cost Maps*, Pasadena, CA, May 2008.

M. Girolami and S. Rogers. Variational bayesian multinomial probit regression with gaussian process priors. *Neural Computation*, 18(8):1790–1817, 2006. Matlab toolkit available at: `http://www.dcs.gla.ac.uk/people/personal/girolami/pubs_2005/VBGP/`.

A. V. Goldberg. Point-to-point shortest path algorithms with preprocessing. In *SOFSEM '07: Proceedings of the 33rd conference on Current Trends in Theory and Practice of Computer Science*, pages 88–102, Harrachov, Czech Republic, Jan. 2007.

A. V. Goldberg and C. Harrelson. Computing the shortest path: A* search meets graph theory. In *Proceedings of the 16th annual ACM-SIAM symposium on Discrete algorithms*, pages 156–165, Vancouver, Jan. 2005.

A. V. Goldberg and R. F. Werneck. Selecting landmarks in shortest path computations. US Patent Application Publication, Publication No. US 2009/0228198, September 2009. [Online], Available at: `http://ip.com/patapp/US20090228198`.

V. Gordeski. Topological mapping for limited sensing mobile robots using the probabilistic gap navigation tree. M.S. Thesis, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge,MA, 2008.

Y. Goto and A. T. Stentz. Mobile robot navigation: The CMU system. *IEEE Expert*, 2(1):44 – 55, 1987.

K. Grant and D. Mould. LPI: Approximating shortest paths using landmarks. In *ECAI 2008 - Workshop on AI and Games*, Patras, Greece, July 2008.

L. Guo, N. Chehata, C. Mallet, and S. Boukir. Relevance of airborne LiDAR and multispectral image data for urban scene classification using random forests. *ISPRS Journal of Photogrammetry and Remote Sensing*, to be published.

R. Hadsell, P. Sermanet, J. Ben, A. Erkan, J. Han, U. Muller, and Y. LeCun. Online learning for offroad robots: Spatial label propagation to learn long-range traversability. In *Proceedings of Robotics: Science and Systems*, Atlanta, GA, USA, June 2007.

E. A. Hansen and R. Zhou. Anytime heuristic search. *J. Artif. Int. Res.*, 28(1):

267–297, 2007. ISSN 1076-9757.

L. R. Harris. The bandwidth heuristic search. In *IJCAI'73: Proceedings of the 3rd international joint conference on Artificial intelligence*, pages 23–29, San Francisco, CA, USA, 1973.

P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions of Systems Science and Cybernetics*, 4(2):100–107, July 1968.

N. Hawes, H. Zender, K. Sjöö, M. Brenner, G.-J. M. Kruijff, and P. Jensfelt. Planning and acting with an integrated sense of space. In *Proceedings of the 1st International Workshop on Hybrid Control of Autonomous Systems – Integrating Learning, Deliberation and Reactive Control (HYCAS)*, pages 25–32, Pasadena, CA, USA, July 2009.

K. Hsiao, L. P. Kaelbling, and T. Lozano-Perz. Grasping POMDPs. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 4685 – 4692, Rome, April 2007.

D. Hsu, J. Latombe, and H. Kurniawati. On the probabilistic foundations of probabilistic roadmap planning. *Int. J. Robotics Research*, 25(7):627–643, 2006.

A. K. Jain and R. C. Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

L. Kaelbling, M. Littman, and A. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

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

S. Koenig. Tutorial on greedy on-line planning. `http://idm-lab.org/slides/greedyonline-tutorial-sven.pdf`, January 2002. Last accessed 1/10/2010.

S. Koenig and M. Likhachev. D* Lite. In *Proceedings of the AAAI Conference of Artificial Intelligence (AAAI)*, pages 476–483, Menlo Park, CA, USA, 2002a.

S. Koenig and M. Likhachev. Improved fast replanning for robot navigation in unknown terrain. In *Robotics and Automation, Proceedings. IEEE International Conference on*, volume 1, pages 968 – 975, Washington, DC, May 2002b.

S. Koenig and M. Likhachev. Incremental A*. In *In Advances in Neural Information Processing Systems (NIPS)*, pages 1539–1546, Vancouver, BC, Dec 2002c. MIT Press.

S. Koenig and M. Likhachev. Improved fast replanning for robot navigation in unknown terrain. Technical Report GIT-COGSCI-2002/3, College of Computing, Georgia Institute of Technology, Atlanta, GA, 2002d.

S. Koenig, C. Tovey, and W. Halliburton. Greedy mapping of terrain. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3594–3599, Seoul, May 2001.

S. Koenig, M. Likhachev, and D. Furcy. Lifelong planning A*. *Artificial Intelligence*,

155(1-2):93–146, May 2004.

P. Komma, C. Weiss, and A. Zell. Adaptive bayesian filtering for vibration-based terrain classification. In *IEEE/RSJ International Conference on Robotics and Automation (ICRA 2009)*, pages 3307–3313, Kobe, Japan, May 2009.

K. Konolige. Improved occupancy grids for map building. *Autonomous Robots*, 4(4): 351–367, October 1997.

K. Konolige, K. Myers, E. Ruspini, and A. Saffiotti. The saphira architecture: A design for autonomy. *Journal of Experimental and Theoretical Artificial Intelligence*, 9:215–235, 1997.

K. Konolige, M. Agrawal, R. C. Bolles, C. Cowan, M. Fischler, and B. Gerkey. Outdoor mapping and navigation using stereo vision. In *Proceedings of the International Symposium on Experimental Robotics*, pages 179–190, Rio de Janeiro, Brazil, July 2006.

K. Konolige, M. Agrawal, M. R. Blas, R. C. Bolles, B. Gerkey, J. Sola, and A. Sundaresan. Mapping, navigation, and learning for off-road traversal. *J. Field Robot.*, 26:88–113, January 2009.

J. Kuffner and S. LaValle. RRT-Connect: An efficient approach to single-query path planning. In *In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 995–1001, San Francisco, CA, Apr 2000.

B. Kuipers and Y.-T. Byun. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Robotics and Autonomous Systems*, 8(1-2):47–63, 1991.

D. Kurniawati, H.; Hsu and W. Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *In Proc. Robotics: Science and Systems.*, Zurich, Jun. 2008.

Y. Landa, R. Tsai, and L.-T. Cheng. Visibility of point clouds and mapping of unknown environments. In J. Blanc-Talon, W. Philips, D. Popescu, and P. Scheunders, editors, *ACIVS*, volume 4179 of *Lecture Notes in Computer Science*, pages 1014–1025. Springer, 2006.

Y. Landa, D. Galkowski, Y. R. Huang, A. Joshi, C. Lee, K. K. Leung, G. Malla, J. Treanor, V. Voroninski, A. L. Bertozzi, and Y.-H. R. Tsai. Robotic path planning and visibility with limited sensor data. In *Proceedings of the 2007 American Control Conference*, pages 5425–5430, New York, NY, Jul 2007.

J.-C. Latombe. *Robot Motion Planning*. Kluwer, Boston, 1991.

S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report TR 98-11, Computer Science Dept., Iowa State University, October 1998.

S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006.

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

N. D. Lawrence, M. Seeger, and R. Herbrich. The informative vector machine: a practical probabilistic alternative to the support vector machine. Technical Report CS-04-07, Department of Computer Science, University of Sheffield, Sheffield, UK, 2004.

M. Likhachev, G. Gordon, and S. Thrun. ARA*: Anytime A* search with provable bounds on sub-optimality. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Proceedings of Conference on Neural Information Processing Systems (NIPS)*. MIT Press, 2003.

M. Likhachev, D. Ferguson, G. Gordon, A. T. Stentz, and S. Thrun. Anytime Dynamic A*: An anytime, replanning algorithm. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 262–271, Monterey, CA, June 2005.

H. Lin, K. Crammer, and J. Bilmes. How to lose confidence: Probabilistic linear machines for multiclass classification. In *Proc. Annual Conference of the International Speech Communication Association (INTERSPEECH)*, Brighton, UK, September 2009.

S. K. Lodha, D. M. Fitzpatrick, and D. P. Helmbold. Aerial LiDAR data classification using AdaBoost. In *3D Digital Imaging and Modeling, International Conference on*, pages 435–442, Los Alamitos, CA, USA, 2007.

T. Lozano-Perez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, 32(2):108–120, 1983.

F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Auton. Robots*, 4(4):333–349, 1997.

V. J. Lumelsky and A. A. Stepanov. Dynamic path planning for a mobile automaton with limited information on the environment. *IEEE Transactions on Automatic Control*, 31(11):1058–1063, November 1986.

A. Makarenko, S. Williams, F. Bourgault, and H. Durrant-Whyte. An experiment in integrated exploration. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Lausanne, Switzerland, 2002.

R. Manduchi, A. Castano, A. Talukder, and L. Matthies. Obstacle detection and terrain classification for autonomous off-road navigation. *Autonomous Robots*, 18: 81–102, 2005. ISSN 0929-5593.

M. Mataric and R. Brooks. Learning a distributed map representation based on navigation behaviours. In *Proceedings of 1990 USA-Japan Symposium on Flexible Automation*, pages 499–506, Kyoto, Japan, 1990.

E. Mazer, J. M. Ahuactzin, and P. Bessiere. The Ariadne's Clew Algorithm. *Journal of Artificial Intelligence Research*, 9:295–316, 1998.

K. Mehlhorn and S. Naher. LEDA - library of efficient data types and algorithms. `http://www.mpi-inf.mpg.de/~mehlhorn/LEDAbook.html`, July 2010.

G. S. P. Miller. The definition and rendering of terrain maps. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 39–48, New York, NY, USA, 1986.

G. A. Mills-Tettey, A. T. Stentz, and M. B. Dias. DD* Lite: Efficient incremental search with state dominance. Technical Report CMU-RI-TR-07-12, Robotics Institute, Pittsburgh, PA, May 2007.

P. E. Missiuro and N. Roy. Adapting probabilistic roadmaps to handle uncertain maps. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 1261 – 1267, Orlando,FL, May 2006.

H. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *Proceedings of the 1985 IEEE International Conference on Robotics and Automation*, pages 116–121, Washington, D.C., 1985.

H. P. Moravec. Sensing versus inferring in robot control. Technical report, Robotics Institute, Carnegie-Mellon University, Pittsburgh, PA, Jan. 1987.

H. P. Moravec. Robot spatial perception by stereoscopic vision and 3D evidence grids. Technical Report CMU-RI-TR-96-34, Robotics Institute, Carnegie-Mellon University, Pittsburgh,PA, September 1996.

W. Mou and A. Kleiner. Online learning terrain classification for adaptive velocity control. In *Proceedings of the IEEE Int. Workshop on Safety, Security and Rescue Robotics*, Bremen, July 2010.

D. Mould and M. C. Horsch. An hierarchical terrain representation for approximately shortest paths. In *8th Pacific Rim International Conference on Artificial Intelligence*, pages 104–113, Auckland, New Zealand, August 2004.

E. Murphy and P. Newman. Using incomplete online metric maps for topological exploration with the gap navigation tree. In *Proc. IEEE International Conference on Robotics and Automation (ICRA'08)*, pages 2792–2797, Pasadena,California, April 2008.

E. Murphy and P. Newman. Planning most-likely paths from overhead imagery. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3059–3064, Anchorage, AK, May 2010.

E. Murphy and P. Newman. Risky planning: Path planning over costmaps with a probabilistically bounded speed-accuracy tradeoff. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Shanghai, China, May 2011. To Appear.

R. Murphy. Dempster-shafer theory for sensor fusion in autonomous mobile robots. *Robotics and Automation, IEEE Transactions on*, 14(2):197 –206, Apr. 1998.

A. Nash, K. Daniel, S. Koenig, and A. Felner. Theta*: Any-angle path planning on grids. In *In Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 2, pages 1177–1183, Vancouver, BC, Jul. 2007.

A. Nash, S. Koenig, and M. Likhachev. Incremental Phi*: Incremental any-angle path planning on grids. In *In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1824–1830, Pasadena, CA, Jul. 2009.

P. Newman. The MOOS - cross platform software for robotics research. `http://www.robots.ox.ac.uk/~mobile/MOOS`, July 2010. Last accessed: 01/10/2010.

P. Newman, M. Bosse, and J. Leonard. Autonomous feature-based exploration. In

Proc. IEEE International Conference on Robotics and Automation, volume 1, pages 1234–1240, Taiwan, Sep. 2003.

P. Newman, G. Sibley, M. Smith, M. Cummins, A. Harrison, C. Mei, I. Posner, R. Shade, D. Schroeter, L. Murphy, W. Churchill, D. Cole, and I. Reid. Navigating, recognising and describing urban spaces with vision and laser. *The International Journal of Robotics Research*, 28(11-12):1406–1433, October 2009.

N. J. Nilsson. A mobile automaton: An application of artificial intelligence techniques. In *Proc. 1st International Conference on Artificial Intelligence*, pages 509–520, 1969.

N. J. Nilsson. Shakey the robot. Technical Report 323, AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, Apr 1984.

I. R. Nourbakhsh and M. R. Genesereth. Assumptive planning and execution: A simple, working robot architecture. *Autonomous Robots*, 3(1):49–67, March 1996.

S. O'Callaghan, F. T. Ramos, and H. Durrant-Whyte. Contextual occupancy maps using gaussian processes. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, Kobe, Japan, May 2009.

C. O'Dunlaing and C. K. Yap. A retraction method for planning the motion of a disc. *Journal of Algorithms*, 6:104–111, 1982.

J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1984.

J. Pearl and J. H. Kim. Studies in semi-admissible heuristics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 4(4):392–399, July 1982.

J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1025 – 1032, Acapulco, Mexico, Aug. 2003.

J. Pineau, G. Gordon, and S. Thrun. Anytime point-based approximations for large POMDPs. *Journal of Artificial Intelligence Research (JAIR)*, 27(1):335–380, 2006.

J. C. Platt. Probabilistic outputs for support vector machines and comparison to regularize likelihood methods. In A. Smola, P. Bartlett, B. Schoelkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 61–74, Cambridge, MA, 2000. MIT Press.

I. Pohl. Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, 1 (3-4):193 – 204, 1970.

J. Poppinga, A. Birk, and K. Pathak. Hough based terrain classification for realtime detection of drivable ground: Research articles. *J. Field Robot.*, 25(1-2):67–88, 2008.

I. Posner, D. Schroeter, and P. Newman. Describing composite urban workspaces. In *Proc. of the Int. Conference on Robotics and Automation*, pages 4962 – 4968, Rome, Apr 2007.

I. Posner, M. Cummins, and P. Newman. Fast probabilistic labeling of city maps. In *Proceedings of Robotics: Science and Systems IV*, pages 17–24, Zurich, Switzerland,

June 2008.

C. E. Rasmussen. *Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2 - 14, 2003, Tubingen, Germany, August 4 - 16, 2003, Revised Lectures*, volume 3176 of *Lecture Notes in Computer Science*, chapter Gaussian Processes in Machine Learning, pages 63–71. Springer-Verlag, Heidelberg, 2004.

C. E. Rasmussen. Advances in gaussian processes, Tutorial at neural information processing systems conf. (NIPS). `http://learning.eng.cam.ac.uk/carl/talks/gpnt06.pdf`, December 2006.

C. E. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

S. Russell and P. Norvig. *Artificial Intelligence - A Modern Approach*. Prentice-Hall Series In Artificial Intelligence, 3rd edition, 2010.

H. Samet. Neighbor finding techniques for images represented by quadtrees. *Comput. Graphics Image Process*, 18:37–57, 1982.

Segway. *Segway Robotic Mobility Platform (RMP) Interface Guide*. Segway LLC, 14 Technology Drive Bedford NH 03110, 2.0 edition, 2007. Available at HTTP: http://rmp.segway.com/.

D. Silver, B. Sofman, N. Vandapel, J. Bagnell, and A. Stentz. Experimental analysis of overhead data processing to support long range navigation. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 2443–2450, Beijing, Oct. 2006.

D. Silver, J. A. D. Bagnell, and A. T. Stentz. High performance outdoor navigation from overhead data using imitation learning. In *Robotics: Science and Systems IV*, pages 262–269, Zurich, June 2008.

R. Sim and N. Roy. Global a-optimal robot exploration in SLAM. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 661–666, Barcelona, Spain, Apr. 2005.

R. Smallwood and E. Sondik. The optimal control of Partially Observable Markov Decision Processes over a finite horizon. *Operations Research*, 21:1071–1088, 1973.

R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. In *Robotics and Automation. Proceedings. 1987 IEEE International Conference on*, volume 4, pages 850–850, Raleigh, NC, Mar. 1987.

T. Smith and R. Simmons. Point-based POMDP algorithms: Improved analysis and implementation. In *Proc. of the Conference on Uncertainty in Artificial Intelligence*, pages 542–549, Arlington, VA, July 2005.

B. Sofman. *Online Learning Techniques for Improving Robot Navigation in Unfamiliar Domains*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, April 2009.

B. Sofman, J. A. D. Bagnell, A. T. Stentz, and N. Vandapel. Terrain classification from aerial data to support ground vehicle navigation. Technical Report CMU-RI-TR-05-39, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, January

2006a.

B. Sofman, E. Lin, J. A. Bagnell, J. Cole, N. Vandapel, and A. Stentz. Improving robot navigation through self-supervised online learning. *Journal of Field Robotics*, 23:1059–1075, 2006b.

C. Stachniss, G. Grisetti, and W. Burgard. Information gain-based exploration using Rao-Blackwellized particle filters. In *Proceedings of Robotics: Science and Systems*, pages 65–72, Cambridge, MA, June 2005.

A. Stentz. Optimal and efficient path planning for partially-known environments. In *Proceedings of the International Conference on Robotics and Automation*, volume 4, pages 3310–3317, San Diego, CA, May 1994a.

A. Stentz. The D* algorithm for real-time planning of optimal traverses. Technical Report CMU-RI-TR-94-37, Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, September 1994b.

A. Stentz. The focussed D* algorithm for real-time replanning. In *In Proceedings of the International Joint Conference on Artificial Intelligence*, volume 2, pages 1652–1659, Montreal, Canada, August 1995.

A. T. Stentz. Optimal and efficient path planning for unknown and dynamic environments. Technical Report CMU-RI-TR-93-20, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 1993.

S. Thrun. Robotic mapping: A survey. In G. Lakemeyer and B. Nebel, editors, *Exploring Artificial Intelligence in the New Millenium*, The Morgan Kaufmann Series in Artificial Intelligence, chapter 1, pages 1–35. Morgan Kaufmann, 2002.

S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, Cambridge, MA, 2005.

S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney. Stanley: The robot that won the DARPA grand challenge. *Journal of Field Robotics*, 23(1):661–692, June 2006.

B. Tovar, S. La Valle, and R. Murrieta. Optimal navigation and object finding without geometric maps or localization. In *Proceedings IEEE International Conference on Robotics and Automation*, volume 1, pages 464 – 470, Taipei, Taiwan, Sep. 2003.

B. Tovar, L. Guilamo, and S. M. LaValle. Gap navigation trees: Minimal representation for visibility-based tasks. In *Proc. Workshop on the Algorithmic Foundations of Robotics (WAFR)*, pages 11–26, Utrecht, July 2004.

B. Tovar, R. Murrieta-Cid, and S. LaValle. Distance-optimal navigation in an unknown environment without sensing distances. *Robotics, IEEE Transactions on*, 23(3):506 –518, Jun. 2007.

C. Urmson and R. Simmons. Approaches for heuristically biasing RRT growth. In *Intelligent Robots and Systems, Proceedings. 2003 IEEE/RSJ International Conference on*, volume 2, pages 1178 – 1183, Las Vegas, NV, Oct. 2003.

N. Vandapel, R. R. Donamukkala, and M. Hebert. Quality assessment of traversability maps from aerial LiDAR data for an unmanned ground vehicle. In *International Conference on Intelligent Robots and Systems (IROS)*, volume 1, pages 305 – 310, Las Vegas, NV, Oct. 2003.

N. Vandapel, D. Huber, A. Kapuria, and M. Hebert. Natural terrain classification using 3-D LiDAR data. In *Proc. IEEE International Conference on Robotics and Automation*, volume 5, pages 5117 – 5122, New Orleans, LA, April 2004.

P. Vernaza, B. Taskar, and D. D. Lee. Online, self-supervised terrain classification via discriminatively trained submodular markov random fields. In *International Conference on Robotics and Automation (ICRA)*, pages 2750–2757, Pasadena, CA, May 2008.

C. Weiss, H. Frohlich, and A. Zell. Vibration-based terrain classification using support vector machines. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4429–4434, Beijing, Oct. 2006.

G. Welch and G. Bishop. An introduction to the kalman filter. Technical Report TR95-041, Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC, July 2006.

D. T. Wooden. *Graph-based Path Planning for Mobile Robots*. PhD thesis, School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, Nov. 2006.

T.-F. Wu, C.-J. Lin, and R. C. Weng. Probability estimates for multi-class classification by pairwise coupling. *J. Mach. Learn. Res.*, 5:975–1005, Dec. 2004.

B. Yamauchi. A frontier-based approach for autonomous exploration. In *CIRA '97: Proceedings of the 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pages 146–151, Monterey, CA, July 1997. ISBN 0-8186-8138-1.

A. Zelinsky. A mobile robot navigation exploration algorithm. *IEEE Transactions on Robotics and Automation*, 8(6):707–717, December 1992.