The Dissertation Committee for Aniket Murarka

certifies that this is the approved version of the following dissertation:

Building Safety Maps using Vision for Safe Local Mobile Robot Navigation

Committee:

Benjamin J. Kuipers, Supervisor

Kristen Grauman

Risto Miikkulainen

Brian Stankiewicz

Peter Stone

Building Safety Maps using Vision for Safe Local Mobile Robot Navigation

by

Aniket Murarka, M.S.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

August 2009

To my family

Acknowledgments

First and foremost I would like to thank my advisor, Ben Kuipers. I often wonder if I would have been able to cultivate my perspective on science – to take a broader view of the problems I am working on, to look for the right questions to ask, and to understand the value of my and other work – if it had not been for Ben. I greatly appreciate his patience in guiding me while at the same time giving me the freedom to pursue my line of thought and then gently leading me back to the right answer if I drifted too far.

I would like to thank my committee members, Peter Stone, Risto Miikkulainen, Kristen Grauman, and Brian Stankiewicz for their well thought out questions and suggestions during my proposal and defense. In addition, my individual interactions with them have helped me as a researcher. Peter's class on multi-agent systems showed me how one could build a complex system from scratch. Risto gave me a thorough introduction to Artificial Intelligence through his class when I was still considering doing a PhD in Computer Science. Kristen's excellent class on Computer Vision helped formalize my knowledge of the area, and Brian helped me understand the importance of evaluating my work properly. I would also like to thank Bruce Porter for graciously agreeing to be a part of the committee during my proposal to make quorum.

Joseph Modayil helped me greatly in formulating the problem I chose to work on in this thesis. His help was timely to say the least. Mohan Sridharan, Patrick Beeson, and, Shilpa Gulati have all contributed to this thesis. Their suggestions and comments from time to time proved to be very useful. Discussions with Jonathan Mugan on the nature

of life (and everything else) have provided much needed intellectual stimulation as have discussions with Changhai Xu, Subramanian Ramamoorthy, Jeremy Stober, Jeff Provost, Matt MacMahon, and Lewis Fishgold, my lab mates from past and present.

My old office mates and department friends, Prem, Misha, Subbu, Amol, Selim, Shivaram, Kotla, Un Yong, Alan, Glenn, Doran, Kokku, and, Arun have provided some of the most fun filled times at UT. My roommates from years past, Bala, Vishy, and, Shrik, and long time Austin friends, Murali and Dhananjay, have made life a bit easier and interesting at the same time.

This dissertation would not have been possible without the support of my family. My parents have been enormously supportive and waited with amazing patience for this day. Their presence in Austin during the final months of this thesis was critical, allowing me to finish much sooner. If there has been someone more worried about my thesis than me it is my sister – her and my brother-in-law's advice and help have been invaluable. I would also like to thank Richa's parents for their quiet understanding and support all these years.

My daughter Anusha has been a constant source of joy and wonder in these final mundane months as I have struggled to meet deadline after deadline. Her smallest smile or cry would brighten my day and give me renewed energy to push on. As I have worked on this thesis over the years no one has put up with more than Richa. Her faith in me and my work have been a source of confidence when there was none and kept me going.

ANIKET MURARKA

The University of Texas at Austin

August 2009

Building Safety Maps using Vision for Safe Local Mobile Robot Navigation

Publication No. _____

Aniket Murarka, Ph.D.
The University of Texas at Austin, 2009

Supervisor: Benjamin J. Kuipers

In this work we focus on building local maps to enable wheeled mobile robots to navigate safely and autonomously in urban environments. Urban environments present a variety of hazards that mobile robots have to detect and represent in their maps to navigate safely. Examples of hazards include obstacles such as furniture, drop-offs such as at downward stairs, and inclined surfaces such as wheelchair ramps. We address two shortcomings perceived in the literature on mapping. The first is the extensive use of expensive laser-based sensors for mapping, and the second is the focus on only detecting obstacles when clearly other hazards such as drop-offs need to be detected to ensure safety.

Therefore, in this work we develop algorithms for building maps using only relatively inexpensive stereo cameras, that allow safe local navigation by detecting and modeling hazards such as overhangs, drop-offs, and ramps in addition to static obstacles. The hazards are represented using 2D annotated grid maps called *local safety maps*. Each cell in the map is annotated with one of several labels: *Level, Inclined, Non-ground,* or, *Unknown*. *Level* cells are safe for travel whereas *Inclined* cells require caution. *Non-ground* cells are unsafe for travel and represent obstacles, overhangs, or regions lower than safe ground. *Level* and *Inclined* cells can be further annotated as being *Drop-off Edges*.

The process of building safety maps consists of three main steps: (i) computing a stereo depth map; (ii) building a 3D model using the stereo depths; and, (iii) analyzing the 3D model for safety to construct the safety map. We make significant contributions to each of the three steps: we develop global stereo methods for computing disparity maps that use edge and color information; we introduce a probabilistic data association method for building 3D models using stereo range points; and we devise a novel method for segmenting and fitting planes to 3D models allowing for a precise safety analysis. In addition, we also develop a stand-alone method for detecting drop-offs in front of the robot that uses motion and occlusion cues and only relies on monocular images.

We introduce an evaluation framework for evaluating (and comparing) our algorithms on real world data sets, collected by driving a robot in various environments. Accuracy is measured by comparing the constructed safety maps against ground truth safety maps and computing error rates. The ground truth maps are obtained by manually annotating maps built using laser data. As part of the framework we also estimate latencies introduced by our algorithms and the accuracy of the plane fitting process. We believe this framework can be used for comparing the performance of a variety of vision-based mapping systems and for this purpose we make our datasets, ground truth maps, and evaluation code publicly available.

We also implement a real-time version of one of the safety map algorithms on a

wheelchair robot and demonstrate it working in various environments. The constructed safety maps allow safe local motion planning and also support the extraction of local topological structures that can be used to build global maps.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Humans are able to move about in the world easily and robustly on their own. In unfamiliar environments we quickly learn new routes for going from one place to another and over time utilize these to build more complex representations of the environment. The learned routes and representations exhibit map like properties and are collectively referred to as the cognitive map [Kuipers, 1978; Trullier et al., 1997]. Building the cognitive map as we move about the world requires solving many problems such as identifying places, figuring out where we are, planning a route, and physically moving from one place to another while avoiding obstacles.

Autonomous navigation for robots aims to replicate the abilities of humans, i.e., to enable robots to move independently and robustly through the world. The problem of robot navigation is broken down along similar lines as in humans: place detection, localization, path planning, obstacle and hazard avoidance, and mapping. These sub-problems can be solved either independently or in conjunction with the others. Since autonomous navigation is a vast problem, researchers focus on particular types of environments and robots. This makes the problem tractable and provides insight into the mechanisms and representations required for a general solution.

It is hoped that by studying autonomous navigation for mobile robots we will get

a better understanding of the structure of the human cognitive map and on how humans represent spatial knowledge. Apart from satisfying scientific curiosity, autonomous agents have enormous application potential. An application that has motivated this work is the development of an autonomous robotic wheelchair to serve as a mobility assistant for humans who have disabilities associated with motion, communication, and perception. Such an intelligent wheelchair has the potential to open up the world for many people who otherwise cannot move around the world on their own.

1.1 Problem: Safe Local Mapping using Vision

The general problem we consider in this work is that of the safe autonomous navigation of mobile robots through an urban environment. There are two parts to the problem of autonomous navigation:

1. The problem of safe local motion, i.e., planning safe paths in the local surroundings of the robot or small scale space.

2. The problem of planning safe routes through a large scale environment.

Here, by a large scale environment we mean one whose spatial structure extends beyond the sensory horizons of the robot, e.g., a building or a university campus. Small scale or local space is defined as space whose structure is within access of the robot's sensors, e.g., a corridor intersection or a room. A critical prerequisite to being able to navigate safely in large scale space is being able to navigate safely in small scale space or locally.

Therefore in this work we focus on local navigation. The specific problem we address is that of building local metrical maps that will allow (a class of) mobile robots to plan safe paths in their local surroundings (in urban environments). In addition, we focus on using cameras as the primary sensor to build the maps. The class of robots we consider

are wheeled mobile robots that can only travel on smooth terrain, e.g., sidewalks and roads, but cannot travel on uneven surfaces or climb stairs.

The urban environments we consider are people friendly environments like a university campus with buildings and open spaces, with mostly pedestrian traffic and slow moving vehicles. We assume the environment conforms to the Americans with Disabilities Act (ADA), and follows various standards and guidelines [U.S. Department of Justice, 1994, 2000], so that most places are accessible by sidewalks and wheelchair ramps.

For a wheeled mobile robot to travel safely and autonomously in such environments, it has to detect and avoid a number of hazards. The most common hazards are obstacles - static obstacles such as walls and trees, and dynamic ones such as people. In addition to obstacles, there are a number of other hazards, such as drop-offs and ramps, that the robot must also detect and avoid or carefully navigate. In this work we focus on detecting (and representing) *static obstacles, overhangs, drop-offs, and, inclines.*

For the purpose of safe local motion for wheeled robots, it is possible to represent the relevant characteristics of the robot's local 3D environment with semantically annotated 2D grid maps, called *local safety maps*. These local safety maps can then used for continually planning safe paths in the local surroundings of the robot as it moves through the world. We introduce the exact safety map representation and explain why a 2D representation is sufficient for the purpose of safe local motion in Chapter 2. *Thus, building safety maps using vision that can be used for safe local navigation will be the main thrust of this thesis*.

To deal with the problem of planning safe routes over longer distances we explain how the local safety maps, when used with an existing large scale mapping framework such as the Hybrid Spatial Semantic Hierarchy (HSSH), permit the construction of large scale global maps (Section 2.2.3) which in turn allow large scale navigation. A nice feature of the HSSH is that it gives us the ability to deal with large scale navigation issues for little effort once we have the local safety maps.

Figure 1.1: The wheelchair robot used as the experimental platform in this work. The robot has a stereo camera and one horizontal and one vertical laser range-finder.

1.1.1 Experimental Platform: Vulcan, the Intelligent Wheelchair

Our experimental platform, representative of the class of robots mentioned above, is the Intelligent Wheelchair, Vulcan, shown in Figure 1.1. It is equipped with two laser range finders, one mounted vertically and the other horizontally, a stereo camera, and optical encoders. The laser range-finders are only used for the purposes of localizing the robot and for providing ground truth data as explained in Chapter 2. Range data from the lasers is not used for constructing the local safety maps.

1.2 Thesis Motivation and Overview

In addition to the potential benefits of autonomous navigation mentioned above, the problem addressed in this work is motivated by two shortcomings perceived in the literature on robot mapping. The first is the extensive use of expensive laser-based sensors for mapping, and the second is the focus on only detecting obstacles when clearly, other hazards such as drop-offs also need to be detected to ensure safe autonomous navigation.

Laser range-finders have been the predominant sensor of choice because they provide accurate range data with very little noise. However, since most laser range-finders can only sense distances in a 2D plane and cannot capture 3D information, the maps built using them miss several hazards, such as drop-offs and overhangs, and cannot be relied on for safe navigation in many urban environments. In recent times laser range-finders that return 3D range data have become more common, as witnessed in the DARPA Grand Challenges [Thrun et al., 2006]. However, such sensors are prohibitively expensive and if we wish for robots to be ubiquitous, an infeasible choice.

Cameras are used less frequently, and then too usually as a secondary source of information, because they are sensitive to environmental changes and the data they return is noisy and difficult to interpret. This is specially true of the range data obtained from stereo cameras that, besides being noisy, has the problem of being unreliable in regions of poor texture.

On the other hand cameras have lower cost than laser-based sensors, are usually smaller in size and provide a lot more information in a single frame of data. Being cheap and small, it is possible to mount several cameras on a single robot to provide a larger field-of-view which can be important for safety. On robots such as the Intelligent Wheelchair where passenger comfort is of prime importance, the flexibility provided by the smaller size is very desirable. Furthermore, the large amounts and types of information from a camera can be used for purposes other than safety, such as for object recognition.

In this thesis, we hope to overcome both shortcomings by introducing methods for building good quality stereo-based maps that can be used for safe navigation by detecting and representing hazards such as drop-offs, ramps, and, overhangs, in addition to obstacles. We begin by first defining local safety maps in Chapter 2. We explain how safety maps fit into the HSSH framework and also provide an overview of the general algorithm used to build safety maps.

In Chapter 3 on related work, we show where our work fits in the literature on robot

navigation and how it relates to other mapping algorithms. In particular we compare our methods against other geometric laser-based and stereo-based mapping methods and also against methods that use visual cues such as color and texture to distinguish between safe and unsafe areas.

Chapter 4 starts with an explanation of stereo camera geometry and then explains three different stereo methods for computing depth maps. One of the methods is a commercially available local correlation-based method and other two are global stereo methods that we propose. The chapter ends with an evaluation of the three methods on a standard dataset.

Once we have stereo depth data, the next step involves accumulating the data over time to build 3D models. In Chapter 5 we introduce two different methods for building the models that also reduce the amount of noise present in stereo data. In Chapter 6 we then show how to construct local safety maps from the 3D models. We present a fast method for identifying potentially traversable regions and fitting planes followed by an analysis to get the final maps.

Chapter 7 describes and evaluates a stand-alone method for detecting frontal drop-offs that only relies on a sequence of images from a monocular camera. The detected drop-off edges can be used to augment the safety maps created using stereo methods.

In Chapter 8 we introduce an evaluation framework for extensively evaluating the algorithms and comparing them to each other. We begin with a description of the datasets and the metrics that are computed and end with the results of a real-time version of one of the safety map algorithms on the Intelligent Wheelchair. Chapter 9 summarizes and outlines avenues for future work.

1.3 Major Contributions

We make the following contributions in this work:

- We introduce the notion of local 2D safety maps and show how the safety maps compress 3D information into a 2D representation sufficient for safe local travel (Chapter 2).
- We propose global color and edge segmentation based stereo methods for computing stereo disparity maps (Chapter 4).
- We introduce a probabilistic data association method for building 3D point cloud models from noisy stereo data. The method works by matching 3D points over time by computing the Mahalanobis distance between the point locations and visual features (Section 5.1).
- We introduce a method for analyzing 3D models for safety to build 2D local safety maps (Chapter 6). This includes a fast method for identifying and segmenting traversable ground regions (i.e., level or inclined surfaces on the ground) in 3D models and fitting planes to them.
- In addition to the 2D safety map we also introduce a hybrid 3D map (based on the above analysis) for representing a mobile robot's local surroundings. The map is a hybrid of a 3D grid, used for modeling obstacles and other unsafe regions, and of planes, for modeling the ground and other potentially traversable regions (Chapter 6).
- We develop a method for detecting drop-off edges in front on the robot based on visual motion and occlusion cues (Chapter 7). The method only requires monocular images and does not utilize stereo data.
- We propose a comprehensive evaluation framework for measuring and comparing the performance of mobile robot mapping algorithms on a set of video datasets (Chapter 8). In the framework, constructed safety maps are compared against ground truth

safety maps to compute different error rates. Latencies present in the system are measured and the accuracy of the plane fitting process is also estimated. We believe the evaluation framework will provide the robotics community with an objective way of comparing different mapping algorithms and towards this end we plan to make the datasets and associated ground truth publicly available [Murarka and Kuipers, 2009a].

- Finally, we provide a real-time implementation of one our algorithms for building safety maps on the Intelligent Wheelchair and demonstrate it being used successfully for local planning (Chapter 8). We also integrate our algorithm in the HSSH framework and demonstrate the successful extraction of local topological maps from safety maps.

Parts of the work presented in this thesis have appeared before in the following publications: [Murarka et al., 2006], [Murarka et al., 2008], [Murarka and Kuipers, 2009b], and, [Murarka et al., 2009].

Chapter 2

Local Safety Maps

2.1 The Local Safety Map Representation

To develop a suitable representation for the local safety maps we have to consider both the motion capabilities of the robot and the characteristics of the environment relevant for safety. In addition, the representation should be of a form usable by standard planning algorithms.

As mentioned in Chapter 1, the robots we consider can travel over smooth terrain only. The surface over which such robots move can be treated as a 2-manifold. It is the characteristics of this manifold that we want to capture, and locally it is possible to do so with a 2D plane. Given this we choose to use *annotated 2D grids* as the safety map representation. The 2D plane of the grid allows the robot to represent traversable surfaces whereas the annotations allow it to represent the hazards present in the 3D environment. Furthermore, the use of a 2D grid means that the safety maps can be used by standard planning algorithms for finding safe paths [1].

Table 2.1 lists the kinds of hazards that wheeled mobile robots have to avoid or carefully navigate in urban environments. Based on these we can annotate cells in the safety

[1] Since 2D grid maps are ubiquitous in mobile robotics, most standard planning algorithms, e.g., D* [Stentz, 1995], value iteration [Burgard et al., 1998], work with 2D grids.

Potential Hazards	Examples
Obstacles: Static	Walls, furniture
Dynamic	People, doors
Invisible	Glass doors and glass walls
Drop offs	Sidewalk curbs, downward stairs
Inclines	Wheelchair ramps, curb cuts, sloped sidewalks
Overhangs	Table tops, railings, tree branches
Uneven surfaces	Gravel paths, grass beds
Narrow regions	Doorways, elevators

Table 2.1: Potential hazards that a mobile robot must deal with in urban environments.

maps with the following labels corresponding to different characteristics of the environment:

1. **Level:** Cells with this label correspond to smooth horizontal ground regions in the world and are safe for travel.

2. **Inclined:** Cells with this label correspond to ramps and inclined ground surfaces that, while safe, require caution when travelling.

3. **Non-Ground:** Cells with this label correspond to either static obstacles or overhangs, or to regions below safe ground regions. They are unsafe for travel.

4. **Uneven:** Cells with this label correspond to terrain that is too rough or soft for travel.

5. **Narrow:** Cells with this label are in a region where there is little room for the robot to maneuver [2].

6. **Unknown:** Cells with this label are in a region for which there is either insufficient or no data.

A cell labeled "Level" or "Inclined" can be further annotated as a **Drop-off Edge**. Ideally, we should annotate the edges between cells as drop-off edges instead of the cells themselves. While it is certainly possible to do this, in practice we choose to annotate the cells

[2] In this work we do not detect or classify "Uneven" and "Narrow" regions.

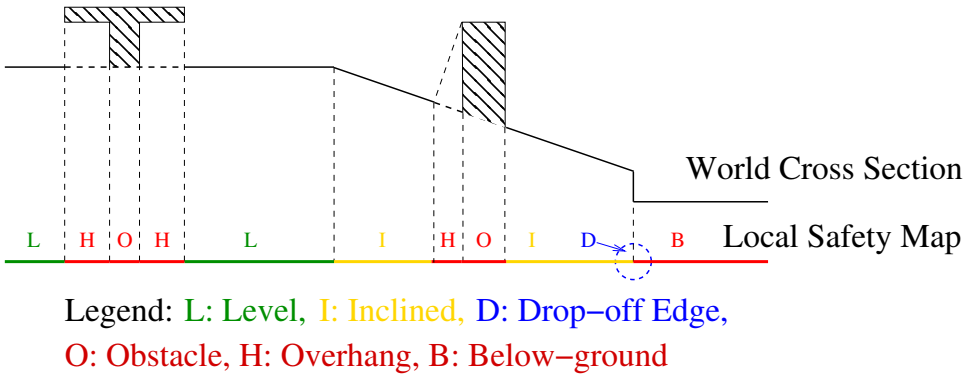


Figure 2.1: **The local safety map and its relation to the local 3D space it models.** "Level" surfaces in the world are safe for travel (marked green in the safety map). "Inclined" surfaces require caution (marked yellow) whereas "Non-ground" regions (in red) are unsafe and represent either obstacles, overhanging objects, or regions below traversable ground regions. "Drop-off Edges" are in blue and are to be avoided as well. Note that, ideally, objects in the 3D world are not projected directly onto the horizontal plane of the safety map. For example, the object on the incline is first projected onto the ground below it - then the region on the ground is projected onto the plane of the safety map. In practice, however, since the slopes of traversable inclines are low, projecting objects directly onto the plane of the safety map gives a good approximation. *This figure and others in this thesis are best viewed in color.*

themselves so that our safety maps can be used by existing planning algorithms - planners can simply treat cells marked as drop-offs as unsafe when planning. Since a drop-off edge is at the boundary between a higher surface and a lower one, we annotate cells adjacent to the boundary on the higher surface, since these cells require more caution during navigation. Section 2.1.1 discusses planning using the safety map in more detail.

Another detail associated with drop-offs edges is that it is not always possible to confirm their existence. This is because regions adjacent to drop-off edges are often occluded and get labelled "Unknown" in the safety map - making it impossible to compare the heights of cells adjacent to a drop-off edge. Thus we are often required to infer the existence of a drop-off edge and for such edges we use the label **Potential Drop-off Edge**.

Figure 2.1 shows pictorially the relationship between the 2D horizontal plane of the safety map and the local 3D space that it models. The projections of objects and regions in the robot's 3D surroundings onto the horizontal plane define distinct regions on the safety map plane. Each of these projected regions is then annotated with one of the labels defined above to give the safety map. The 2D representation and the simple semantics of the local safety map permit efficient local motion planning in a 3D world.

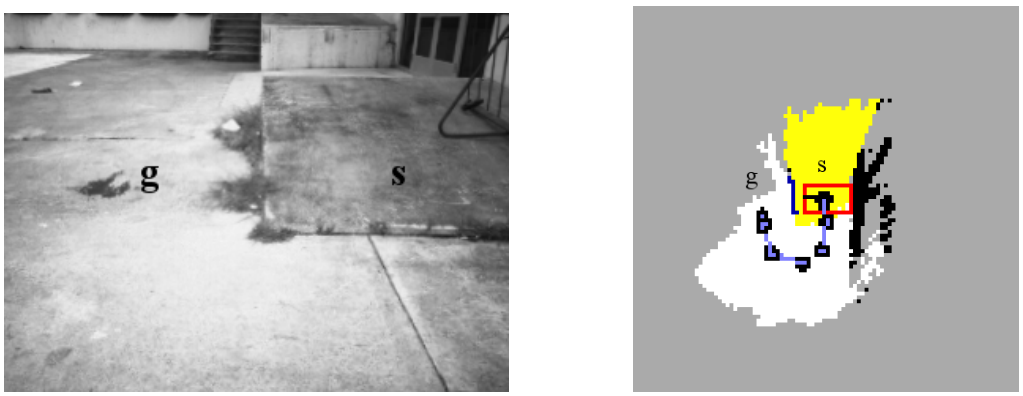
2.1.1 Navigating using the Safety Map

There are two parts to navigating using the safety map - the first is planning paths to desired goal points, and the second is making control and travel decisions while following the path.

The safety map can be used in a straightforward manner by 2D planners. Cells marked "Level" or "Inclined" in the safety map can be treated as safe, whereas cells marked "Non-ground" and "Drop-off Edge" should be treated as unsafe when planning a path to a goal point. Treatment of cells marked "Unknown" or "Potential Drop-off Edge" will depend on the application and the amount of risk the robot is willing to take. Usually it is acceptable to treat such cells as safe when planning, i.e., a planned path may pass through such cells, but it is not acceptable for the robot to actually drive over such cells. However, in practice, robots plan continuously as they move through the world and it is often the case that as the robot gets near such ambiguous regions, additional information usually removes or reduces the ambiguity allowing the robot to correctly plan through such regions.

Once the robot has a path to follow, the safety map can also assist the robot in making control decisions such as the speed with which it should travel. In "Inclined" regions, it makes sense for the robot to be more cautious and drive slowly and, as mentioned above, a robot may choose to avoid driving over an "Unknown" region when following a planned path.

Figure 2.2 shows the results of using an existing planner (a combination of a Rapidly-exploring Random Tree (RRT) planner [Kuffner and LaValle, 2000] and a trajectory gen-



(a) (b)

Figure 2.2: **Path planning using the safety map.** (a) Image from the stereo camera on the Intelligent Wheelchair showing an environment with an upward ramp and the resulting drop-off edge. (b) The safety map of the environment showing the robot (in red) and a planned path (in blue-black) from start, **s**, to goal, **g**. The start and end points are also shown in the stereo camera image. The safety map coloring scheme is as follows: "Level" regions are in white; "Non-ground" regions in black; "Inclined" regions are in yellow (representing the upward ramp in this case); "Unknown" regions are in gray; and "Drop-off Edge" cells on the ramp are in blue. The planning algorithm treats these blue cells (next to the actual drop-off boundary) as unsafe and plans a path around them even though the goal is directly ahead of the robot.

erator [Gulati et al., 2009]) on a safety map built using the Intelligent Wheelchair. The planning algorithm treats all regions marked "Level", "Inclined", or "Unknown" as safe and all other regions as unsafe, including cells labeled as "Drop-off Edges". The RRT planner finds a path by sampling points in the robot's configuration space, obtained by growing the safe space in the safety map by the size of the robot, and choosing a path that goes through safe cells with the required clearance. The trajectory generator smooths the path returned by the RRT planner, while satisfying velocity and other dynamic constraints, to generate the final path shown in Figure 2.2(b).

2.2 Relationship to Large Scale Mapping

In this section we explain how local safety maps fit in the framework of a large scale mapping method, in particular the Hybrid Spatial Semantic Hierarchy (HSSH), and thus can be used towards solving the problem of large scale navigation. We first examine the two main paradigms of mapping: metrical and topological, and their applicability to the problem of mapping large scale spaces. We then discuss the HSSH, a hybrid mapping framework that combines both metrical and topological mapping methods.

2.2.1 Metrical Mapping

As the name suggests, the goal of metrical mapping is to build geometric maps of environments. In recent years, great progress has been made in the field of robot metrical mapping [Thrun, 2002]. Using powerful probabilistic methods it is possible to construct accurate 2D maps of level urban environments. Most of these methods employ laser range-finders as their primary sensor and construct 2D occupancy grid maps of the world (Figure 2.3(a)). Laser range-finders have also been used for constructing 3D maps, e.g, [Newman et al., 2006] use a 2D laser range-finder mounted on a rotating assembly to build a 3D model of an urban environment. Apart from robotics researchers, people working in computer vision have also successfully investigated building metrical models of environments [Hartley and Zisserman, 2000]. Most of the above metrical mapping methods have been applied to small environments, e.g., a room or a small building.

Metrical mapping of larger environments also attempts to build geometrically accurate maps except on a much large scale. This requires solving a number of difficult problems such as maintaining accurate 3D pose estimates over large distances and detecting loop closures in three dimensions [Modayil et al., 2004]. While having global metrical maps can be useful for some applications, on the whole they seem largely unnecessary and cumbersome for large scale navigation specially when dealing with environments as large as a University campus or even a city. A more compact representation is required - topological maps

provide such a representation and are discussed next.

Nevertheless, metrical methods are very effective at building accurate geometrical models (2D or 3D) of small scale environments that prove to be very useful for planning safe local motion.

2.2.2 Topological Mapping

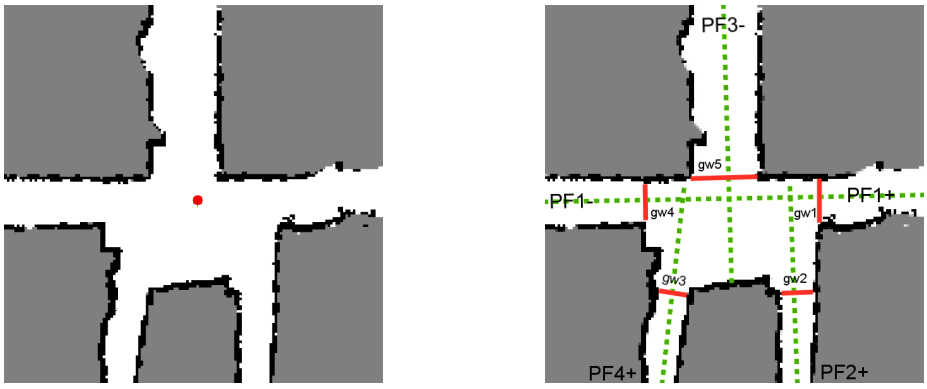
A topological map models the world as a collection of *paths* and *places* with connections and order relations amongst them. Informally, *places* are areas of interest in the world, e.g., street corners, corridor and street intersections, or rooms. *Paths* usually correspond to sidewalks and corridors. *Place* and *path* elements form the basis of most topological methods, though the methods differ in their definitions [Kuipers, 2000; Choset and Nagatani, 2001].

Thus, a topological map provides a much more compact representation of the world than a metrical map, that can be used easily for large scale navigation. A topological map does not require maintaining highly accurate pose estimates over large distances, and as a result problems of large scale environments such as closing large loops become less expensive and simpler to solve [Kuipers et al., 2004].

Despite these advantages, we can see that in building *only* topological maps of large scale space we lose detailed metrical information that is useful for local motion planning. However, it is possible to combine metrical maps with topological maps, and hybrid mapping methods do just that.

2.2.3 Hybrid Mapping: The HSSH

Hybrid mapping approaches combine the strengths of metrical and topological approaches. Generally in hybrid mapping, topological methods are used to map the large scale structure of space and metrical methods are used to model the small scale structure [Kuipers et al., 2004; Bosse et al., 2003]. This gives both the required detailed metrical information and a compact representation of large scale environments.

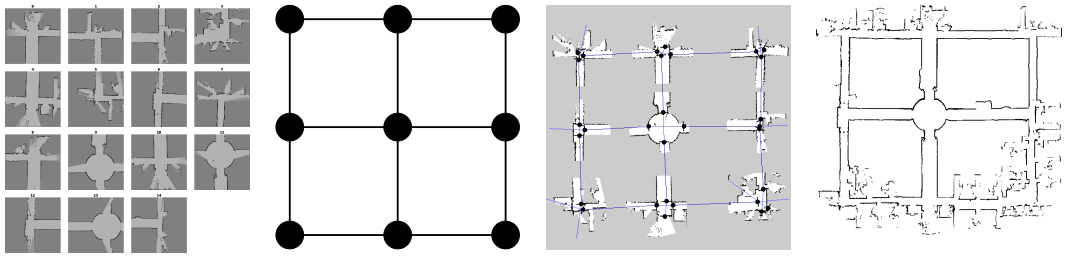


(a) **(b)**

Figure 2.3: **A local metrical map and its local topology.** (a) A grid map of a small scale environment constructed using laser range-finders. The white areas represent free space, black areas represent obstacles, and grey areas represent unexplored space. We call such a small scale map a local metrical map. The robot is shown as a circle in its center. (b) A pictorial representation of the local topology of the *place* described by the local metrical map. The dashed green segments represent *path* fragments (PF1, PF2, etc) that pass through or terminate in the *place* whereas the solid red segments (gw1, gw2, etc) are *gateways* into and out of the *place* [Beeson et al., 2005].

We use one particular hybrid approach, the Hybrid Spatial Semantic Hierarchy (HSSH) [Kuipers et al., 2004], as our framework for constructing large scale maps from local maps. The HSSH uses metrical mapping methods to give the robot detailed metrical information about its small scale space. The local metrical map, as it is called, *scrolls* with the robot when it moves between *places* in the world providing it with the necessary metric information for travel. When the robot arrives at a *place*, the local map stays stationary as the robot explores the *place* completely. The completed local metrical map of a *place* (Figure 2.3(a)) then allows the robot to extract its *local topology*. Figure 2.3(b) shows a pictorial representation of the local topology of a *place* which is a circular ordering on the directed *gateways* and directed *path* fragments in the *place* [Kuipers et al., 2004].

To construct a global map from the local metrical maps in the HSSH, *places* in the robot's environment have to be detected [Beeson et al., 2005] and their local metrical maps



(a) **(b)** **(c)** **(d)**

Figure 2.4: **Building large scale maps from local metrical maps.** (a) Construct local metrical maps of every *place* in the robot's environment. (b) Extract the local topology of every *place* and search through the space of possible topological maps consistent with the local topologies and the robot's exploration sequence to find the correct global topological map. (c) Annotate the local metrical maps with the topological map to get a *patchwork map*. (d) A global metrical can also be constructed, if required, using the global topology and robot travel estimates [Modayil et al., 2004].

constructed (Figure 2.4(a)). The local metrical maps allow the extraction of local *place* topologies and these along with travel information, are used by the HSSH to construct a large scale topological map (Figure 2.4(b)). The topological map can be annotated with local metrical maps at every *place* to give a *patchwork map* (Figure 2.4(c)). It is also possible to construct a global metrical map of the environment using the global topological map and robot travel estimates, as shown in Figure 2.4(d). Thus, the HSSH framework gives us the ability to construct maps of large scale environments using 2D local metrical maps.

The similarity of the local safety map to the local metrical maps used in the HSSH, means that the safety maps can be used in the HSSH with almost no effort. In Chapter 8 we show the results of integrating the safety maps with our existing HSSH implementation. The safety maps permit the extraction of local topologies as shown in Figures 8.12 and 8.13 in Chapter 8, and at the same time enable safe navigation.

2.3 Constructing the Local Safety Map

We now give an overview of the process followed in this work for building a safety map. We begin with a framework that outlines the major components that algorithms for constructing safety maps, and more generally local metrical maps, should have.

2.3.1 General Framework

The framework we introduce here helps to categorize the work that has already been done in the mapping literature and therefore also identifies the major problems that are yet to be solved. The process of constructing local safety maps can be broadly thought of as consisting of the following steps:

1. **Localization and Geometric Reconstruction:** To obtain a useful and accurate safety map the robot has to construct a geometric model of its local surroundings. The geometric model tells the robot the location of various objects and surfaces in its surroundings with respect to its own position. Localization is an integral part of this reconstruction process as it keeps track of the robot's pose while the robot moves through the local environment.

2. **Safety Classification:** The different regions in the safety map, corresponding to various objects and regions in the robot's surroundings, have to be classified as being safe or unsafe (or more precisely, classified with one of the labels defined in Section 2.1). The classification process has two components:

   (a) **Classification based on geometric information.** Geometric information about objects/regions can be used to deduce a large amount of necessary safety information. For example, all vertical objects can be considered unsafe, whereas all level surfaces can be considered potentially safe. The majority of the work presented in this thesis is based on using geometric information.

(b) **Classification based on non-geometric information.** Pure geometric information alone is not sufficient for distinguishing between safe and unsafe regions. For example, both a sidewalk (safe) and its dirt shoulder (unsafe) can look safe to a robot if it only considers geometry. However, color and texture information can distinguish between them. It is important to note that though such properties help in safety classification, we still require geometric information to locate the classified objects/regions in the safety map. In Chapter 7 we introduce a method that uses non-geometric information to identify drop-off edges.

2.3.2 Overview of the Process for Constructing Safety Maps

In this work we propose several algorithms for building local safety maps using vision. The algorithms have a common structure – they consist of three major steps. As the robot explores its local surroundings it gets a constant stream of stereo images. Each time the robot receives a new stereo image pair/frame, it processes the images to update its current knowledge of the world. The following steps are involved in processing each new stereo frame (the steps are shown pictorially in Figure 2.5):

1. **Computing a Depth Map:** A disparity map relative to the left image is computed using a stereo method (Figures 2.5(a) and 2.5(b)). The range readings obtained are transformed into world or map coordinates from the camera coordinates (we assume that the robot is able to localize - see below). We use three different stereo methods for computing the depth maps. The end result of this step is a collection of range readings in the map coordinate frame.

2. **Building a 3D Model:** The range readings from stereo are used to update a 3D model. The 3D model also helps in handling the noise present in stereo data. We present two methods for constructing the 3D model - one based on occupancy grids and the other a probabilistic method based on matching range points across frames. In both cases the final form of the 3D model consists of a 3D point cloud (Figure 2.5(c)) with a 3D

grid (Figure 2.5(d)) overlaid on top. The points and grid are in the map coordinate frame.

3. **Analyzing the 3D Model for Safety:** In the final step the 3D model is analyzed for safety to yield the annotated local safety map. The safety classification is based only on geometric properties of the environment. We present three ways of making the safety map.

   (a) **Based on Height Thresholds:** The 3D model is simply thresholded by height using knowledge of the robot's current location. This produces a safety map that can only be used in level environments.

   (b) **Based on Grid Traversability:** A traversability analysis of the 3D grid is performed that identifies traversable ground regions or segments in the grid. This produces a safety map that works for many non-level environments. However, since the method identifies small changes in height as being traversable, it cannot distinguish between level and inclined regions.

   (c) **Based on 3D Planes:** This method fits planes to the segments identified by the previous method and then analyzes the relationship between neighboring planes to produce the final safety map (Figure 2.5(e)). We believe this safety map should be usable in a wide variety of urban environments. The output of this method can be used to produce a hybrid 3D map (Figure 2.5(f)), consisting of a 3D grid and planes, that might be useful for other applications.

Since the focus of this work is on building models of the environment appropriate for safe navigation, we assume that the robot pose is known. For our experiments we use a laser based 3 degrees of freedom (DOF) localization algorithm [Beeson et al., 2006] for providing the robot's pose to our mapping algorithms (the 3-DOF method can be replaced by a camera based 6-DOF SLAM algorithm [Comport et al., 2007]). Since we use a 3-DOF localization module, our robot is restricted to travelling only on level surfaces for

the experiments reported in this work. However, our algorithms are general and applicable without modification to the case when the robot's motion has 6-DOF.

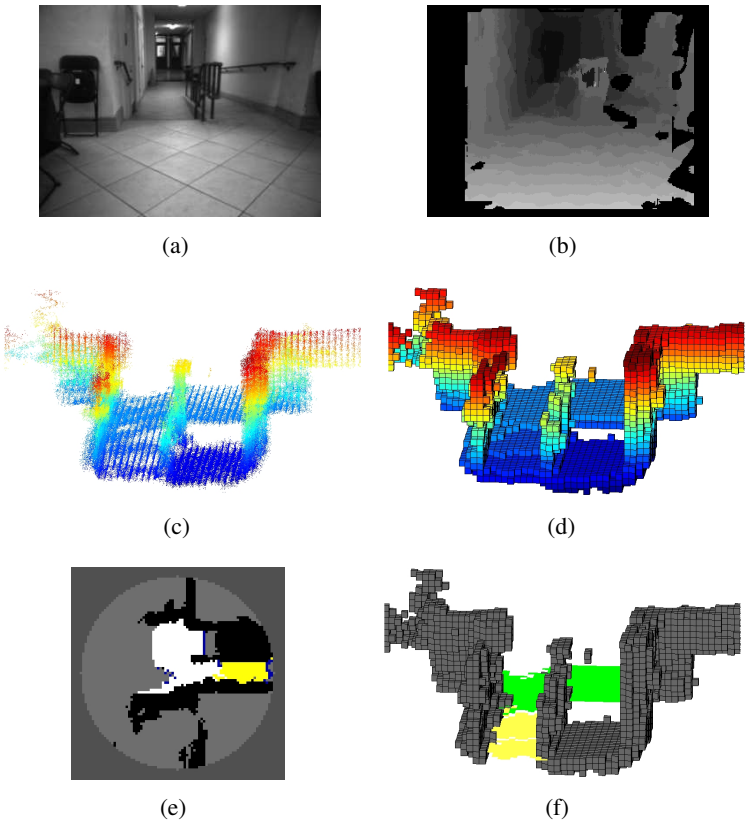


Figure 2.5: **Constructing Safety Maps.** *Step 1: Computing a Depth Map*. (a) Left image from the stereo camera showing a scene from an environment with a drop-off and a ramp. (b) Disparity map computed for the stereo pair (brighter areas closer) using a correlation stereo method. *Step 2: Building a 3D Model*. (c) 3D point cloud and (d) 3D grid (rendered from a viewpoint different from the image above) of the environment constructed using an occupancy grid method. *Step 3: Analyzing the 3D Model for Safety*. (e) A safety map produced by analyzing the 3D grid for traversability followed by plane fitting and analysis of the relationship between planes for safety. The color scheme is the same as that used Figure 2.2 with the addition of dark grey areas (outside the circle) for representing unexamined regions. (f) A hybrid 3D map of the environment that can also be constructed once the safety analysis is done. Unsafe regions are represented using voxels (grey). Safe ground regions are represented using planes: green for level planes; yellow for inclined planes – in this case a wheelchair ramp.

Chapter 3

Related Work

In this chapter we review other robot mapping systems that are related to our safety mapping system as a whole. We also review other pieces of work that are related to the individual components that form part of our mapping system. Some of these individual works are described in Chapters 4 to 7 where we felt it was more appropriate to discuss them.

3.1 Robot Mapping Systems

We begin by reviewing several related robotic systems that use lasers, cameras, and other sensors to determine the traversability or safety of the robot's environment. The systems are varied, from off-road vehicles that participated in the DARPA Grand Challenge to humanoid robots navigating indoor environments, and therefore have different traversability requirements.

We describe in the following sections robot systems that primarily use stereo information for determining traversability, followed by systems that use visual cues other than stereo, i.e., appearance-based systems. We then discuss laser-based systems and systems that combine information from multiple sensors to determine traversability.

3.1.1 Vision based Systems: Stereo

We discuss several stereo vision based mapping methods for detecting different kinds of hazards. We begin with two pieces of work that are closest to our own. Both methods construct annotated maps with labels and use stereo vision to do so.

Gutmann et al. [2008] present a real-time stereo based mapping system for humanoid robots that creates a 2D annotated grid map similar to our safety map. Their method for building maps consists of first segmenting a disparity map, produced using a local stereo method [Sabe et al., 2004], into planes using a variation of a range image segmentation method [Jiang and Bunke, 1994] [1]. They use all range readings to update a 3D occupancy grid and only readings corresponding to horizontal planes to update a floor height map. The occupancy grid and floor height map are then combined to create a Floor Obstacle Grid (FOG) map with the regions classified into six categories: *obstacle, tunnel, border, floor, stairs,* and *unknown*.

Therefore, similar to our work they detect obstacles, overhangs, and drop-off edges. However, they do not detect inclines – their system assumes a level ground when building the floor map. In our work, we segment the 3D model and fit planes to find inclines. Also, when labeling drop-off edges they only consider height differences and do not consider the case that regions beyond drop-off edges might be unknown like we do (Section 6.4). While we do not explicitly label stairs like their system does (since we are interested in wheeled robot navigation), our system is capable of finding stairs as shown in Figures 8.4 and 8.7 in Chapter 8.

Rusu and colleagues [Rusu et al., 2008; Morisset et al., 2009] present a real-time stereo algorithm that creates a labeled 3D map consisting of polygonal patches. They use a local stereo algorithm to compute depths from every stereo image pair and create point clouds. An octree data structure is created for storing the points and discretizing space

[1] In our work, the segmentation stereo algorithms of Chapter 4 achieve a similar result. The difference is that in their method the planes are obtained after stereo computation whereas in the our method the planes are obtained as part of the stereo computation.

(this is similar to our 3D models that consist of a 3D grid and point cloud). Then points in each occupied octree cell are modeled using convex polygonal planar patches – a plane is fit to points in each cell using a variation of RANSAC [Fischler and Bolles, 1981] and convex hulls are found to get the polygonal patch for each cell. Polygonal patches across different stereo frames are combined and refined to remove duplicates. They also have a simple outlier removal process in which they remove lone planar patches. The final 3D model consisting of polygonal planar patches is annotated based on the patch slopes with one of five labels: *ground, level* (here level refers to traversable regions that are above the ground plane), *vertical, steps,* and, *unknown*. The *ground* and *level* labels include inclined surfaces.

Like our work, their system works in environments with inclined surfaces but unlike our work, inclines are not distinguished from horizontal surfaces. They do not detect drop-off edges or overhangs. Also, the analysis in their method is very local – each patch is analyzed individually to determine its label. Furthermore, they do not have a proper process for removing noise from stereo range readings. We believe that the combination of these two reasons, specially a local analysis, can lead to incorrect labels. In our work we have methods for removing stereo noise (Chapter 5) and we label on the basis of much large regions (Chapter 6).

Unfortunately, neither Gutmann and colleagues, nor Rusu and colleagues, quantitatively evaluate the accuracy of their final maps thereby not permitting an objective comparison with our work. We believe that our datasets with their associated ground truth and our evaluation methodology can fill this gap by providing a common evaluation framework to the mapping community.

Singh et al. [2000] also use 2D local grid maps made using stereo for the navigation of a planetary rover. They construct the grid maps from "goodness" maps made at each frame. The "goodness" of a grid cell is a combined measure of the pitch, roll, and roughness of a plane fitted to a local patch of stereo range data around that cell. In addition a certainty

measure for each cell is computed based on the number of range points in its patch and the "age" of the cell. Finally, the robot plans routes based on the local map and calculates the traversability of each route based on goodness and certainty values. The system is tested in complex simulated environments and outdoors in simpler environments. Again, the analysis of traversability is very local in nature – it is done on the basis of small planar patches that can lead to errors. Furthermore, they focus primarily on detecting obstacles.

Other stereo algorithms along these lines include work by Iocchi et al. [2000] and Murray and Little [1998]. Iocchi et al. [2000] present a stereo vision based system for building planar 3D models. However, they only consider environments with vertical walls and a level ground. In the method by Murray and Little [1998], 2D occupancy grid maps are built by compressing 3D stereo range data into 2D range data before adding the range scans to the map. This is very similar to how horizontal 2D laser range-finders are used for creating maps – the existence of the ground is assumed meaning that hazards like drop-offs cannot be detected.

Rankin et al. [2005b] evaluate several vision based methods for the detection of various hazards such as obstacles, overhangs, drop-offs, trees, water bodies, and highly sloped regions. Most of the methods use stereo vision however some methods use other visual cues in addition such as color. They evaluate two methods for the detection of drop-offs. The first method by Bellutta et al. [2000] looks for gaps (unknown regions) directly in stereo range data. The second method looks for height differences *and* gaps in local terrain height maps of the environment to identify drop-offs. The results from both methods are combined to identify drop-offs. Both the above methods are designed for detecting drop-offs caused by holes, i.e., a drop-off edge at a sidewalk curb won't be detected. Our methods for detecting drop-offs (Chapters 6, 7) detect both types and in addition use other visual cues such as occlusion and motion (Chapter 7).

Another method by Rankin et al. [2005a] identifies highly sloped regions by fitting planes to fixed size local patches to determine the slopes of the planes. If the slope is

determined to be above a threshold the region is marked as such. In our work highly sloped and vertical regions are detecting by analyzing the 3D grid model as described in Chapter 6. Also of interest in the above work are methods for the detection of hazards that we do not consider in our work, such as water [Rankin et al., 2005a], which is detected using color, texture, and stereo range (estimating depth of reflections). Tree detection [Huertas et al., 2005] is another example where edge detection is used in addition to stereo (trees will get marked as "Non-ground" regions in our work).

Other methods for detecting drop-offs include work by Se and Brady [2002] and Rankin et al. [2007]. Se and Brady [2002] present a method for detecting drop-off edges at sidewalk curbs using a stereo vision camera. They assume that all ground surfaces are horizontal and find the curb drop-off by comparing the height difference between planes fit to the before and after the curb edge. They do not consider inclined surfaces and the case that regions beyond drop-off edges might be unknown like we do. Rankin et al. [2007] merge stereo vision and thermal signatures to detect drop-offs at night. Drop-offs are identified using temperature differences between them and surrounding areas. This method requires the use of special infrared cameras.

In addition to the ones described, stereo vision has been used for obstacle and hazard detection for a variety of applications. Labayrade et al. [2005] use *v-disparities* to detect obstacles for a vehicle driving on city roads; Byrne et al. [2006] use stereo for obstacle detection for an unmanned air vehicle; Se and Jasiobedzki [2006] develop a hand held stereo vision system that they use for mine navigation. Diebel et al. [2004] use "active" stereo vision to build 3D maps with the ICP algorithm. The methods are "active" in the sense that they have use artificial lighting to create patterns on the environment allowing for better depth computation using stereo methods. A review of vision based methods for navigation is presented in DeSouza and Kak [2002].

3.1.2 Vision based Systems: Appearance

In our work we mostly focus on detecting hazards based on geometric information (the exception is the motion and occlusion based method in Chapter 7 for detecting drop-offs). However, as discussed in the last chapter, while geometric information provides the majority of safety information it is not sufficient – there are situations when visual cues such as color and texture can provide important information. In this section we briefly review systems from the literature that use appearance based cues for determining traversability – we begin with systems that use color and texture and then describe a few learning based systems from the DARPA Learning Applied to Ground Robotics (LAGR) project. These methods can be thought of as extensions to be used on top of the geometric safety mapping techniques introduced in our work.

Ulrich and Nourbakhsh [2000] use the colormaps of previously traversed surfaces to classify new surfaces as being safe or dangerous. As the robot moves over surfaces known to be safe (either through initialization or past experience), it learns the colormaps of these surfaces. When the robot moves into a new region, it uses previously learned colormaps to distinguish between safe and unsafe surfaces in the new region. Then assuming it is in an environment with level ground, the robot uses simple geometry to determine the physical extent of the safe surfaces. The system is tested in both indoor and outdoor urban environments. A similar method presented by Dahlkamp et al. [2006] was used on the Stanford DARPA Grand Challenge vehicle (described in the next section). Such methods based on using only color are simple and work well in limited environments. They suffer from a lack of generalizability (e.g., what happens if the robot has only seen unsafe regions that are black in color and all of a sudden encounters unsafe regions that are white) and can only be used as add-ons to existing systems.

Alon et al. [2006] develop a texture and optical flow based method to find the path for an off-road vehicle. Their system has two independent modules for finding the path and then uses the result of the module with higher confidence. One module assumes the world is

level and has a road bounded by parallel edges. It then uses camera roll and pitch and optic flow to determine the road boundaries and direction of the road ahead. The other module trains a classifier to use texture for classifying the world into road and non-road regions. Because the system uses two modules it is fairly robust to failure but there are instances when the system gets confused, e.g., in the presence of shadows.

The DARPA LAGR project emphasized the use of learning and image appearance for making long-range traversability decisions [Jackel et al., 2006]. The robots provided as part of the project had stereo vision cameras that gave short range depth information. The robots used the depth data and the appearance of pixels, for which the depths had been calculated, to predict the traversability of far-off regions seen in the robot's images. The appearance-based vision systems were used as add-ons to the base stereo based navigation system. We discuss here three representative methods developed as part of the LAGR program. All the systems developed were meant to be used for outdoor navigation over vegetated terrain.

The LAGR system by Konolige et al. [2006] uses color information to find paths and line-of-sight information to find obstacle free regions (regions upto an obstacle along the line-of-sight are obstacle free and hence traversable). A classifier learns color models associated with path and non-path regions by training on image pixels labeled using stereo data. The classifier then labels image regions not labeled using stereo data. Traversable image regions, as determined using the classifier and the line-of-sight method, are projected from image space into 3D Euclidean space to update a local map (to which stereo information is also added). The projection requires the assumption that the ground is part of a single plane which is computed by fitting a plane to stereo points using RANSAC.

The method by Kim et al. [2006] uses texture to identify traversable regions. Texture, stereo, and, tactile information is used to train a classifier to categorize patches in an image. To collect training data the robot initially explores the environment unsupervised and constructs local maps using stereo. The robot is free to explore and even allowed

to bump into obstacles, and uses this exploration experience to automatically label image patches (corresponding to grid cells in the local map) as being either traversable or not traversable. It then uses this self-labeled data to train a classifier for future prediction. From their results the system seems to work fairly well at detecting obstacles. By using its experience from bumping into obstacles, the robot is able to learn that tall grass is traversable by their robot even though it appears to be an obstacle based on height.

Hadsell et al. [2007] do not explicitly use color and texture information and instead used feature vectors obtained from image patches. They create a 2D local map (in polar coordinates) and a global map for short and long distance planning respectively. The maps include information from the combined output of a short range stereo system and a long range vision system. At each frame a plane is fit to the ground in front of the robot (assuming that the ground can be approximated using a single plane). Then with the help of the ground plane, stereo range readings are added to a 2D quad-tree that stores the stereo points and their labels (ground vs. obstacle). A multi-scale image pyramid is created and small overlapping windows in the pyramid are annotated with the labels of corresponding stereo points. The system learns beforehand, using existing datasets, a set of radial basis function (RBF) centers. The image windows and the RBF centers are used to construct feature vectors that are used along with the labels to train a logistic regression classifier. Training of the classifier is done online at each image frame. After the classifier parameters have been updated for a given frame, the output of the classifier is used to classify *all* windows in the image pyramid (including those with stereo labels) as being part of the ground or obstacles. The system works fairly well with an average error rate of about 15 percent. This method was recently extended to use deep belief networks to learn features instead of RBFs [Hadsell et al., 2008].

One of the assumptions made in all the methods above is that the ground is either relatively flat or can be represented using a single plane – this is necessary to project distant image points into 3D space. Furthermore, unlike our work, all the above LAGR methods

focus on the detection of obstacles – other hazards like drop-offs are ignored.

Although the above systems focus on using appearance based information they all also build metric models of the environment for navigation. Michels et al. [2005] present a robot system that directly uses images to navigate - no explicit metric model is built. They use supervised learning to learn depth in monocular images from color, texture, and other visual cues, (similar to [Saxena et al., 2005] in Section 3.2.2). Then they divide the image into vertical stripes and predict the depth of the closest obstacle in each stripe and learn a policy for driving their robot (a radio controlled car) using reinforcement learning. Thus, they do not build an explicit metric model of the world and just use the predicted depth in images to navigate the robot making the method purely reactive. However, as mentioned, pure appearance based methods have limitations and can only work in a restricted range of environments – in the above system the environment is implicitly assumed to be near level and free of other kinds of hazards.

3.1.3 Laser based Systems

In this section we describe robot systems that rely primarily on laser range data to determine the safety/traversability of their environment. We give examples of systems for finding various kinds of hazards including obstacles, surface roughness, drop-offs, dynamic obstacles, etc. Although we only review a handful, there is a great variety of robotic systems that rely on laser data for navigation.

Amongst the most well known of robot systems are the ones that took part in the DARPA Grand and Urban Challenges. We briefly overview the methods used on Stanley [Thrun et al., 2006], the robot that won the second DARPA Grand Challenge [Iagnemma and Buehler, 2006]. The robot vehicle had multiple lasers mounted on top that were used to construct a local 3D point cloud of the road and other regions in front of the car. To identify obstacles the difference between the height coordinates of nearby points was used. If the

difference was large it was taken as evidence for an obstacle [2]. The obstacle information was incorporated into a 2D grid map and areas were classified as *drivable* and *obstacles*. The terrain analysis was quite simple but sufficient for the needs of the challenge. They also used a camera on top of the vehicle to identify drivable areas at large distances using color [Dahlkamp et al., 2006], as mentioned in the previous section. The vision system was only used for speed control.

Stavens and Thrun [2006] present a system that learns the roughness of surfaces based on laser range data. The system uses a self-supervised approach in that the robot drives over terrain and applies roughness labels, based on the shock its accelerometers experience, to corresponding laser data. This data is then used to train a learning algorithm that predicts roughness based on new laser data. This method is a promising way to extend our own work for detecting rough surfaces using stereo.

Wellington and Stentz [2004] also use self-supervised learning to determine terrain traversability for a tractor robot navigating in fields. The fields have tall grass, some of which is traversable, and the task is to estimate the true height of ground hidden beneath the dense vegetation. They train a classifier to predict the true ground height based on various features extracted from laser range data. They model the world geometrically as a 2D horizontal grid where each cell stores the height of the ground above it. They gather training data by having the robot drive over tall grassy terrain and label corresponding voxels with their true height. The labeled data is then used to train the classifier. The system seems to work quite well and the problem tackled here is an unusual one, because the desired geometry of the environment is "hidden" from the robot's range sensors and has to be deduced.

Heckman et al. [2007] present a method for finding drop-offs (or negative obstacles) using 3D laser data. They create a 3D grid and label voxels in the grid as belonging to ground and non-ground objects (the non-ground objects are classified further). They ray-

[2] Surprisingly, one of the main problems in getting this simple obstacle detection method to work was getting accurate pose estimates which was particularly hard as the road was rough.

trace to find occluded voxels in the 3D laser grid and determine the cause of the occlusions – whether the occlusions are caused by ground or non-ground objects. Furthermore, they check to see if the region after the occluded voxels is occupied. Based on this they identify the occluded voxels as *potential negative obstacles* and annotate it with details on the nature of the occlusion (occlusion caused by ground or non-ground objects; region after occluded voxels is occupied or unknown). This analysis is similar to what we do in Chapter 6 for finding potential drop-off edges using stereo data. However, we fit planes to the data before finding the drop-off edges which gives a better estimate of the height difference between surfaces – this is specially important for stereo data (as illustrated in Chapter 6).

Thrun et al. [2004] present a method for modeling corridor style environments using planes. They fit the planes to a 3D point cloud of laser range points using EM where each plane is a bounded rectangular region. The final model constructed consists of several such planes overlapping each other. We discuss the EM based method for fitting planes and compare it with our own plane fitting method further in Chapter 6.

Prassler et al. [1999] use a 2D laser range-finder to identify dynamic obstacles. They use their method to navigate a wheelchair robot during rush hour in a railway station and other environments – the system is unique in its performance in crowded *dynamic* environments and a promising direction for extending our work.

3.1.4 Multiple Sensor based Systems

We end our review of robot mapping systems by presenting a couple of representative methods for mapping that use multiple sensors. We have already seen some examples of such methods in the previous sections in the works by Kim et al. [2006] (cameras and bump sensors), Thrun et al. [2006] (lasers and cameras) and Rankin et al. [2007] (regular cameras and thermal cameras). The use of complementary sensors on a robot allows a wider variety of problems to be solved more easily. Problems that are difficult to solve using one sensor might be easier to solve using another sensor. In our own work, we expect to use additional

sensors in the future such as sonars and bump sensors to detect invisible obstacles (e.g., glass walls) that can be very difficult to detect using vision.

Wellington et al. [2005] improve on the methods discussed above [Wellington and Stentz, 2004] for detecting true ground height beneath vegetation by using multiple sensory modalities: laser range data, laser reflectance, infrared temperature (from an infrared camera), and color data. First, the world is modeled geometrically using a 3D grid of voxels where each voxel can be classified as ground, vegetation, free-space or as an obstacle. Then they use a self-supervised approach to gather training data by having the robot drive over tall grassy terrain and label the voxels, corresponding to the terrain, with the categories defined above. Finally, the labeled data is used to train multiple MRFs each modelling the different object classes. The trained MRFs are then used to predict the true ground height for each of the classes. The use of several sensors allows them to distinguish between the different object categories better and predict ground height more accurately. An example given is the use of the infrared camera to detect a person hidden in vegetation (the person has a higher temperature than the vegetation). The use of the infrared camera allows the corresponding region to be labeled as an obstacle instead of vegetation.

[Rasmussen, 2006] use both laser and vision based methods for driving a vehicle on rural and desert roads. The vision component uses texture to find ruts and tracks in the ground which are used to find the road vanishing point (that can be hard to find using lasers). This yields a direction for steering the vehicle. The laser component is used to detect obstacles (that are harder to find using vision) along that road direction to center the vehicle in the road. The system was tested (and used) on the Caltech DARPA Grand Challenge vehicle and gave good performance.

3.2 Methods for Computing Depth and 3D Reconstruction

An important part of almost all the robot mapping methods discussed above is the ability to estimate the depth of objects in the environment and reconstruct a model of the environment.

Laser range-finders directly return depth measurements to objects that are accurate and almost free of false positive readings allowing environments to be reconstructed relatively easily. The information provided by cameras on the other hand has to be heavily processed to produce the depth estimates. In this section we look at different methods for computing depth from images and for obtaining 3D reconstructions of environments. In our work we have focussed on using stereo depth information but here we also look at methods that involve computing depth from a single image and *Structure from Motion* methods [Hartley and Zisserman, 2000].

3.2.1 Depth from Stereo Images

Scharstein and Szeliski [2002] provide an excellent overview of stereo image based methods for computing depth. Broadly, stereo algorithms can be split into two types – local algorithms and global algorithms.

Local algorithms do not take the overall structure of the image into account when computing the depth at a pixel – they only consider a local image patch around the pixel. Some of the most commonly used local methods are sum-of-squared differences (SSD) and normalized cross-correlation [Scharstein and Szeliski, 2002]. Local algorithms have the advantage of being very fast and amenable to real-time implementation. However, most local methods have trouble computing disparities in untextured regions (because when looking at a small patch in such regions everything looks similar). Furthermore, local methods tend to compute incorrect disparities at boundary regions.

Global stereo methods attempt to overcome the problems of local methods. Such methods are called global because the depth at a single pixel is dependent on the entire image as global methods usually minimize a cost function over the whole image. The cost function usually consists of two terms - a data term and a smoothing term. The data term measures how well the depth at a pixel agrees with image intensities and the smoothing terms encodes smoothing (and other) assumptions made by the algorithm. Such cost func-

tions also have a Bayesian interpretation (based on Markov Random Fields) given by Geman and Geman [1984]. A variety of optimization methods are used to minimize the cost function.

A well known global method is that by Kolmogorov and Zabih [2001]. They optimize a cost function that in addition to the two terms above, has a term for handling occlusions. Furthermore, they use a fast Graph Cuts based method to minimize the cost function that gives a strong local minima. Other methods use Belief Propagation to minimize the cost function [Sun et al., 2003].

The above algorithms operate on pixels. Recently, algorithms that operate on larger image regions – segments – have become popular [Tao et al., 2001; Hong and Chen, 2004]. The algorithms first segment the image based on color and then find matches based on the segments. These methods have the advantage of shorter computation times as the optimization is done over a smaller set of elements (segments as opposed to pixels). This is relevant as most global methods have a long computation time. In our work we use one local method and two global segmentation based methods as described in Chapter 4.

For a more in depth review of stereo methods we refer the reader to the work by Scharstein and Szeliski [2002]. Furthermore, the paper also links to a website [Scharstein and Szeliski, 2009] where the latest algorithms and a common evaluation can be found.

3.2.2 Depth from Monocular Images

In recent times researchers have looked at inferring the geometry of a scene from a single image. A learning/statistical modeling approach is taken. We describe here three representative methods and discuss their applicability to our work.

Torres-Mendez and Dudek [2004] develop a method for inferring a complete depth map of an image from a partial depth map of the image. They use Markov Random Fields (MRFs) to model the statistical relationship between variations in the intensity values of pixels in the image and variations in the scene depth of those pixels. The MRF is then used

to interpolate the scene depth of the remaining pixels in the image for which depth is not available. In their system they use a laser range-finder to get the partial depthmaps. They test their system in indoor environments and get fairly accurate and complete depth maps.

Saxena et al. [2005] train MRFs to predict the depth of small patches in a single image based on various visual properties of the patch (and its neighbors) such as color and texture at multiple scales. To train the MRFs they use images that have ground truth depth maps (obtained using lasers). They test their method in various environments and are able to get reasonably accurate depth maps. Though the depthmaps do not seem to be as accurate as the ones obtained by the method above, they are able to estimate depth over larger distances.

Hoiem et al. [2007] take a slightly different approach and train a classifier to label various regions of an image into one of three categories: "support, "vertical", or "sky". "vertical" is further subdivided into five categories. The classifier uses various features in an image region, such as color, texture, location in image, shape, lines, and, vanishing points to label the region. The classifier is trained using images that have been manually labeled. To create a 3D model they assume that the ground is a level plane and all regions marked "vertical" are perpendicular to the ground plane. Then all they need to do is find the intersection of the "vertical" regions with the ground plane and extract the camera parameters from the scene. This gives a simple 3D "pop-up" model of the scene [Hoiem et al., 2005].

It seems plausible to use the first two methods to improve the output of stereo methods specially in areas of low texture. Partial depth data available in different parts of the image and learned relationships between image features and depth can help get more complete depth maps. In fact, a similar approach is taken in [Saxena et al., 2007] where the output of a monocular system is shown to improve stereo depth estimates. The third method is similar in flavor to the appearance based methods of Section 3.1.2 in that parts of images are labeled as ground or non-ground. Thus, it is possible to get image-based safety information using this method from a single image.

3.2.3 Structure from Motion

We review here a couple of representative systems for 3D reconstruction that use *Structure from Motion* (SFM) techniques and their relation to our work. SFM techniques require a sequence of images from a moving camera. The camera can be calibrated or uncalibrated. The focus of SFM techniques is to simultaneously estimate the relative poses of the camera for the sequence of images taken and the 3D structure of the environment (usually a sparse 3D reconstruction is obtained).

The general process used in SFM methods is as follows. First, features are detected in the current image and matched to features in the previous image. The matched features are used to compute either a Fundamental matrix (for uncalibrated cameras) or an Essential matrix (for calibrated cameras) relating the two images. Usually a method like RANSAC [Fischler and Bolles, 1981] is used to eliminate bad matches and find the correct matrix. The matrix can then be used to find the relative motion between the two images. Once relative motions between several image pairs are obtained a global optimization over all camera poses and features can be done to obtain the relative poses of all camera locations – this is known as bundle adjustment. The feature locations can be computed in Euclidean or Projective space using the camera poses depending on whether the cameras were calibrated or not, giving the final 3D reconstruction. There are numerous variations of this process, e.g., using three images to estimate a trifocal tensor instead of a Fundamental matrix or using line segments instead of features or the amount of camera calibration information assumed available.

[Fitzgibbon and Zisserman, 1998] use both point and line features and two and three images to obtain a 3D reconstruction. Once a 3D model of points is available, they fit planes to the model using RANSAC iteratively to create a VRML model. In their work they assume that the images be "interesting", i.e., the images have texture. One of the differences between this work and ours is that they use only sparse feature points to construct the 3D models (we use dense stereo depth maps). Also, their models don't catch all the de-

tails that we consider important as the models they construct are from a computer graphics perspective.

[Akbarzadeh et al., 2006] present an SFM based method for reconstructing urban scenes on a large scale using a suite of cameras. Their goal is to provide texture mapped models that can be used by people for various purposes such as route finding. SFM based techniques along with GPS and an Inertial Navigation System (INS) provide accurate camera pose information in their method. Once camera pose is available a multi-view stereo method is used to compute depths maps of the environments. The depth maps from different times are fused and smoothed to produce a triangulated, texture-mapped 3D model. The results are quite good, although like the above method, they are more interested in large scale structure as opposed to finer structure (that is more suitable for robot navigation).

SFM methods are very useful for getting accurate camera (and hence robot) pose estimates and thus can be used in our work for localizing the robot as we discuss below. The 3D reconstructions are usually sparse and hence need to be augmented with denser data such as from stereo. However, the use of line segments for building 3D models can be a useful future extension to our work.

3.3 Methods for Localization

In this section we give a brief overview of visual localization methods. These methods are taken from the robotics SLAM (Simultaneous Localization and Mapping) literature and the vision SFM literature. In our work we assume that the robot has the ability to localize itself in its surroundings. Currently for our robot, localization is done using laser range-finders [Beeson et al., 2006]. In the future we would like to use vision-based localization methods to remove the dependency on laser data. In addition we would like that the localization be done in all 6 degrees-of-freedom available to the robot.

Visual localization methods in the literature use a variety of mathematical techniques to estimate robot pose. Davison [2003] uses an Extended Kalman Filter (EKF) for

localization. The EKF tracks the pose of the robot and observed visual landmarks in the world simultaneously and provides good small scale 6 DOF localization. Nister [2004] uses SFM techniques for real-time 6 DOF visual localization. He introduces a efficient algorithm for computing the relative camera motion between two calibrated images given five pairs of matching points/features. He reconstructs parts of the environment in 3D using the features detected. This method also seems to give good localization results in small scale environments. Sim and Little [2006] use Rao Blackwellized Particle Filters to localize their robot in 3 DOF and construct 3D SIFT feature-based maps.

[Konolige et al., 2008] use features in *stereo* images for localization. They use RANSAC to estimate motion between image pairs and then do incremental bundle adjustment to compute the final visual odometry. They then fuse the visual odometry result with odometry obtained using Inertial Measurement Units (IMUs). The use of IMUs helps reduce error. They demonstrate the system working well over large distances on the order of kilometers. Other methods based on using stereo images include those by Comport et al. [2007], who show their system working over several hundred meters on a car traveling at high speeds, and Howard [2008], who demonstrate similar accuracy over hundreds of meters on legged robots in addition to mobile robots. It is clear that such stereo image based localization methods are particularly relevant to our work.

Chapter 4

Computing Stereo Depth

In this chapter we describe three methods that we use for computing depth maps from stereo images. One of the methods is a commercially available local stereo method and the other two are global methods we propose. The two global methods use color and edge information to compute the depth maps respectively. The reason we consider global methods in addition to the local method is because global methods are considered to provide better depth estimates than local methods in general, particularly in regions of low texture (Section 3.2.1). Therefore, we hope to be able to build better safety maps using global methods than using local methods. In Chapter 8 we evaluate this hypothesis by comparing the safety maps built using all three stereo methods.

In addition to the evaluation done in Chapter 8, we do an *initial* evaluation of all methods towards the end of this chapter using an existing *standard* stereo evaluation framework [Scharstein and Szeliski, 2002] that is popular in the stereo vision literature. The purpose behind this standard evaluation is to provide a basis for comparing the stereo methods we present here with other stereo methods present in the literature [1].

Before describing the stereo methods, we provide a description of stereo camera ge-

[1] We shall see later in Section 4.4, that the results of the standard evaluation are *not* a good indicator of the relative quality of safety maps that can be constructed using the stereo methods.

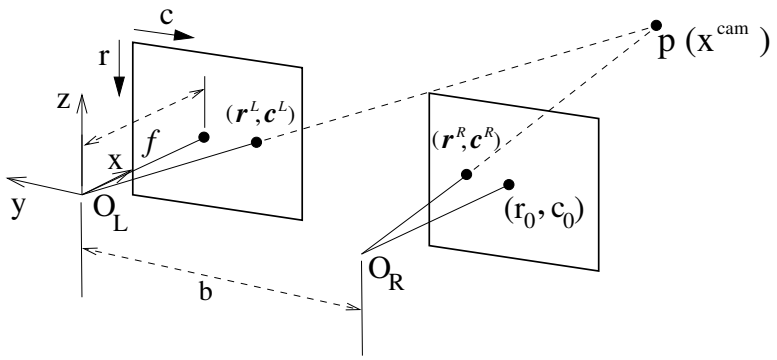


Figure 4.1: Geometry of the left and right stereo imagers observing a point p in the world with camera coordinates $\mathbf{x}^{cam}$.

ometry and the math behind computing depth values from stereo images in the next section.

4.1 Stereo Geometry: Computing Depth and Error

In this section we see how, given the disparity of a point in a stereo image pair, we can compute the distance of the camera to that point. We also compute the errors associated with the distance estimate – information that is used in Chapter 5.

The idealized geometry of a stereo camera is shown in Figure 4.1. The distance between the centers of the two imagers, b, is called the stereo camera's baseline. The focal length of both imagers is f (units are in pixels). (r_0, c_0) are the row and column coordinates of the center of each imager (that happen to be the same for each imager). The camera coordinate frame is attached to the optical point O_L of the left imager.

Consider a point p in the robot's environment that is seen in both imagers. Let the left row and column coordinates of the image point (or pixel) that corresponds to p, be (r^L, c^L), and right coordinates be (r^R, c^R). Then the 3D coordinates, in the camera's

coordinate frame, of the point p are given by [Forsyth and Ponce, 2002]:

$$\mathbf{x}^{cam} = \begin{pmatrix} x^{cam} \\ y^{cam} \\ z^{cam} \end{pmatrix} = \lambda(\mathbf{z}) = \begin{pmatrix} bf/d \\ (c_0 - c^L)\, b/d \\ (r_0 - r^L)\, b/d \end{pmatrix} \tag{4.1}$$

where,

$$d = c^L - c^R \tag{4.2}$$

is the point's disparity and $\mathbf{z} = (r^L, c^L, d)^T$, are defined to be p's image coordinates (*we shall use the abbreviated version* $\mathbf{z} = (r, c, d)^T$ *for denoting a point's image coordinates from here onwards – by default* (r, c) *without superscripts will refer to the left image row and column coordinates*). Thus, given the disparity, the point's 3D location in the camera coordinate frame, $\mathbf{x}^{cam}$, can be computed.

The point's location in the robot's coordinate frame can be computed using the equation,

$$\mathbf{x}^r = \begin{pmatrix} x^r \\ y^r \\ z^r \end{pmatrix} = R_c \mathbf{x}^{cam} + T_c \tag{4.3}$$

where R_c and T_c are known (and constant) rotation and translation matrices, obtained through a calibration process, relating the camera's coordinate frame to the robot frame. Localization gives the position of the robot in the map coordinate frame, $\mathbf{x}_R^m = (x_R^m, y_R^m, \theta_R^m)^T$ [2]. Given this we can also compute the point's position, $\mathbf{x}^m$, in the map coordinate frame:

$$\mathbf{x}^m = \kappa(\mathbf{x}^r, \mathbf{x}_R^m) = \begin{pmatrix} x_R^m + x^r cos(\theta_R^m) - y^r sin(\theta_R^m) \\ y_R^m + x^r sin(\theta_R^m) + y^r cos(\theta_R^m) \\ z^r \end{pmatrix} \tag{4.4}$$

Hence, the 3D location of a point in the map's coordinate frame can be calculated

[2] For simplicity, we assume here that the robot's pose only has three degrees of freedom.

given the robot pose and the point's disparity. The disparity in turn can be estimated using any one of the stereo methods described below. Computing the disparity requires finding image points in the left and right imagers that represent the same world point – this usually involves matching image points in the left imager to points in the right imager. The matches are not perfect and there is a lot of noise associated with the process (the camera resolution also plays a role). The noise or errors can be random or systematic – this is discussed further in Section 4.4.

4.1.1 Error Modeling and Propagation

The random noise or error in the image coordinates, $\mathbf{z}$, of a point p can be modeled by a Gaussian distribution with zero mean [Matthies and Shafer, 1997]. Let $\Sigma = diag(\sigma_r^2, \sigma_c^2, \sigma_d^2)$ be the covariance matrix of the Gaussian distribution in the image coordinate frame. The covariance matrix Σ^{cam}, of the point's position in the camera reference frame is obtained by propagating error using Equation (4.1) [Matthies and Shafer, 1997],

$$\Sigma^{cam} = \left[\frac{\partial \lambda}{\partial \mathbf{z}}\right] \Sigma \left[\frac{\partial \lambda}{\partial \mathbf{z}}\right]^T \tag{4.5}$$

Since localization gives the position of the robot in the map coordinate frame, $\mathbf{x}_R^m$, we can also estimate the covariance Σ^m of the point in the map coordinate frame, using Equations (4.4) and (4.5):

$$\Sigma^m = \left[\frac{\partial \kappa}{\partial \mathbf{x}^{cam}}\right] \Sigma^{cam} \left[\frac{\partial \kappa}{\partial \mathbf{x}^{cam}}\right]^T \tag{4.6}$$

We assume that localization error is negligible in the above Equation.

Thus, the quantity Σ^m gives us an approximation to the error in the location of the world point p as estimated using stereo. We use this information in the data association method described in Chapter 5 for building 3D models.

(a) Left image of stereo pair. (b) Right image of stereo pair. (c) Disparity map with respect to the left image.

Figure 4.2: Stereo image pair and disparity map computed by the local stereo algorithm [Videre Design, 2006]. The stereo images are from the ACESRAMP dataset, one of several datasets that we use in this work.

4.2 Local Correlation Stereo

The local stereo method we use is a standard off-the-shelf multi-scale correlation stereo method [Videre Design, 2006]. Correlation stereo methods work by matching pixels in the left image to pixels in the right image. To find a match for a pixel in the left image, the method compares the intensities of pixels in a small window centered around the original pixel, against the intensities of pixels in a similar window centered around a pixel in the right image. For each pixel in the left image, this computation is performed for several pixels in the right image. The pixel in the right image for which the intensities match best is chosen as the matching pixel.

Figure 4.2 shows the disparity map computed using this method for a stereo image pair obtained from the stereo camera on the intelligent wheelchair. The disparity map shown was obtained after post-processing the disparities obtained from the local stereo method. We removed range readings that were significantly different from neighboring range readings. Such readings have a high likelihood of being incorrect and their removal improved our system's performance.

Computation Time: The local stereo method is fast and can run at frame rate (30 Hz), however the post-processing step slows the speed down to around 10 Hz.

4.3 Global Segmentation Stereo

One of the drawbacks of local correlation stereo methods is their inability to provide good depth estimates in regions of low texture. This is because when intensities in the left and right images are compared locally in low texture regions, several pixels around the correct pixel in the right image appear to match the pixel in the left image, leading to ambiguity and incorrect depth estimates.

Global stereo methods that perform an optimization over the whole image, have been shown to give better depth estimates than local stereo methods [Scharstein and Szeliski, 2002], including in regions of low texture. We draw on a particular type of global method, based on color image segmentation [Hong and Chen, 2004], to develop our own global stereo methods.

The first global method that we propose, described in the following section, utilizes a color-based image segmentation to compute disparity maps [3]. The second method (Section 4.3.2) is the same as the first method in all respects except that it utilizes an edge-based image segmentation algorithm that we propose.

4.3.1 Color Segmentation based Stereo

Color segmentation algorithms work by first segmenting an image into segments of homogeneous color and then computing the disparity of each segment as a whole. Based on the premise that significant disparity discontinuities do not occur inside a region of homogeneous color, most methods fit a disparity plane to each color segment in one stereo image. The quality of the fit is measured by warping the image based on the computed disparities, and comparing the warped image to the remaining stereo image. Since the assumption of planar color segments is not always true, most approaches tend to over-segment the image to better approximate the true disparity [Hong and Chen, 2004]. Over-segmentation

[3] The method we describe here is a variation of a color segmentation stereo method we presented in previous work [Murarka et al., 2008].

allows non-planar surfaces to be approximated using several small planes. We believe that for urban environments, which are mostly composed of planar or smooth surfaces, this approximation should work fairly well in most cases. The final step of global segmentation methods then involves minimizing a global cost function that measures the quality of fit over the entire image and adherence to different constraints (e.g., smoothness).

Algorithm: The main steps in our algorithm are as follows (shown in Figure 4.3).
(i) Color Segmentation. In the first step, the left image (Figure 4.3(a)) of a stereo pair is color-segmented (Figure 4.3(b)) using a color-based segmentation algorithm. We use one of two methods by [Comaniciu and Meer, 2002] and [Felzenszwalb and Huttenlocher, 2004] to color segment the image using code available online from [Georgescu and Christoudias, 2009] and [Felzenszwalb and Huttenlocher, 2007] respectively [4]. In Section 4.4 we present results obtained by using both methods.
(ii) Correlation Stereo Disparity Computation. In the second step, the disparity map for the stereo image pair is computed using a local correlation stereo algorithm (Figure 4.3(c)). Again, we use one of two local algorithms – the algorithm described in Section 4.2 [Videre Design, 2006] and the other a SSD (Sum-of-Squared Differences) correlation stereo algorithm [Scharstein and Szeliski, 2002] that we implement. In Section 4.4 we present results obtained using both methods.
(iii) Fitting Disparity Planes to Segments. The end result of the first two steps is a set of segments and disparities associated with (almost) all pixels in the image. The next step consists of fitting planes in disparity space to the segments. We can define planes P of the form,

$$P: \qquad d = a_1 c + a_2 r + a_3 \tag{4.7}$$

in the 3D space defined by the row and column coordinates of pixels and their disparities.

[4] The parameter settings used were as follows: "SpatialBandwidth = 8; RangeBandwidth = 6; MinimumRegionArea = 80;" for the method by [Comaniciu and Meer, 2002] and "sigma=0.5; K=20; min=100" for the method by [Felzenszwalb and Huttenlocher, 2004].

These planes in disparity space correspond to planes in 3D Euclidean space. For each segment, we fit a disparity plane to the pixels in the segment and their disparities using an iterative weighted least squares method [Hong and Chen, 2004]. This method is able to handle outliers that arise due to bad correlation stereo disparity estimates.

Figure 4.3(d) shows the resulting disparity map found after fitting planes to all segments and using the determined plane parameters to recompute the disparities of all pixels. The dark areas in the figure arise because we do not fit planes to small segments (i.e., segments with less than 100 pixels) or segments with an insufficient number of valid disparities (i.e., at least 50 valid disparities).

The goal of this step is to find a candidate set of planes representative of the major planes in the robot's environment and not, as it seems, to find the best fitting disparity plane for each segment. The following steps describe how the "best" plane is chosen for each segment.

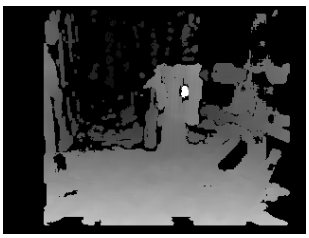
(iv) Adding Additional Planes. In order to ensure that the candidate plane set contains all the major horizontal planes in the robot's environment (horizontal planes are of obvious significance when considering robot navigation), several pre-calculated evenly distributed horizontal planes are added to the candidate set. For example, all horizontal planes between -2 and 2 meters that are 0.1 meters apart are added. This increases the probability that the correct ground (and below ground) planes are included in the set of candidate planes.

(v) Refining the Set of Planes. The plane set, which now contains several similar planes, is refined by clustering similar planes and then picking the plane with maximum *support* in each cluster. This refinement process reduces the number of planes and results in a overall increase in the speed of the algorithm as it reduces the time taken by the final global optimization process.

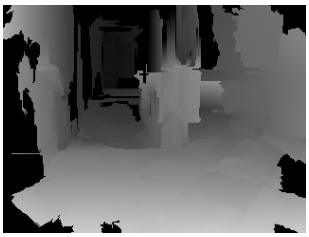
Plane clusters are found very simply: planes that have very similar normals, $\widehat{n}$ (angular difference is less than 0.25 degrees), and similar distances to the origin d_O (less than 0.5 pixels) are grouped in a single cluster. The normal and distance for a plane,

(a) Left image of stereo pair.

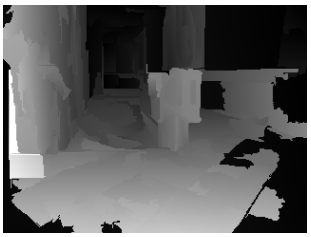
(b) Color segmented left image [Felzenszwalb and Huttenlocher, 2004].



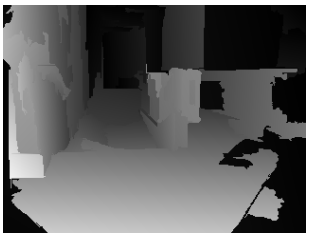
(c) Correlation stereo disparity map [Videre Design, 2006].



(d) Disparity map after fitting planes.



(e) Disparity map corresponding to initial labeling f_0.



(f) Final disparity map after energy optimization.

Figure 4.3: Images from various stages of the color segmentation stereo algorithm.

$P = (a_1, a_2, a_3)^T$, are computed as follows.

$$\widehat{n} = \frac{(a_1, a_2, -1)^T}{(a_1^2 + a_2^2 + 1)^{1/2}} \tag{4.8}$$

$$d_O = \frac{a_3}{(a_1^2 + a_2^2 + 1)^{1/2}}. \tag{4.9}$$

Next, for each disparity plane P and each pixel (r,c) in the image, we compute the following difference in intensities,

$$\Delta I_P(r,c) = |I_L(r,c) - I_R(r,c-d)|. \tag{4.10}$$

where d is computed using Equation (4.7). Here $I_L(r,c)$ and $I_R(r,c)$ are the left and right image intensities at (r,c) (for color images the RGB intensities at each pixel are added to compute $\Delta I_P(r,c)$). Now for each plane P, we find the total number of pixels, for which

$\Delta I_P(r,c)$ is below a threshold T, called the support s_P of the plane in the image,

$$s_P = \sum_{\forall(r,c)} \delta(\Delta I_P(r,c) < T) \tag{4.11}$$

where $\delta(\Delta I_P(r,c) < T)$ is 1 when its argument is true otherwise 0. The best plane in each cluster then is the plane with maximum support. The best planes from each cluster now form the set of candidate planes.

(vi) Energy Minimization for Selecting Correct Planes. In the final step, the set of disparity planes from the candidate plane set, that minimize a global energy function over the two stereo images are chosen as the "correct" planes. To define the energy function we first define a matching cost for a given disparity plane P and segment S,

$$C(S,P) = \sum_{(r,c)\in S} |I_L(r,c) - I_R(r,c-d)| \tag{4.12}$$

where d is computed using Equation (4.7).

We also define a labeling f that assigns a plane to every segment S in the image. For a given labeling f, $f(S)$ then gives the plane corresponding to S. The energy function corresponding to a labeling f of an image is:

$$E(f) = \sum_S C(S, f(S)) + \sum_{S,S'} L_{S,S'} \delta(f(S) \neq f(S')) \tag{4.13}$$

where the first sum is over all segments and the second sum is over all pairs of neighboring segments [5]. $L_{S,S'}$ is proportional to the common boundary length between segments S and S' – computed as the number of boundary pixels of segment S that are adjacent to S' assuming an eight pixel neighborhood (the constant of proportionality is chosen to be 30). $\delta(f(S) \neq f(S'))$ is 1 when its argument is true otherwise 0. Minimizing this energy function, over all segments and all planes in the candidate set, gives us the optimal labeling f for the image

[5] Similar energy functions arise when we consider Markov Random Field based formulations of the stereo matching problem [Li, 1995].

which in turns gives the "correct" disparity plane for each segment S.

The first term of the energy function is often called a data-term – it measures the cost of assigning planes to segments computed on the basis of image intensities only (the "data"). The second term is a smoothing function which takes interactions between neighboring segments into account. It imposes a penalty if two neighboring segments have different disparity planes assigned to them, thereby trying to ensure that disparities change smoothly across the image.

The initial labeling f_0 for starting the energy optimization process is chosen by minimizing $C(S,P)$ for each segment S independently,

$$f_0(S) = arg \min_{\forall P} C(S,P) \tag{4.14}$$

Figure 4.3(e) shows the disparity map for the initial labeling f_0.

Methods for minimizing energy functions of the form given by Equation (4.13), are well documented in literature. In particular, we use an α-expansion graph cuts algorithm to find a strong local minima of the energy function [Boykov et al., 2001; Hong and Chen, 2004] using code available online [Bagon, 2006; Kolmogorov and Zabih, 2004; Boykov and Kolmogorov, 2004]. Figure 4.3(f) shows the final disparity map obtained after finding the "correct" labeling.

Issues: Unfortunately, in practice, the depth estimates obtained from the global color segmentation stereo algorithm are also noisy and contain incorrect range readings. Incorrect range readings can arise due to several reasons – some of the major ones are as follows.

- When the wrong plane is found for a segment. This leads to incorrect disparities for all pixels in that segment.
- When the set of candidate planes does not contain all the planes actually present in the environment. This can happen when the correlation stereo disparities are incorrect or very noisy and the plane fitting process is unable to find the correct planes.

- When the image segmentation is incorrect. This results in segments that straddle depth discontinuities – it means that no single plane can provide the correct set of disparities for that segment.

Therefore, this method is susceptible to producing poor quality disparity maps at times, which means that we need to have a noise removal process, like those discussed in Chapter 5, to properly handle the depth estimates produced by this method

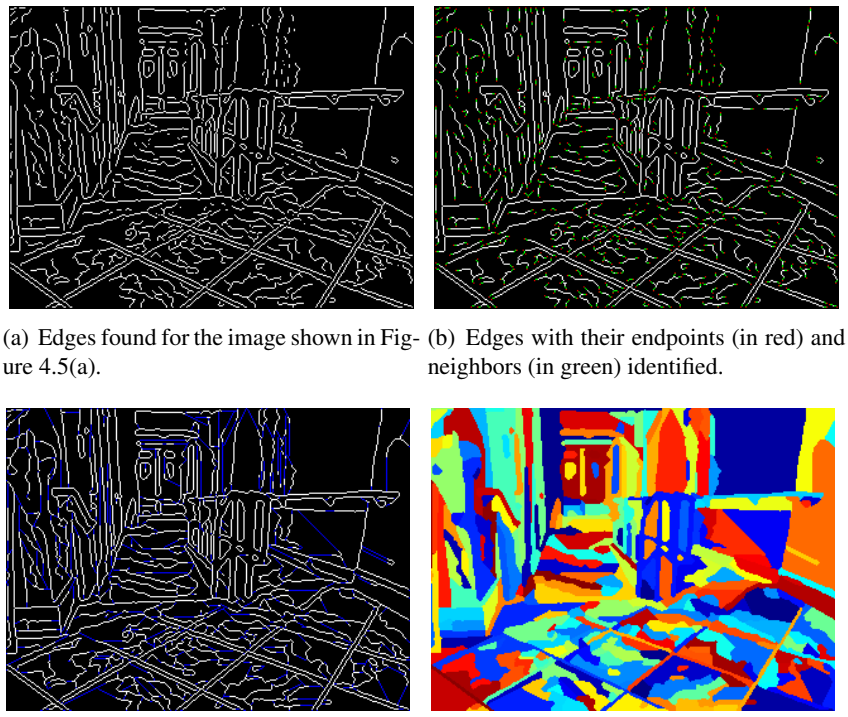
Computation Time: On a machine with a dual core processor, this stereo method takes on average 6 seconds per stereo frame. The code we use is a mix of C++ and Matlab code that is reasonably optimized. We believe that an optimized pure C++ implementation should be able to run at 1 Hz on current dual core processors.

4.3.2 Edge Segmentation based Stereo

As mentioned, the second global stereo method utilizes an edge-based image segmentation algorithm that we propose. The rationale behind using edge-based segmentation is the low amount of local color variation in the urban environments that we consider and a large number of edges. For example in the scene in Figure 4.3(a), several surfaces have the same color even though they are oriented differently. This can be a problem for color segmentation algorithms, in that a segment may straddle regions with different disparity planes. However on the other hand, the environment does have edges where depth discontinuities occur.

Thus, our hypothesis is that by using an edge-based image segmenter, we might be able to overcome the problem of segments straddling depth discontinuities, leading to better overall performance. The rest of the global stereo algorithm stays exactly the same. We describe our proposed edge-based segmentation method next.

Algorithm: The main steps of our edge-based image segmentation algorithm are as follows (shown in Figure 4.4).



(a) Edges found for the image shown in Figure 4.5(a).

(b) Edges with their endpoints (in red) and neighbors (in green) identified.

(c) Edges after growing (grown parts in blue).

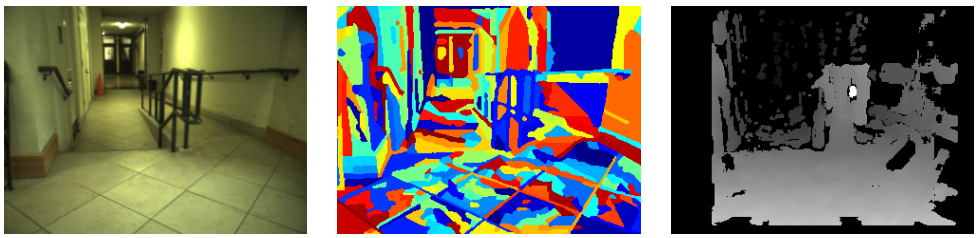
(d) The segments found.

Figure 4.4: Edge based segmentation.

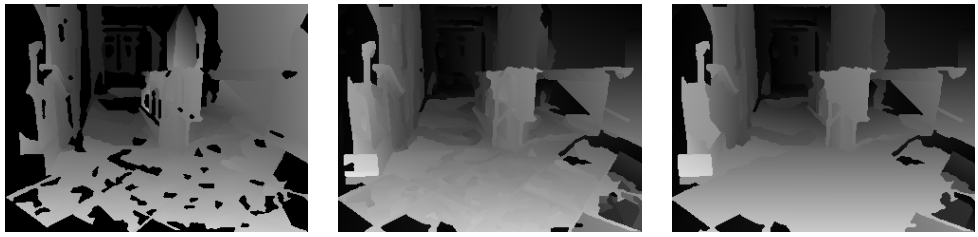
(i) Edge Detection: The first step is detecting edges in the left stereo image. We use the Canny edge detector [Canny, 1986] to find the edges (using code available in Matlab). A low threshold on the edge strength [6] is used so as to detect weak edges as well. Figure 4.4(a) shows the edges found for the stereo image in Figure 4.5(a). Simply finding edges is not enough to get a segmentation since usually the entire edge is not detected. This can lead to segments that straddle depth discontinuities and so we need to "'complete" the edges by growing them before we find segments.

(ii) Finding Edge Endpoints and Neighbors: In the next step, the endpoints of edges are found by tracing the edges. In addition, the first two neighbors of the endpoints are also found. Figure 4.4(b) shows the identified endpoints and neighbors.

[6] This corresponds to parameter settings of "t_high=0.04; t_low=0.4×t_high" in the Matlab code.



(a) Left image of stereo pair. (b) Edge segmented left image. (c) Correlation stereo disparity map [Videre Design, 2006].



(d) Disparity map after fitting planes. (e) Disparity map corresponding to initial labeling f_0. (f) Disparity map after energy optimization.

Figure 4.5: Images from various stages of the edge segmentation stereo algorithm.

(iii) Growing Edges: The endpoint of an edge along with its neighbors helps define the direction, as defined by the vector between the endpoint and its farthest neighbor, in which to grow that end of the edge. All ends of an edge are grown in the respective directions specified by these vectors. Growth is stopped whenever the end of an edge hits another edge or the image boundary. Figure 4.4(c) shows the final edges obtained after growing the edges.

(iv) Finding Contiguous Regions: In the last step all contiguous regions enclosed by the edges are found to give a set of segments. These segments initially don't contain the edges themselves – the edges are randomly assigned to the segments afterwards to give the final image segmentation shown in Figure 4.4(d).

Figure 4.5 shows the results of the global segmentation stereo method when it is based on the edge segmentation algorithm.

Computation Time: On a machine with a dual core processor, the edge-based stereo method takes on average 10 seconds per stereo frame. The increase in the time over the color-based stereo method is largely due to the use of Matlab for implementing the edge-based segmentation algorithm.

4.4 Evaluation and Results

We perform an initial evaluation of all three stereo methods in this section using the stereo evaluation framework presented in [Scharstein and Szeliski, 2002]. The framework has become a standard way for evaluating stereo algorithms in the stereo vision literature. There are two reasons for performing this evaluation: (i) the evaluation provides a common basis for comparing our methods with other stereo methods, and, (ii) the evaluation provides us with a simple way of comparing our stereo methods against each other. A more extensive evaluation of our stereo methods on video datasets, comparing the quality of safety maps produced, is done in Chapter 8. The results of the evaluation in Chapter 8 show that the standard evaluation framework discussed here is not a good predictor of how suitable the stereo methods are for building safety maps (see below and Section 8.2 for more details). Nevertheless, performing the standard evaluation is important for placing our stereo methods in the framework used by the stereo vision community.

The stereo evaluation framework of [Scharstein and Szeliski, 2002] consists of evaluating the methods on several stereo image pairs that have ground truth disparity maps associated with them. The stereo images are shown in Figure 4.6 and their associated ground truth disparity maps are shown in the first row of Figure 4.8 [7]. The disparity maps computed by the various stereo methods for the stereo images are compared against the ground truth disparity maps and three metrics are computed:

[7] The framework evaluates on four image pairs, whereas we evaluate on only three of the four image pairs. The reason for doing this is that one the image pairs is in gray-scale which causes one of the color-segmentation algorithms (external code) we use to crash.

(a) Left stereo image from Tsukuba dataset.

(b) Left stereo image from Venus dataset.

(c) Left stereo image from Sawtooth dataset.

Figure 4.6: Stereo image pairs used in the evaluation framework presented in [Scharstein and Szeliski, 2002]

1. B_{nonocc}: The percentage of pixels in non-occluded regions $\bar{O}$ of the stereo image that have an absolute disparity error greater than 1 pixel.

$$B_{nonocc} = \frac{1}{N_{\bar{O}}} \sum_{(r,c)\in\bar{O}} | \, d(r,c) - d_{GT}(r,c) \, | > 1 \tag{4.15}$$

   Here, $N_{\bar{O}}$ is the total number of pixels in non-occluded regions, $d(r,c)$ is the disparity computed at pixel (r,c) by a stereo method and $d_{GT}(r,c)$ is the ground truth disparity. Figure 4.7(a) shows the non-occluded regions for the one of the test stereo images.

2. B_{untex}: The percentage of pixels in texture-less regions of the stereo image that have an absolute disparity error greater than 1 pixel (defined similarly to B_{nonocc}). Figure 4.7(b) shows the texture-less regions for one of the stereo pairs.

3. B_{disc}: The percentage of pixels in regions with depth discontinuities of the stereo image that have an absolute disparity error greater than 1 pixel (defined similarly to B_{nonocc}). Figure 4.7(c) shows the regions with discontinuities for one of the stereo pairs.

We evaluate and compare four different versions of our stereo methods. In addition the results of our stereo methods are compared against the results of the state-of-art color segmentation based stereo method presented in [Hong and Chen, 2004]. The algorithms we

(a) Non-occluded regions (in white).

(b) Texture-less regions (in white).

(c) Regions with depth discontinuities (in white).

Figure 4.7: Various regions for the Tsukuba dataset [Scharstein and Szeliski, 2002]. Notice that there is a boundary region around all images where the stereo algorithms are not evaluated since the results of many stereo algorithms are considered unreliable in boundary regions.

evaluate are as follows.

- **CS:** The correlation stereo method from Section 4.2.
- **HC:** The algorithm by [Hong and Chen, 2004]. The results presented here are taken from their paper – they use the same evaluation framework that we use and the same images.
- **SS-Best:** A version of the color segmentation stereo method from Section 4.3.1. It uses the color segmentation algorithm presented in [Comaniciu and Meer, 2002] and our SSD based correlation stereo algorithm. We have tuned this version to work best on the three stereo image pairs that form part of the evaluation of [Scharstein and Szeliski, 2002] (Figure 4.6).
- **SS-Used:** A version of the color segmentation stereo method from Section 4.3.1 that uses the color segmentation algorithm presented in [Felzenszwalb and Huttenlocher, 2004] and the correlation stereo algorithm presented in Section 4.2. This is the version we use when we evaluate on our video datasets in Chapter 8. This version has been tuned on one of our video datasets - it is not tuned on the stereo images we test on here. Unfortunately, due to the difference in the quality of the images obtained

from our stereo camera from the ones used here, we have to use a different version of the color segmentation algorithm when evaluating on our video datasets. *The SS-Best method does not work with all of our video datasets giving very poor results for some*.

- **ES-Used**: A version of the edge segmentation-based stereo method from Section 4.3.2. This is the version we use when we evaluate on our video datasets in Chapter 8 and is tuned similarly to SS-Used.

Table 4.1 gives the values of the three metrics obtained for the five stereo methods and Figure 4.8 shows the disparity maps found by the stereo methods. We make the following observations.

- From Table 4.1 we see that all three versions of the segmentation stereo methods we propose are able to get much better performance than the correlation stereo method. This is true for untextured regions as well, providing some justification of our original reason for using global stereo methods.

- The SS-Best algorithm gets good performance in non-occluded and untextured regions. While the performance of HC is clearly better than that of SS-Best, the absolute errors for SS-Best are quite low (between 2 and 7 percent) in non-occluded and untextured regions. However, the performance of SS-Best is not as good in regions with depth discontinuities.

- The SS-Used and ES-Used algorithms perform reasonably well, but not as well as SS-Best. This is expected because the methods are not tuned for these datasets.

- The disparity maps in Figure 4.8 for SS-Best also suggest that it performs quite well, except in regions with depth discontinuities (in agreement with Table 4.1).

- The disparity maps for the correlation stereo method tell a slightly different story. The method seems to perform better than what the metrics in Table 4.1 suggest. It

Dataset →	Tsukuba			Venus			Sawtooth		
Metric →	B_{nonocc}	B_{untex}	B_{disc}	B_{nonocc}	B_{untex}	B_{disc}	B_{nonocc}	B_{untex}	B_{disc}
Algo. ↓									
CS	26.34	30.10	35.64	28.56	51.68	29.56	18.22	29.59	26.16
HC	1.23	0.29	6.94	0.08	0.01	1.39	0.30	0.00	3.24
SS-Best	6.44	3.73	19.64	3.58	5.53	17.12	2.87	3.47	12.30
SS-Used	10.97	10.92	26.64	7.65	11.23	23.90	7.92	8.45	21.39
ES-Used	11.91	12.77	25.74	10.43	18.13	21.23	7.04	14.81	16.14

Table 4.1: Three error metrics for five different stereo algorithms evaluated on the three stereo image pairs in Figure 4.6. CS denotes the correlation stereo algorithm from Section 4.2; HC denotes the algorithm by [Hong and Chen, 2004]; SS denotes the color segmentation stereo algorithm from Section 4.3.1; and ES denotes the edge segmentation algorithm from Section 4.3.2.

loses out in the quantitative evaluation partly because of not estimating disparities in many regions, specially around the boundary. In regions where it does estimate disparities, it does fairly well.

- The CS method also clearly shows a "fattening" effect near the boundaries of objects in the disparity maps meaning it does not estimate disparities near boundaries well.

- Looking at the disparity maps of Figure 4.8, we can see that there are two kinds of error (or noise) present in the disparities computed using the stereo methods - random error and systematic error. Systematic errors manifest themselves as disparities that are completely incorrect, whereas random error is present in (almost) all other disparities. In Chapter 5 we discuss how to handle both kinds of error (or noise) to build good 3D models.

Based on the above evaluation we can conclude that the segmentation stereo methods perform better than the correlation stereo method on the stereo image pairs in Figure 4.6 and are likely to perform better than the correlation stereo method at building safety maps. However, our evaluation in Chapter 8 does not support this. The correlation stereo method performs very well at building safety maps and actually better than the segmentation stereo methods in one crucial metric (the False Negative error rate in Section 8.2). Why does this

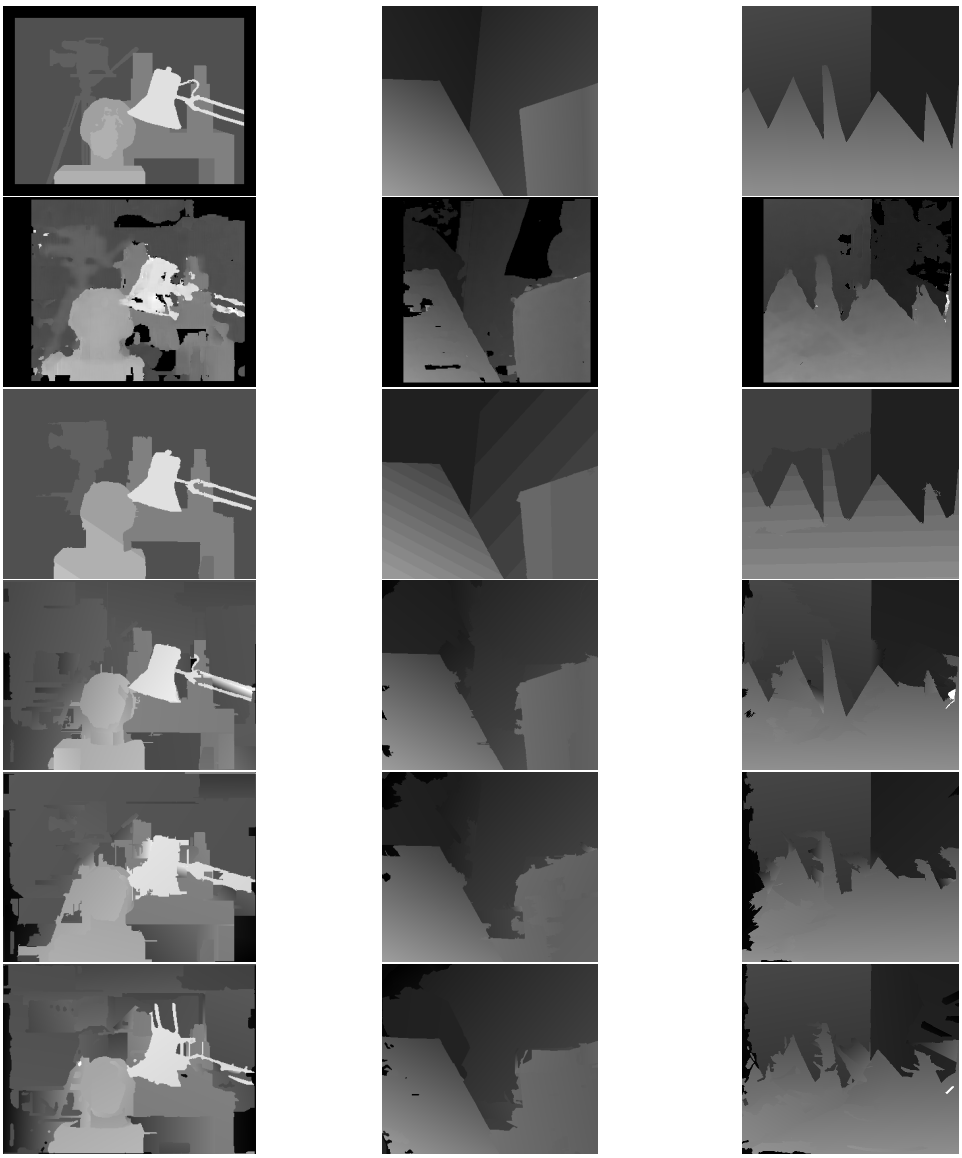


Figure 4.8: Disparity maps for three different data sets. The disparity maps for each row are computed by the same algorithm. Row 1 shows ground truth disparity maps. Row 2 shows maps for the correlation stereo method, CS. Row 3 shows maps from HC [Hong and Chen, 2004]. Row 4 and 5 shows maps for the color segmentation stereo methods, SS-Best and SS-Used respectively. Row 6 shows maps for the edge segmentation method, ES-Used.

happen? We believe there are several reasons.

First of all, the three metrics B_{nonocc}, B_{untex}, B_{disc} are not very suitable for the purpose of evaluating stereo methods for use in robot mapping. The metrics in effect only measure errors in disparities upto a resolution of 1 pixel. For our stereo camera, a 1 pixel error in disparity at a distance of 4 meters (corresponding to a disparity of approximately 6 pixels) leads to an error of 0.6 meters or more in the depth estimate. Clearly, such an error is unacceptable when building maps for robot navigation. To get more sensitive metrics we need stereo images with more accurate ground truth disparity maps.

Secondly, a problem with the evaluation framework of [Scharstein and Szeliski, 2002] is the very small number of images the stereo methods are evaluated on. With careful tuning, and given the low resolution of the metrics, it is possible to get good performance out of a stereo method on the datasets – thus not truly reflecting the stereo methods performance under a wider variety of situations. For example, the texture-less regions in the stereo image pairs in Figure 4.6 do not reflect the kinds of texture-less environments a robot in urban environments has to deal with. If we compare Figure 4.2(a) to Figure 4.6(a) we see that the environment in Figure 4.2(a) has much less texture. This is reflected in the disparity maps (Figure 4.2(c) and Figure 4.8) computed by the correlation stereo method for both images where the disparity map in Figure 4.2(c) has more blacked out regions where no good disparities are estimated.

Thirdly, in the framework the methods are tested on the image pairs on which they are tuned. Thus, there is no separation of training and testing data that can lead to stereo methods that have been overfit to the datasets and do not really generalize.

In our video datasets in Chapter 8, a stereo method runs on thousands of images meaning that in practice we get a great variation in the quality of the disparity maps computed. This is especially true for the segmentation-based stereo methods because it is quite likely any one of the issues with such methods, mentioned in Section 4.3.1, can prevail leading to noisy disparities. Additionally, we tune all our algorithms on only about a dozen

images from one single video dataset and test on all the datasets, providing a better test of the generalization capabilities of the stereo methods.

Therefore, we need to be careful when interpreting the results from the evaluation framework of [Scharstein and Szeliski, 2002]. The main problem with setting up a stereo evaluation framework is that it is quite hard to get good ground truth stereo data leading to low resolution metrics and few datasets.

4.5 Related Work and Summary

Related work on stereo methods is discussed in Section 3.2.1 in Chapter 3.

In this chapter we have presented three different stereo methods for computing depth maps - one existing local method and two global methods that we propose. The color segmentation-based global stereo method works by first segmenting images into regions of (almost) uniform color. Disparity planes are then fit to the segments and a global objective function minimized to get the final disparity map. The edge segmentation-based stereo method works in a similar manner except that the image is segmented using edges present in the image.

We perform an initial evaluation of all three methods using the standard evaluation framework of [Scharstein and Szeliski, 2002]. The results suggest that better safety maps can be built using global stereo methods, however, our results in Chapter 8 show otherwise. We believe this happens due to several reasons, amongst them the low "resolution" of the error metrics and the very small number of stereo images on which the methods are evaluated in the framework of [Scharstein and Szeliski, 2002].

Once the depth maps have been computed for a stereo image pair, we use the equations in Section 4.1 to obtain the 3D locations in the map coordinate frame of stereo points corresponding to pixels in the left stereo image. We also compute the error in the 3D point location estimates.

Chapter 5

Building 3D Models

In this chapter we describe two methods that build 3D models from stereo range data by accumulating depth information over time. The data has to be accumulated in an intelligent manner to build good quality models. The reason for this, as discussed in the previous chapter, is that depth estimates from stereo are noisy. As mentioned in Section 4.4, the estimated depth of a pixel in a stereo image can have noise in one of two forms:

- The estimated depth can be in the ballpark of the pixel's true depth. Such depth estimates can be modeled as arising due to the addition of zero mean random Gaussian noise to the true pixel depth (Section 4.1.1).
- The estimated depth can be completely incorrect. We call such depth estimates, *false positive* depths or range readings and they can be thought of as having noise produced due to systematic errors.

It is important that both types of noise in the estimated depth be handled, specially the latter, since these lead to incorrect models of the environment being built that can place the robot in danger or hinder its motion.

Accumulating range data over time and filtering out some of the data helps attenuate the effects of both types of stereo noise. Also, since stereo data is usually sparse in regions

of low texture, gathering data also allows the robot to model such low texture regions adequately over time.

The two methods for building 3D models that we discuss in the chapter are both probabilistic. The first method, discussed in Section 5.1, is based on associating points in 3D Euclidean space over time [1]. The second method (Section 5.2) is based on occupancy grids that are well established in the robotics mapping literature. *The models that we construct are 3D point clouds with a 3D grid overlaid on top*. The points in the model are all in a single frame of reference – the map coordinate frame.

An important aspect of the 3D models is that they are *local* models. The models are of a bounded size ensuring that computation is bounded. There are two ways in which we implement local 3D models. In the first method, the model is always (approximately) centered on the robot and "scrolls" with the robot's motion. As the robot moves, newer regions of the environment fall in the area enclosed by the model, and other regions fall out. Models of the regions that fall out, are lost and have to reconstructed from scratch when the robot sees them again. The other method for constructing local maps is useful when the robot arrives at a *place* or is in a fixed environment like a room. In such cases, the model does not "scroll" with the robot. Instead the robot moves in the model as it explores its environment. Implementing both types of local models requires that separate coordinate frames be maintained for the robot and the model.

5.1 Probabilistic Data Association of Points in 3D Space

The probabilistic data association method we introduce here takes as input at every frame a collection of points in 3D space outputted by the stereo method. Each *stereo point* should have associated with it an estimate of the error in its location. In addition, the point may also *optionally* have a *visual feature vector* associated with it, e.g., a point may have associated with it a vector containing the RGB values of the image pixel that corresponds to it.

[1] The method described here was first presented in [Murarka et al., 2006].

In the method, every stereo point is assumed to have been generated by a (point) *landmark* in the world whose true location is unknown. When the robot first wakes up in the world, it does not know of any landmarks, and so all 3D points from the first stereo image frame are used to instantiate a set of *temporary landmarks*. Points from the following stereo frames are compared to the temporary landmarks. If a match, based on some criteria, is established between a point and a landmark, the landmark's location is updated using the point's location. The temporary landmarks that match points consistently over many frames are made into *permanent landmarks*. All the permanent landmarks that exist at a particular time are taken together to give the 3D point cloud model at that time. A grid is then overlaid on top of point cloud to give the final 3D model (Section 5.1.4). Thus, to build a map of landmarks, four steps (described in the following sections) are performed at every frame:

1. Associations/matches are determined between the points observed in the current stereo frame and existing (temporary or permanent) landmarks. The criteria used for determining matches are the similarities of the point and landmark locations and the visual feature vectors associated with both. Points that do not match any existing landmarks are used to initialize new temporary landmarks.

2. Existing landmark location estimates are then updated based on the observed locations of the matching points. The feature vectors associated with the landmarks are also updated.

3. Temporary landmarks that have not had sufficient matches in a fixed period of time, are removed from the landmark database. Such landmarks are considered to correspond to false positive range readings. On the other hand, temporary landmarks that match a minimum number of points in the time period are made permanent.

4. The collection of permanent landmarks obtained at the end of the above steps gives the 3D point cloud model. A grid is overlaid on top of this point cloud to give the final 3D model.

5.1.1 Associating Points with Landmarks

The problem is to pick from all stereo points observed at time t, the point that has the highest probability of being associated with landmark l_i, based on the point and landmark locations and visual features. This has to be done for every single existing temporary and permanent landmark that is currently in the field of view of the robot.

Let the location of a stereo point, p_j, observed in the current time step (or frame) t, be $\mathbf{x}^t_{p_j}$ in the map reference frame[2] (obtained from Equation (4.4)). Let the covariance (or error) in its 3D position in the map reference frame be, $\Sigma^t_{p_j}$ (Equation (4.6)). Let $\mathbf{d}^t_{p_j}$ be the visual feature vector associated with point p_j at time t.

Let the *true* location of a landmark, l_i, in the map frame of reference, be $\mathbf{x}_{l_i}$. Let $\widehat{\mathbf{x}}^t_{l_i}$ be the landmark's location *estimate* at time t. The estimate is modeled using a Gaussian probability distribution, $\widehat{\mathbf{x}}^t_{l_i} = \{\mu^t_{l_i}, \Sigma^t_{l_i}\}$. Each landmark also maintains a database of visual feature vectors consisting of the feature vectors of all points that have matched the landmark in the past (including the feature vector of the point used to instantiate the landmark). Associated with each feature vector in the database is the original viewing direction of its corresponding point from the robot's camera. The feature vector, $\mathbf{d}^t_{l_i}$, of the landmark l_i at time t, is taken to be the feature vector in the database whose viewing direction is closest to the current viewing direction of the landmark (we explain how the viewing direction is computed and the database updated in Section 5.1.2). This ensures a certain amount of viewpoint invariance for the feature vectors.

Each stereo frame generates on the order of thousands of stereo points, and matching each point to every landmark can take a long time. To reduce computation time, for each landmark l_i, an initial set of points, $P_i = \{p_1, p_2, ..., p_{n_i}\}$, is selected from all the stereo points using a fast process. This initial set can be obtained in various ways, for example by collecting all points within a cuboidal region around the landmark. Since we assume that each point can be associated with exactly one landmark, let $a_i \in P_i$ denote the point with

[2] We drop the superscript m in this section as all locations are in the map frame of reference unless otherwise stated.

which landmark l_i is associated. The point p^* most likely to be associated with l_i is then given by,

$$p^* = \arg\max_{[p_j \in P_i]} p(a_i = p_j \mid I_X, I_D) \tag{5.1}$$

where, $I_X = \{\widehat{\mathbf{x}}_{l_i}^{t-1}, \{\mathbf{x}_{p_k}^t\}_{k=1..n_i}\}$, is the collection of landmark and point locations, and $I_D = \{\mathbf{d}_{l_i}^t, \{\mathbf{d}_{p_k}^t\}_{k=1..n_i}\}$, is the collection of landmark and point feature vectors.

Derivation [3]**.** We now show how Equation (5.1) can be simplified into a usable form. Using Bayes rule and making an independence assumption between locations and visual features we get,

$$\begin{aligned} p(a_i = p_j \mid I_X, I_D) &\propto p(I_X, I_D \mid a_i = p_j)\; p(a_i = p_j) \\ &\propto p(I_X \mid a_i = p_j)\; p(I_D \mid a_i = p_j)\; p(a_i = p_j) \\ \textit{or}\quad p(a_i = p_j \mid I_X, I_D) &\propto p(I_X \mid a_i = p_j)\; p(I_D \mid a_i = p_j)\; . \end{aligned} \tag{5.2}$$

Where we get the final equation by assuming a uniform prior for $p(a_i = p_j)$ for all $p_j \in P_i$ and noting that its value will not matter when taking the *arg max* in Equation (5.1). The first term on the RHS in Equation (5.2) can be simplified as follows using the rules of probability,

$$\begin{aligned} p(I_X | a_i = p_j) &= p(\widehat{\mathbf{x}}_{l_i}^{t-1}, \{\mathbf{x}_{p_k}^t\}_{k=1..n_i} \mid a_i = p_j) \\ &= p(\mathbf{x}_{p_j}^t \mid \widehat{\mathbf{x}}_{l_i}^{t-1}, \{\mathbf{x}_{p_k}^t\}_{k=1..n_i,\; k\neq j}, a_i = p_j) \cdot \\ &\quad p(\widehat{\mathbf{x}}_{l_i}^{t-1} \mid \{\mathbf{x}_{p_k}^t\}_{k=1..n_i, k\neq j},\; a_i = p_j) \cdot\; p(\{\mathbf{x}_{p_k}^t\}_{k=1..n_i,\; k\neq j} \mid a_i = p_j) \\ &= p(\mathbf{x}_{p_j}^t \mid \widehat{\mathbf{x}}_{l_i}^{t-1},\; a_i = p_j) \cdot\; p(\widehat{\mathbf{x}}_{l_i}^{t-1}) \cdot \prod_{\substack{k=1..n_i \\ k\neq j}} p(\mathbf{x}_{p_k}^t) \end{aligned} \tag{5.3}$$

$$p(I_X | a_i = p_j) \propto p(\mathbf{x}_{p_j}^t \mid \widehat{\mathbf{x}}_{l_i}^{t-1},\; a_i = p_j)\; . \tag{5.4}$$

[3]The reader can skip directly to the final result in Equation (5.10) if desired.

Where we assume that the locations of the points are independent of each other to get the first term and third terms in (5.3), and also that the locations of points in the current frame, that are not associated with l_i, do not affect its estimated location in the past frame, to get the second term in (5.3). To get the final equation, we note that the value of the second term in (5.3) is common across all probability terms in (5.1), and so is the value of the third term under the assumption that $p(\mathbf{x}^t_{p_k})$ is uniform $\forall k$, making these terms irrelevant when finding the *arg max*.

The RHS of (5.4) can be further evaluated by marginalizing over the true landmark location $\mathbf{x}_{l_i}$,

$$p(\mathbf{x}^t_{p_j}|\widehat{\mathbf{x}}^{t-1}_{l_i}, a_i = p_j) = \int p(\mathbf{x}^t_{p_j} \mid \mathbf{x}_{l_i}, \widehat{\mathbf{x}}^{t-1}_{l_i}, a_i = p_j)\; p(\mathbf{x}_{l_i} \mid \widehat{\mathbf{x}}^{t-1}_{l_i}, a_i = p_j)\; d\mathbf{x}_{l_i} \quad (5.5)$$

$$= \int N_{\mathbf{x}^t_{p_j}}(\mathbf{x}_{l_i}, \Sigma^t_{p_j})\; N_{\mathbf{x}_{l_i}}(\mu^{t-1}_{l_i}, \Sigma^{t-1}_{l_i})\; d\mathbf{x}_{l_i} \quad (5.6)$$

$$p(\mathbf{x}^t_{p_j}|\widehat{\mathbf{x}}^{t-1}_{l_i}, a_i = p_j) \propto exp\{-\frac{1}{2}(\mathbf{x}^t_{p_j} - \mu^{t-1}_{l_i})^T(\Sigma^t_{p_j} + \Sigma^{t-1}_{l_i})^{-1}(\mathbf{x}^t_{p_j} - \mu^{t-1}_{l_i})\} \quad (5.7)$$

The left term in the integral in (5.5) is a generative model. It is the probability distribution over the observed point locations given the true location of the landmark. For a Gaussian generative model it reduces as shown. The right term under the integral in (5.5) is the probability distribution over the true location given the parameters of the estimated Gaussian distribution and reduces to the estimated Gaussian distribution itself. To go from Equation (5.6) to (5.7) we use standard formulae [Bishop, 2006]. Combining equations (5.4) and (5.7) we get,

$$p(I_X \mid a_i = p_j) \propto exp\{-\frac{1}{2}(\mathbf{x}^t_{p_j} - \mu^{t-1}_{l_i})^T(\Sigma^t_{p_j} + \Sigma^{t-1}_{l_i})^{-1}(\mathbf{x}^t_{p_j} - \mu^{t-1}_{l_i})\} \quad (5.8)$$

We proceed in a similar fashion for the second term on the RHS in (5.2) to get,

$$p(I_D \mid a_i = p_j) \propto exp\{-\frac{1}{2}(\mathbf{d}^t_{p_j} - \mathbf{d}^t_{l_i})^T\Sigma_d^{-1}(\mathbf{d}^t_{p_j} - \mathbf{d}^t_{l_i})\} \quad (5.9)$$

where Σ_d is taken to be a diagonal matrix containing a set of constant parameters that have to be tuned based on the types of visual features that are being compared.

Finding the Best Match. If we combine equations (5.1), (5.2), (5.8), and (5.9) and take the negative log, we get that the point, p^*, most likely to be associated with l_i, is the one with the lowest negative log likelihood.

$$p^* = \arg\min_{[p_j \in P_i]} \quad (\mathbf{x}^t_{p_j} - \mu^{t-1}_{l_i})^T (\Sigma^t_{p_j} + \Sigma^{t-1}_{l_i})^{-1} (\mathbf{x}^t_{p_j} - \mu^{t-1}_{l_i}) + (\mathbf{d}^t_{p_j} - \mathbf{d}^t_{l_i})^T \Sigma_d^{-1} (\mathbf{d}^t_{p_j} - \mathbf{d}^t_{l_i}) \; . \tag{5.10}$$

The first term on the RHS is the square of the Mahalanobis distance between the observed point location at t and estimated landmark location at $t-1$. This implies that points with a lower Mahalanobis distance to the landmark will be favored. The second term, which is also the square of a Mahalanobis distance, ensures that points with visual features similar to that of the landmark are favored.

Since, matches for each landmark are found independently, occasionally we can get the same point matching more than one landmark. In such cases we pick the match with the lower Mahalanobis distance (ties are broken randomly).

No Matches and New Landmarks. To consider the possibility of no point being associated with landmark l_i, we set a threshold, M_{max}, on the maximum Mahalanobis distance allowed between only the *locations* of a point and a landmark. To set the threshold, we think of the Mahalanobis distance as the Euclidean distance measured in units of standard deviation [Duda et al., 2000], and set the maximum distance, in standard deviation units, that a point can be from a landmark in order for a match to be considered (the threshold can be found by looking up a χ^2 distribution table) [4]. An added benefit is that points, that do not lie within the Mahalanobis distance threshold of *any* landmark, are used to initialize

[4] We use $M_{max} = 3.66$ for all our experiments.

new landmarks.

5.1.2 Updating Landmark Locations and Feature Vector Database

We use a Kalman filter [Gelb, 1974], to update a landmark's location estimate based on the observed location of its matching point. This helps reduce the amount of error in the landmark location estimate. Given that point p_j is associated with l_i at time t, the landmark location estimate from $t-1$ can be updated as follows:

$$\begin{aligned}(\Sigma_{l_i}^t)^{-1} &= (\Sigma_{l_i}^{t-1})^{-1} + (\Sigma_{p_j}^t)^{-1} \\ \mu_{l_i}^t &= \Sigma_{l_i}^t \left((\Sigma_{l_i}^{t-1})^{-1} \mu_{l_i}^{t-1} + (\Sigma_{p_j}^t)^{-1} \mathbf{x}_{p_j}^t \right)\end{aligned} \tag{5.11}$$

When a point is used to first initialize a landmark, the landmark's location and covariance are simply taken to be the point's location and covariance.

The landmark's feature database is updated by adding to it the matching point's feature vector $\mathbf{d}_{p_j}^t$ and the viewing direction to the point from the robot's camera. The viewing direction is taken to be the vector from the camera's optical point, O_L (Figure 4.1), to the point's location, $\mathbf{x}_{p_j}^t$. To find the coordinates of the camera's optical point in the map coordinate frame we set $\mathbf{x}^{cam}$ to $(0,0,0)^T$ in Equation (4.3) and use Equation (4.4). The feature vector in the database, with the closest viewing direction to the landmark's current viewing direction is the one whose viewing direction vector has the smallest angle to the landmark's viewing direction. This feature vector is used as the landmark's feature vector, $\mathbf{d}_{l_i}^t$, at time t.

5.1.3 Identifying Permanent and False Positive Landmarks

As mentioned, newly initialized landmarks are temporary to begin with. Only after a minimum number of points, *cmin*, match the landmark within a given number of frames, $fmax$,

of the landmark being initialized, is the landmark made permanent [5]. If enough matches are not found, the landmark is removed. This removes false positive range readings from the set of observed points.

The parameter *cmin* is of importance as it controls the amount of noise in the 3D model. A high value of *cmin* means that a temporary landmark needs to have several matches before it will be made into a permanent landmark – so setting a high value for *cmin* will lead to false positive range readings being removed aggressively. However, this can also lead to correct range readings being removed as well which can be dangerous to the robot. In Chapter 8 we measure the effect of *cmin* on the quality of the safety maps constructed.

5.1.4 Overlaying a 3D Grid on the Point Cloud

At the end of performing all the above updates for a given stereo image frame at time t, we get a set of permanent landmarks along with an estimate of their locations. A 3D grid is overlaid on top of this cloud of point landmarks. All voxels in the grid in which any permanent landmark falls are marked as occupied and all other voxels are marked as being free.

The length of a voxel's side is denoted as l_v and is taken to be 0.1m in our experiments. We determine the indices of the voxel $(u,v,w)^T$ in which a point $(x,y,z)^T$ falls as follows,

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} [x/l_v] \\ [y/l_v] \\ [z/l_v] \end{pmatrix} \tag{5.12}$$

where $[\cdot]$ represents a rounding operator.

Figure 5.3 shows an example of overlaying a grid on top of the landmarks. *The set of permanent landmarks at time t and the grid together form the 3D model available to the*

[5] In the experiments reported in Chapter 8 we let $fmax = cmin$ and only change the value of *cmin*.

robot at that time.

5.1.5 Results and Discussion

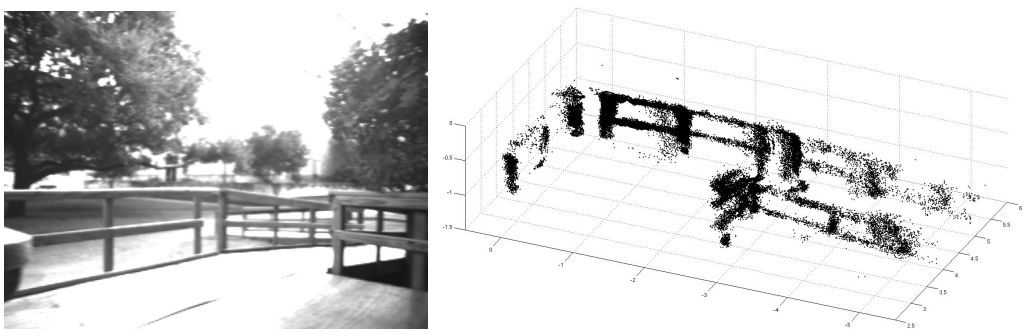
In this section we discuss some properties of the probabilistic data association method and present qualitative results. Quantitative results and comparison with the occupancy grid method for building 3D models are presented in Chapter 8.

Figure 5.1 shows a picture of an outdoor wheelchair ramp and a 3D reconstruction of the ramp's railing using the probabilistic data association method. Qualitatively, the 3D reconstruction obtained is quite good. Figure 5.2 shows another example of the quality of 3D reconstructions obtained. The figure also demonstrates the advantage of using 3D stereo information over 2D laser range data. Figure 5.3 shows the effect of overlaying a 3D grid on top of the cloud of point landmarks.

As can be seen from all the figures, the probabilistic data association method is very effective at removing false positive range readings. The readings are effectively removed by the two constraints of minimum number of required point matches, *cmin*, and maximum Mahalanobis distance, M_{max} between locations. Not using either of these, particularly the Mahalanobis metric, leads to poor maps. Figure 5.4 shows what happens when, instead of the Mahalanobis distance, standard Euclidean distance is used as a metric (the stereo maps shown the figure are made by stereo points obtained using SIFT features as explained in the figure). Both figures show a laser generated map of the lab overlaid with a map built using stereo. In the stereo map on the left, false positives are eliminated which results in the visual landmarks lining up well with the laser map. In the map on the right, Euclidean distance fails to eliminate false positives leading to an unusable stereo map [6].

The Mahalanobis distance proves to be a very effective metric for building 3D models even when we only use point locations and no visual features. The stereo map shown on the left in Figure 5.4 was made using both location information and visual feature vec-

[6] It is possible to use a tighter threshold when working with Euclidean distance but then this leads to a map with few landmarks that is not adequate for safety.



(a) (b)

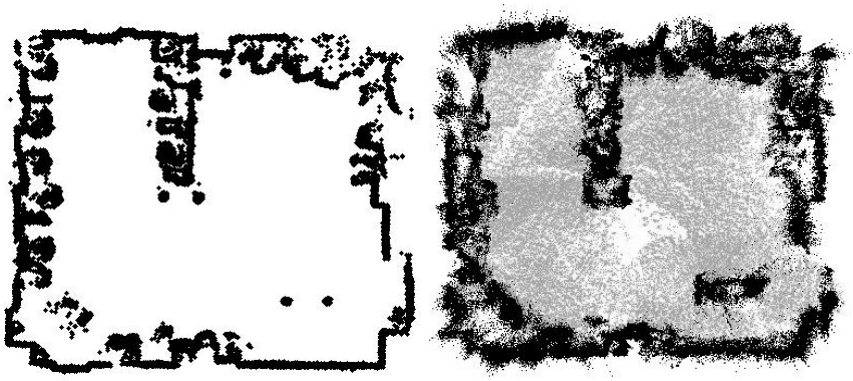
Figure 5.1: (a) A wheelchair ramp with a wooden railing (this is a sample image from our TAYLORRAMP dataset). (b) 3D point cloud of the ramp railing constructed using the probabilistic data association method. The correlation stereo method of Section 4.2 was used for computing the stereo range readings.

tors consisting of *scale* and *orientation* information associated with SIFT features [Lowe, 2004]. However, when we used only location information to build the map we got very similar results. In fact the 3D models shown in Figures 5.1, 5.2, and, 5.3 were all built using only location information.

To conclude, the probabilistic data association method presented here, successfully gets rid off most false positive range readings and also reduces the effect of random noise in the remaining range readings. A nice feature of the method is that it combines the use of point locations and visual features for matching into a single framework. An added advantage of the method is that it works with both sparse (Figure 5.4) and dense stereo range data (Figures 5.1 to 5.3) unlike the occupancy grid method that we present next which only works well with dense stereo data.

Computation Time. The average computation time taken by the data association method (excluding the time taken for overlaying the grid) for a region about $8m \times 8m \times 3m$ in size is 1.4 seconds per frame for an unoptimized C++ implementation. We believe that an optimized version can run at a cycle rate of 2-3 Hz.

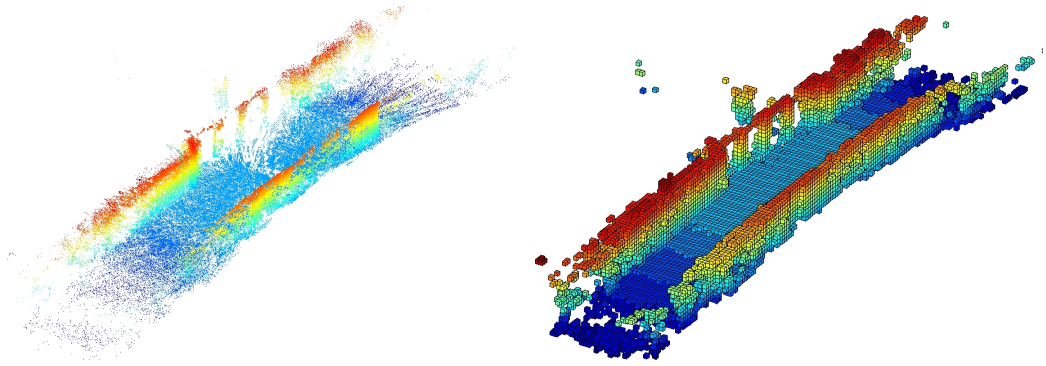
(a)



(b) (c)

(d) (e)

Figure 5.2: (a) A sample image from our TAYLORLAB dataset. Note that there is a table in the image to the right. (b) A 2D laser map of the lab with obstacles shown in black and clear areas in white. (c) Top view of a 3D landmark map of the same environment constructed using the probabilistic data association method and stereo range information (Section 4.2). In addition to containing all obstacles (in black), the stereo map also explicitly represents the ground plane (in grey), unlike the laser map where clear areas are assumed to represent flat ground. (d) and (e) Zoomed in views of the table in image *(a)*, as represented in the laser map (only the table legs are seen) and in the stereo map (almost the entire table is visible), showing another advantage of using 3D stereo data over 2D laser data.



(a) (b)

Figure 5.3: (a) 3D point landmark model of a long corridor style environment with two wheelchair ramps at each end (this is a model of the BIORAMP environment shown in Figure 8.1(b)). The points are color coded according to height. (b) The 3D grid model overlaid on top of the point cloud model. All voxels in which a landmark falls are shown.

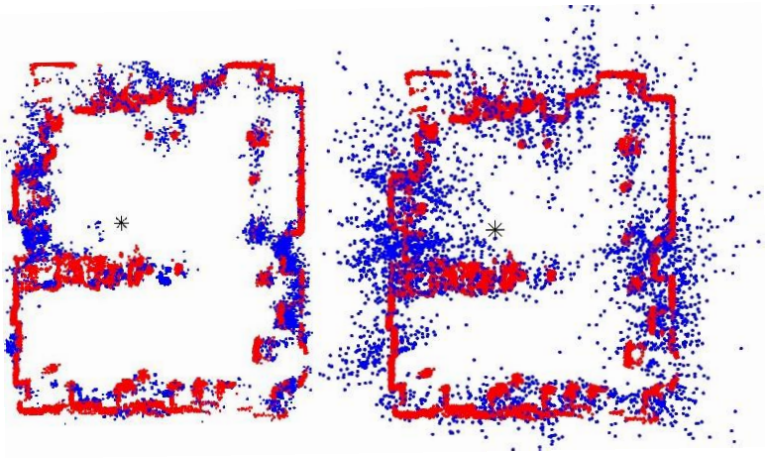


Figure 5.4: Left: Laser map (red) overlaid with a stereo map (blue dots) made using Mahalanobis distance. Right: Laser map overlaid with stereo map made with Euclidean distance. The Mahalanobis distance is much more effective at removing false positive range readings than Euclidean distance. The stereo maps shown here were made using SIFT features [Lowe, 2004]. SIFT features are detected in a stereo image pair and their disparities computed by matching features in the left image to the right image. This gives a set of range readings that are then used in the probabilistic data association method to give the stereo maps shown. The feature vector associated with each SIFT stereo point consisted of the *scale* and *orientation* of the SIFT point [Lowe, 2004].

5.2 3D Occupancy Grid

In this section we briefly discuss our implementation of an occupancy grid based method for building 3D models. Our method is a variation of the method presented in [Konolige, 1997] and in addition is used to build a 3D grid instead of a 2D grid.

Let the stereo range reading corresponding to a point, p_j, observed in the current time step (or frame), be $r_j = D_j$, where D_j is the distance from the camera to the point p_j (D_j is computed as the Euclidean-norm of p_j's camera coordinates obtained from Equation (4.1)). Let v be a voxel in the 3D occupancy grid and let $occ(v)$ denote the proposition that voxel v is occupied. We denote by $O(occ(v))$ the log odds probability that v is occupied, defined as,

$$O(occ(v)) = log\ \frac{p(occ(v))}{p(\neg occ(v))}\ . \tag{5.13}$$

We are interested in finding the log odds probability, $O(occ(v) \mid r_j = D_j)$, that voxel v is occupied given range reading $r_j = D_j$. It can be shown that this log odds probability is given by [Konolige, 1997],

$$O(occ(v) \mid r_j = D_j) = \lambda(r_j = D_j \mid occ(v)) \ + \ O(occ(v)) \tag{5.14}$$

where,

$$\lambda(r_j = D_j \mid occ(v)) = \frac{p(r_j = D_j \mid occ(v))}{p(r_j = D_j \mid \neg occ(v))}\ . \tag{5.15}$$

is the sensor model and is called the likelihood ratio. The numerator in the likelihood ratio, $p(r_j = D_j \mid occ(v))$, measures the probability of a range reading, $r_j = D_j$, being produced when voxel v is occupied. Similarly, the denominator, $p(r_j = D_j \mid \neg occ(v))$, measures the probability of a range reading, $r_j = D_j$, being produced when voxel v is not occupied. So, for a range reading that is produced in the neighborhood of a voxel, we expect the numerator to have a large value and the denominator to have a small value, giving a large value for the likelihood ratio. Similarly, for a range reading produced far away from a voxel, we expect

the likelihood ratio to have a small value.

Sensor Model. Based on the above, we use a simple sensor model as follows. For each range reading, r_j, a ray is cast from the camera to the 3D point, p_j. Only voxels along the ray have their occupancy probabilities affected by the range reading. The likelihood ratio, for a voxel in which the range reading ends (i.e., the voxel in which point p_j falls), will be a high positive value, denoted by $incr_1$, and is going lead to an increase in the log odds probability of occupancy of the voxel. For voxels just before and just after the above voxel we also expect the likelihood ratio to have a high (but slightly lower than $incr_1$) positive value, denoted by $incr_2$. For all other voxels, between the camera and the above voxels, we expect the log odds probability of occupancy to decrease as they are likely to be unoccupied. The likelihood ratio is negative for such voxels and is denoted by $decr$.

Occupancy Grid Update. We use Equation (5.14) to update the 3D occupancy grid given the current set of range readings. From the equation we get that for each range reading, the log odds probability of a voxel v being occupied, given that range reading, is simply the sum of the likelihood ratio of the range reading and the prior log odds probability of the occupancy of the voxel. We start with the assumption that the occupancy of all voxels is unknown in the beginning, and so the prior probability of occupancy is 0.5 leading to a log odds probability of $O(occ(v)) = 0$. Then for each range reading, r_j at the current time, a ray is cast from the camera to the 3D point, p_j, and voxels along the ray have their log odds probability of occupancy $O(occ(v) \mid r_j = D_j)$ updated by adding to their prior log odds probability the value of the likelihood ratio as discussed above. At the next time step, the value of $O(occ(v) \mid r_j = D_j)$ calculated in the current time step, is used as the value of the prior log odds probability $O(occ(v))$.

Tuning the occupancy grid algorithm amounts to tuning the values of the increment parameters, $incr_1$ and $incr_2$, and the decrement parameter $decr$ [7]. In Chapter 8 we examine

[7] We have manually tuned the sensor model parameters in this work. Ideally, we would like to learn the

the effect of different decrement values ($decr \in \{-1, -3, -5\}$) on the quality of the 3D models constructed for fixed increment values (we use $incr_1 = 10$ and $incr_2 = 4$).

Dealing with Correlated Stereo Data. One of the extensions that we make to the occupancy grid algorithm is to deal with the fact that occupancy grid algorithms assume that range readings are independent, whereas stereo range readings are highly correlated. Since stereo returns a large number of range readings, several readings can fall in a single voxel and so to reduce the effect of correlation we update the grid for only one range reading per voxel per timestep (we just use the first range reading that falls in a voxel). All other range readings that fall in the voxel are discarded. This reduces the effect of correlation because range readings that fall in a voxel are from neighboring rays (and hence neighboring pixels) and discarding some approximates the independence assumption. Furthermore, we also discard entire stereo frames when the robot is stationary since range readings across such frames are highly correlated.

Point Database. In addition to the occupancy grid, a point database is also updated at each frame. For each voxel a list of the points corresponding to the range readings that fall in the voxel over time is maintained. The list is of a fixed size and at each frame the list is updated with the current range point that falls in the voxel. Each voxel's list is ordered according to the distance of the camera from the point the moment it was seen. This way when the list is pruned, points seen from closer distances are given preference over other points since closer points have lower error. This point database corresponds to the landmark database created by the probabilistic data association method.

The final 3D model at a given time then consists of the occupied voxels in the 3D occupancy grid and the list of sorted points associated with each occupied voxel (voxels that have a log odds probability of occupancy greater than a certain threshold $occ_t = 100$

parameters and even the whole model from the data [Stronger and Stone, 2008]. This is a direction for future work.

are considered occupied).

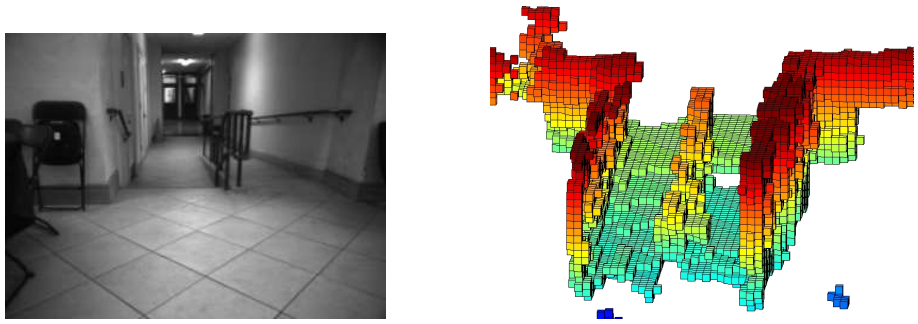
5.2.1 Results and Discussion

Figure 5.5 shows an occupancy grid model of our ACESRAMP dataset obtained after processing 459 stereo frames. The figure shows how ramps and drop-offs appear in the 3D grid models we create. Ramps are discretized into a set of long low steps whereas a drop-off edge is characterized by unoccupied or unknown voxels near it (although this is not always the case). Low drop-offs can also appear as steps, similar to the ramp, and hence can even be mistaken for a ramp – we show this happening for the ACES2302 environment in Chapter 8. This makes detecting drop-offs and ramps and distinguishing them from stairs hard. Figure 5.6 shows the occupancy grid created for the BIORAMP dataset and also the corresponding 3D point database that is maintained with the occupancy grid.

Computation Time. The average computation time taken by the occupancy grid method for a region about $8m \times 8m \times 3m$ in size is 1 second per frame for an optimized Matlab implementation. A real-time C++ version of the algorithm gives a cycle rate of 4-9 Hz.

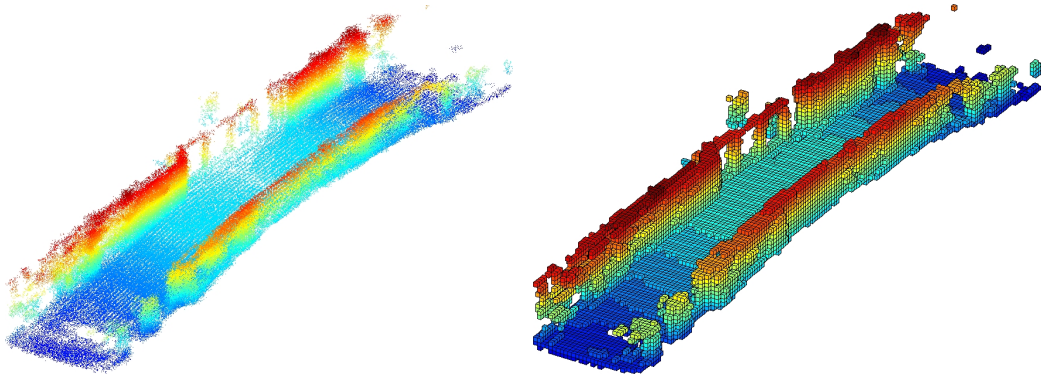
5.3 Related Work

The probabilistic data association method for building 3D models introduced here draws on several earlier pieces of work. The idea of using Kalman filters to track the locations of visual features is well established, e.g., in the localization method of Harris [1993], Kalman filters are used to track 3D points in the environment. Likewise the Mahalanobis distance has been used in landmark based SLAM techniques [Dissanayake et al., 2001]. Probabilistic methods for solving data association problems have been around for some time, particularly in the tracking literature. For example, Reid [1979] solves the data association problem associated with tracking multiple moving objects such as airplanes using radar (and other sensors) by maintaining multiple hypotheses and computing their likelihoods.



(a) (b)

Figure 5.5: (a) Sample image from the ACESRAMP dataset with a ramp to the right and a drop-off to the left of a railing. (b) 3D occupancy grid of the environment showing all occupied voxels, colored according to height (produced after processing 459 stereo frames in the dataset). The grid is shown from a viewpoint different from viewpoint of the image. The image is obtained by looking down the ramp whereas the grid image is obtained by looking from the other direction, up the ramp. This is done to show how the ramp and drop-off appear in a 3D grid. The ramp is seen as a set of long steps whereas the drop-off is characterized by unoccupied voxels near it.



(a) (b)

Figure 5.6: (a) Point cloud and (b) grid model of the BIORAMP environment shown in Figure 8.1(b) produced using the occupancy grid method.

The main points that distinguish our probabilistic data association method from the other methods are: (i) the principled combination of landmark locations and visual feature vectors into a single framework; (ii) the ability to handle large numbers of landmarks; and

(iii) the ability to handle many false positive landmarks. These properties of our method, particularly the last two, allow our method to be used for tracking dense stereo data points. The other methods, described above, have been developed for tracking sparse landmarks, e.g., corner features in robot mapping, or for tracking objects such as airplanes which are also very sparsely distributed. Furthermore, the incidence of false positives is much lower in such problems.

5.4 Summary

In this chapter we presented two methods for building 3D models from stereo range data. The 3D models consist of a 3D point cloud with a 3D grid overlaid on top. Both methods for building the 3D models are able to handle noisy and false positive stereo range readings.

The first method, which we propose, is a probabilistic data association method that uses the Mahalanobis distance between point locations and visual features to find matches between stereo points across several images. This helps in getting rid of false positive range readings and allows good quality 3D models to be built, as the results show. The results also show the effectiveness of using the Mahalanobis metric as opposed to Euclidean distance at finding correct matches. One of the important characteristics of the data association method is its ability to handle a large number of densely packed landmarks.

The second method we present is based on occupancy grids. The qualitative results for this method show that it also builds good quality 3D models from stereo range data. In Chapter 8 we evaluate and compare both methods based on the quality of the safety maps constructed using them.

Chapter 6

Safety Analysis of 3D Models

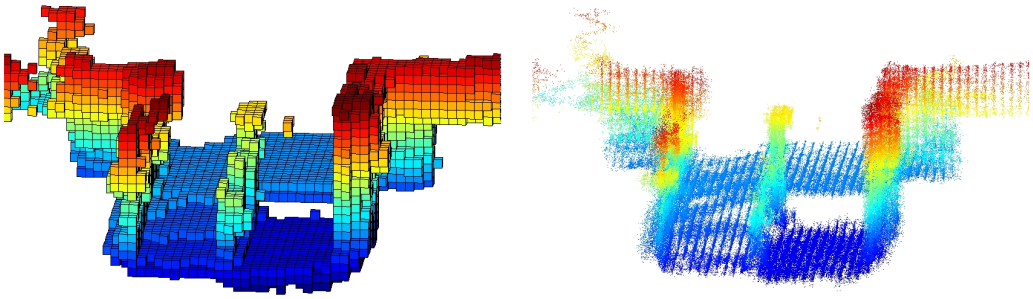
In this chapter we present ways of analyzing 3D models for safety. Figure 6.1 shows an example of a 3D model, consisting of a 3D grid and a 3D point cloud, built for the ACES-RAMP environment after several hundred stereo frames have been processed. We introduce three methods of analyzing such models in order to construct local safety maps of the environment. The methods, in increasing order of complexity, are as follows.

1. Safety Analysis based on Height Thresholds: Simple height thresholds on the 3D grid are used to create safety maps.

2. Safety Analysis based on Grid Traversability: A more sophisticated analysis of the 3D grid is done that considers the reachability of regions from the robot's current location. This process also results in a segmentation of traversable ground regions in the grid.

3. Safety Analysis based on 3D Planes: Planes are fit to the 3D point cloud utilizing the above segmentation. The planes and their relationships with neighboring planes are analyzed to create the safety maps [1].

[1] The second and third methods described here have also been presented in [Murarka and Kuipers, 2009b].



(a) (b)

Figure 6.1: 3D model of the ACESRAMP environment (shown in Figure 5.5(a)) built after the first 459 stereo frames of the dataset have been processed. On the left is the 3D grid and on the right is the 3D point cloud model. The voxel size in the grid shown is $0.1m \times 0.1m \times 0.1m$. This model was built using the occupancy grid method of Chapter 5 and the correlation stereo method of Chapter 4. Note that the environment has a ramp on the left and a drop-off edge on the right in the models shown. Also note the discretization imposed on the ramp by the 3D grid model. The 3D model shown here has been truncated by the planning radius discussed in Section 6.1 so as to allow easier comparison with the rest of the figures in this chapter. The full 3D grid part of the model is shown in Figure 5.5.

In addition to constructing the local safety maps, we also construct hybrid 3D models. Such models are a hybrid of a 3D grid and 3D planes. The grid is used to represent obstacles and other non-ground regions in the world whereas the planes are used to represent traversable ground regions in the world. The planes are annotated as being either "Level" or "Inclined" (the hybrid 3D models can be thought of as 3D versions of the local safety maps).

Notation. We introduce a bit of notation to assist us in explaining the algorithms in this chapter. The grid part of the 3D model in Figure 6.1(a) consists of voxels indexed using the variables u, v, and, w along the x, y, and z directions (with the z direction along the vertical) respectively (see Section 5.1.4). Thus a voxel is addressed as (u,v,w) and, a column of voxels is addressed as (u,v). The height of a voxel is taken to be the index w (also referred to as the *indexed height* of the voxel). Let the location of the ground above which the robot is currently located be w_g. The ground height is known since we know the robot's pose

and height. The robot's height is denoted by w_r and can be computed by dividing the z-coordinate of the position of the highest point on the robot by the length of a voxel's side l_v (see Equation (5.12)). The voxel column on which the robot is located is denoted as (u_r, v_r).

6.1 Safety Analysis based on Height Thresholds

We begin with a simple scheme for building safety maps using 3D models. The method labels regions that are within a certain height of the ground plane as safe and labels the rest as unsafe or unknown. The safety map thus created is called a **threshold-based safety map** (to distinguish it from the other two safety maps created in this chapter). The purpose of this analysis here is to show the drawbacks associated with a simple analysis of the 3D models and provide a base case to which we can compare the two methods presented later in the chapter [2].

Height Map. The first step in building a safety map using this method, is the creation of a 2D *height map*. The height map is a 2D grid with the same dimensions as the 3D grid along the x and y axes. Each cell $c = (u, v)$ in the 2D grid contains the height w_c of the highest occupied voxel in the corresponding voxel column (u, v). Only occupied voxels in the 3D grid that are below the robot's height w_r are considered. This is because the robot can travel under occupied voxels that are higher than it – such as voxels corresponding to the roof. Such occupied voxels higher than the robot's height are called *overhanging voxels*. In later sections we will also refer to $w_c(u, v)$ as the height of voxel column (u, v).

Creating the Threshold-based Safety Map. The safety map is also a 2D grid with the same dimensions as the height map. Let L denote the label attached to cell $c = (u, v)$ in the safety map. All cells in the safety map are initialized with the label "Unknown". The cells are then re-labeled as follows based on the height w_c of the corresponding cell $c = (u, v)$ in

[2] In addition we use the safety maps created by this method to illustrate certain points in Chapter 7.

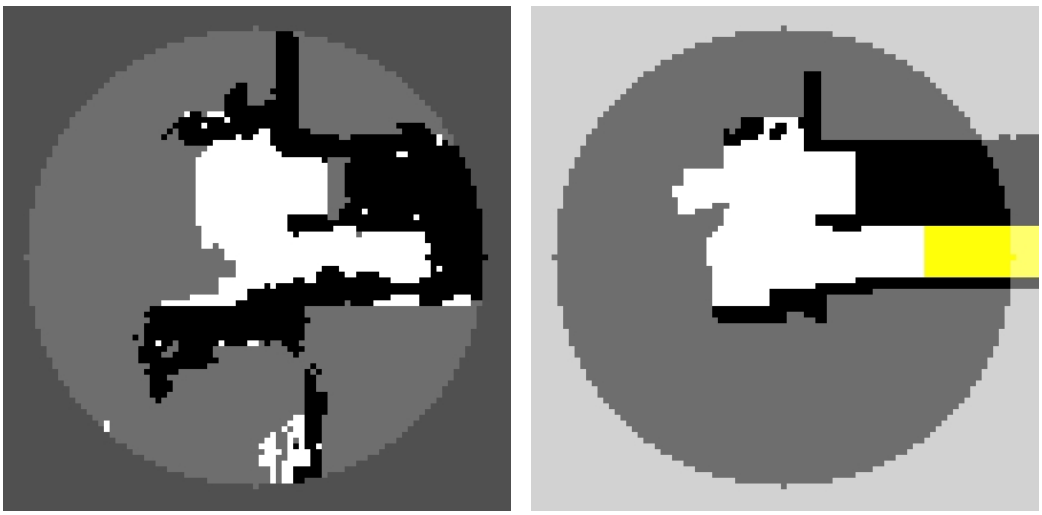
the height map,

$$L(u,v) = \begin{cases} \text{Level} & \text{if } -w_T \leq w_c(u,v) \leq w_T \\ \text{Non-ground} & \text{if } w_c(u,v) < -w_T \text{ or } w_c(u,v) > w_T \\ \text{Unexamined} & \text{if } [(u-u_r)^2 + (v-v_r)^2]^{1/2} > R_{plan} \end{cases}$$

where w_T is a threshold on the maximum and minimum height of traversable ground regions and thus determines the cells that should be considered "Level" and safe. The last label "Unexamined" is applied to all cells in the safety map that are more than a certain distance, R_{plan} (called the *planning radius*), away from the robot's current location (u_r, v_r). This is because stereo returns good range data only within a limited distance and considering cells far away results in many errors (see Section 8.3.2 for an experimental justification). We use a planning radius of 4 meters in our implementation (this corresponds to $R_{plan} = 40$ for a grid size of 0.1 meters).

The threshold-based safety map thus created labels all cells corresponding to voxel columns in the 3D grid that are within a certain vertical distance of the known ground (where the robot is currently located), as "Level". All other voxels columns that are much too high, e.g., those representing walls, or too low, e.g., those representing a downward staircase, have their corresponding cells in the safety map marked as "Non-ground" and hence unsafe.

Results. Figure 6.2 shows a threshold-based safety map for the ACESRAMP environment and also the corresponding ground truth safety map. As can be seen, the ramp in the environment gets incorrectly labeled. The part of the ramp that is within the height threshold of the ground (where the robot is currently located) is marked as "Level" whereas the part of the ramp that goes below the height threshold is marked "Non-ground". Hence this simple scheme for creating local safety maps can only work in environments where the ground is horizontal and there are no inclines.



(a) (b)

Figure 6.2: (a) **Threshold-based safety map** corresponding to the 3D grid of the ACES-RAMP environment (Figure 6.1(a)). Regions in white are "Level" and safe whereas regions in black are marked "Non-ground" and hence unsafe. Grey areas inside the circle are "Unknown" and regions outside the circle are labeled "Unexamined". The robot is located at the center of the circle (not shown). (b) **Ground truth safety map** with the portion corresponding to the safety map highlighted. Note that half of the region corresponding to the ramp in the threshold-based safety map gets marked as being safe and the other half gets marked as being unsafe. This is the main disadvantage in using such a simple scheme - inclined surfaces cannot be handled.

6.2 Safety Analysis based on Grid Traversability

In the second method for creating safety maps, a more sophisticated analysis is performed on the grid part of the 3D model. Voxels in the 3D grid are classified based on their height and reachability from the robot's current location into traversable ground regions and non-ground regions and a safety map is constructed using the classification. The traversable ground regions are identified by a process in which the grid is segmented into regions with the same height. We call the safety map constructed using this method a **traversability-based safety map**.

Algorithm 1 Finding Traversable Ground Segments in a 3D Grid

Require: 3D grid (with occupied voxels marked) and the robot's location in the grid.

1: Construct a height map as described in Section 6.1.
2: Initialize $c = (u_r, v_r)$ – the cell in the height map where the robot is currently located.
3: $k \leftarrow 1$.
4: Create an empty list L_U. The list will contain cells that have to be examined.
5: Create a new segment S_k and add to it the voxel column corresponding to cell c.
6: **while** A cell $c_i \in S_k$ can be found whose height has not yet been compared to that of its neighbors **do**
7: Let w_{c_i} be cell c_i's height.
8: Find the list of cells, N_c, neighboring c_i.
9: **for each** $c_n \in N_c$ **do**
10: Let w_{c_n} be cell c_n's height.
11: **if** $w_{c_n} = w_{c_i}$ **then**
12: Add the voxel column corresponding to c_n to segment S_k.
13: **else if** $|w_{c_n} - w_{c_i}| \leq w_T (= 1)$ **then**
14: Add c_n to L_U. c_n's height is not that different from c_i's and hence it can lead to the creation of a new segment.
15: **end if**
16: **end for**
17: Mark c_i as a cell whose height has been compared to that of its neighbors.
18: **end while**
19: **if** $L_U \neq \phi$ **then**
20: Pop L_U (i.e., remove cells from L_U) until a cell c not yet part of any segment is found. If no such cell is found: **Return** the list of segments S_k $\forall k$ found.
21: $k \leftarrow k + 1$
22: Goto step 5.
23: **else**
24: **Return** the list of segments S_k $\forall k$ found.
25: **end if**

Segmentation of the 3D Grid. Segmentation of the 3D grid into regions with the same height is achieved as explained in Algorithm 1. Figure 6.3(a) shows the segments obtained for the ACESRAMP 3D grid. Each segment consists of a set of connected voxel columns that have the same height. The segments are found using the height map since each cell (u, v) in the height map contains the height w_c of the corresponding voxel column. Starting from the cell (u_r, v_r) in the height map where the robot is currently located, the cell's neigh-

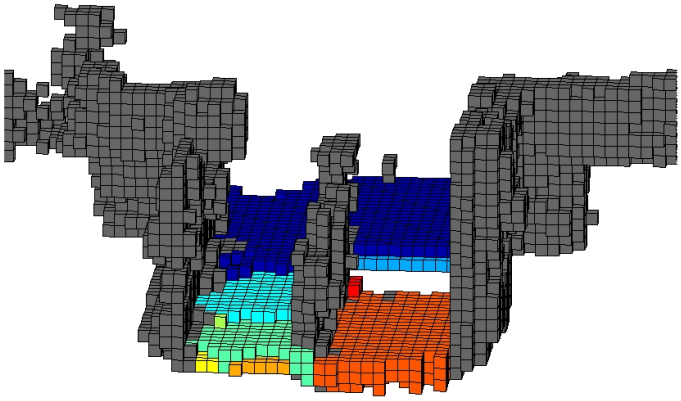
bors with the same height $w_c(u_r, v_r)$ as the cell, are added to the first segment. Then cells with the same height neighboring the cells that have already been added to the first segment are added and process is carried on recursively until no more cells can be found to add to the first segment.

Next, cells neighboring the first segment's cells that are within a certain height w_T of $w_c(u_r, v_r)$, and not equal to w_c, are used to initialize new segments (these segments are then the neighbors of the first segment). All the cells belonging to the new segments are found (cells part of each segment are connected and have the same height) and the process is continued recursively by finding neighboring segments of the existing segments (such that the neighboring segments have heights within w_T of the existing segment heights), until all segments in the 3D grid have been found.

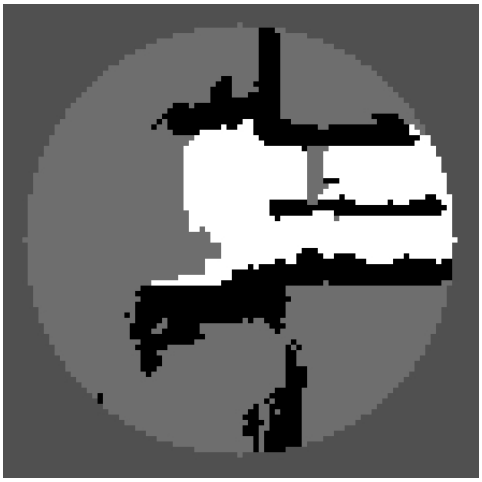
Note that we can think of the segments as both 3D and 2D entities. As 3D entities the segments can be thought of as consisting of voxel columns, and as 2D entities they can be thought of as consisting of cells (corresponding to the voxel columns).

The use of the height threshold w_T above ensures that the height difference from one segment to the next is not much allowing a mobile robot to travel from one segment to another. As Figure 6.1(a) shows, the grid based representation discretizes inclined surfaces in the world, forcing us to use a threshold like w_T to determine if it is possible for the robot to go from one segment to the next.

Creating the Traversability-based Safety Map. All the cells and the corresponding voxel columns that are part of the segments are considered to be traversable ground regions. All remaining cells and corresponding voxel columns that are not part of any segments are either outside the height threshold (and thus obstacles) or not reachable from the robot's current location and are considered to be non-ground regions. The traversability-based safety map captures this information: traversable ground cells are marked "Level" and are safe and non-ground cells are marked "Non-ground" and considered unsafe. Cells corresponding to voxel columns with no occupied voxels are marked "Unknown". Cells outside the planning



(a)



(b)

Figure 6.3: (a) **Grid segmentation.** Segments (coded by color) produced for the the 3D grid (Figure 6.1(a)) of the ACESRAMP environment. Each segment consists of connected voxel columns with each column having the same height (as given by the height map). (b) **Traversability-based safety map.** The color scheme is the same as that used for the threshold-based safety map in Figure 6.2(a). The region corresponding to the ramp gets correctly identified as being safe this time – although it gets labeled as "Level". The below-ground region on the other side of the railing is incorrectly classified as being safe (see text for more details).

radius are marked "Unexamined" as done for the threshold-based safety map.

Results. Figure 6.3(b) shows the traversability-based safety map for the ACESRAMP 3D grid model. As can be seen, this safety map correctly identifies the ramp as being traversable unlike the threshold-based safety map. However, the traversability-based safety map cannot identify the region corresponding to the ramp as being an incline because that region might as well be a series of steps with little height difference between the steps. Thus, the traversability-based safety map cannot distinguish between inclined regions and steps. As a result we label all traversable ground regions as "Level" in this safety map.

The below-ground region on the other side of the railing, as the ramp, is incorrectly identified as being reachable [3] via the ramp even though there is a railing between the ramp and the region. This happens because of two reasons: (i) the top of the railing is not modeled completely in the 3D model causing the regions between the vertical rails to appear as free space, and, (ii) the height difference between the ramp segment and the below-ground segment is within the height threshold w_T making the below-ground appear reachable. Figure 6.4(b) later in the chapter shows that there is actually a fairly significant height difference between the two regions and so the below-ground region is in fact not reachable by the robot. This case gets handled correctly when we fit planes to the 3D grid.

Computation Time. The average computation time for segmenting the 3D grid and creating the traversability-based safety map is around 250ms for a 3D grid about $8m \times 8m \times 3m$ in size for unoptimized code written in *MATLAB*. For C++ code, the computation time is 4ms for a 3D grid of size $10m \times 10m \times 3m$ on a 1.83 GHz dual core CPU.

Using Mean Height. We describe here a slight variation on the method for segmenting the 3D grid. In addition to computing the height of voxel columns as described in Section 6.1, we can compute another type of height for each voxel column that we refer to as the *mean height* of the voxel column. The mean height, m, of a voxel column, (u,v), is the average

[3] The below ground region is actually reachable via the ramp in the environment. However, given the robot's limited planning radius the robot cannot (and should not) infer the below-ground region as being reachable.

value of the z-coordinate of all points (from the 3D point cloud) that are associated with the voxel column. The variation consists of replacing step 14 in Algorithm 1 with the following steps:

1. **If** $[m_n/l_v] = [m_i/l_v]$ **then**
2. Add the voxel column corresponding to c_n to segment S_k.
3. **else**
4. Add c_n to L_U. c_n's height is not that different from c_i's and hence it can lead to the creation of a new segment.
5. **end for**

Here m_n and m_i are the mean heights of the voxel columns corresponding to cells c_n and c_i respectively and l_v is the voxel size. $[m_n/l_v]$ gives the mean height in terms of the grid indices where $[\cdot]$ is a rounding operator.

The use of the mean height helps in finding larger sized segments. It is an extra check, done in addition to that done using the indexed heights w, for making sure that cell c_n is assigned to the correct segment. The indexed height can be incorrect sometimes because of stereo noise or lack of range data that leads to regions where the cells or voxel columns have different indexed heights although ideally they should all have the same height. However, even if the indexed height is different, it is possible that the mean height of the voxel columns is the same. Thus, whenever the indexed height of two voxel columns does not differ by much (Step 13 in Algorithm 1) we compare the mean heights to make sure the two columns indeed belong to different segments.

We use the mean height in our real-time implementation of a safety map algorithm in Chapter 8 to provide an additional level of robustness to the analysis [4].

[4]Running an algorithm in real-time on a robot can be different from running an algorithm on a dataset. This is because in real-time situations the system is forced to forgo processing stereo frames when it falls behind whereas when running on datasets one has the luxury of processing every frame. As a result, lesser data is available to the system in real-time situations leading to models with more unknown areas.

6.3 Fitting Planes to 3D Grid Segments

The above segmentation of ground regions in the 3D grid proves to be very useful and allows us to model the ground using planes. The segmentation identifies potentially distinct ground regions in the robot's environment, e.g., the segmentation in Figure 6.3(a) finds regions that are level (dark blue in the figure) and distinguishes them from regions that form part of the ramp (light blue and green and on the left in the figure). The method over-segments the ramp but the segments are large enough to allow good planes to be found. Since we are interested in wheeled mobile robots, surfaces with high inclines are considered non-traversable and not considered (by appropriately setting the height threshold w_T). This ensures that an incline is not broken up into numerous small segments.

Once we have the segmentation, the plane fitting process is straightforward. For all cells in a segment S, we find the corresponding (occupied) voxel columns and fit a plane of the form,

$$z = b_{S_1} x + b_{S_2} y + b_{S_3} \tag{6.1}$$

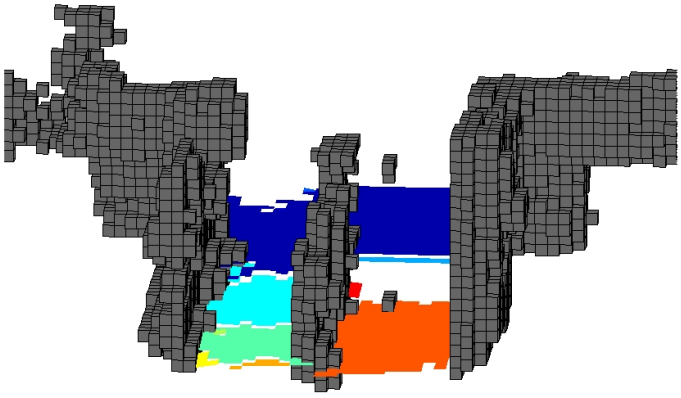
to points associated with the voxel columns. We use a standard linear least squares formulation to find the best fitting plane parameters $b_S = (b_{S_1}, b_{S_2}, b_{S_3})^T$.

Figure 6.4(a) shows the fitted planes for the segmentation produced in Figure 6.3(a). Qualitatively the plane fits seem very good and we verify this quantitatively in Chapter 8.

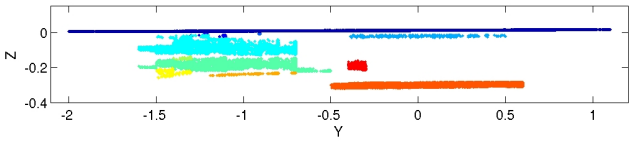
The grid segmentation algorithm and the least squares plane fitting process taken together can be thought of as a method for fitting planes to 3D point clouds. In Section 6.5 we go over related methods for fitting planes to point clouds and explain the advantages of the method introduced here over other methods.

6.4 Safety Analysis based on 3D Planes

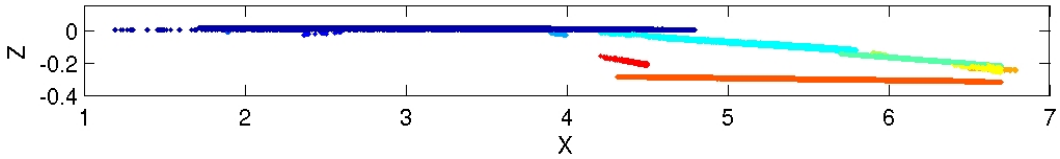
The final method analyzes the traversability-based safety map, the segments, the planes fit to the segments, and the relationships between neighboring segments (or planes) to create a



(a)



(b)



(c)

Figure 6.4: (a) The planes obtained for each of the ground segments after fitting – the colors used for the planes here are the same as those used for the segments in Figure 6.3(a). (b) and (c) show two different cross-sectional views of the planes to show more detail and the quality of the fit.

plane-based safety map. In the following discussion we (mostly) treat the segments as 2D entities.

Constructing the Plane-based Safety Map. The first step of the safety analysis consists of finding the boundary and interior cells of all segments. Interior cells are those that have eight neighboring cells, with all of those neighbors belonging to the segment itself. Boundary cells are those that have either fewer than 8 neighbors or some neighbors that belong

to other segments (or neighbors that are labeled as either “Non-ground”, “Unknown”, or “Unexamined” in the traversability-based safety map.). Next we find neighboring segments by identifying two segments as neighbors if any of their boundary cells are neighbors.

The segments are then analyzed for connectivity and labeled for safety as described in Algorithm 2. The segment labels parallel those assigned to cells in a safety map. To assign labels to the segments we start with the first segment on which the robot is currently located and label it as ”Safe”. Next the height differences at the boundaries between this segment and neighboring segments are computed using the plane parameters (see Algorithm 2 for details). If the height difference at a sufficient number of boundary cells between two neighboring segments is within a threshold then it is assumed the boundary between the two segments is traversable. This allows us to determine which of the segments neighboring the first segment are reachable by the robot. The process is then continued recursively starting with the segments deemed reachable from the first segment. All reachable segments are labeled “Safe” and unreachable segments are labeled “Non-ground”. The “Safe” segments are re-labeled as ”Level” or ”Inclined” depending on their slope.

Before the above process is started, small and thin segments are labeled as “Unknown” since such segments are usually a result of insufficient or poor depth information (most of these segments disappear with additional stereo data). In addition, segments with a large incline are labeled “Non-ground”.

Cells in the traversability-based safety map are then updated with the labels of the segments to which they belong to give the plane-based safety map.

Drop-off Edges. The boundary cells of segments labeled “Level” or “Inclined”, can be further annotated as being drop-off edges. Drop-off edges at boundary cells can be found with the help of the following two criteria.

1. When some of the cells neighboring a boundary cell have a lower height than the boundary cell, it indicates a drop-off edge.

Algorithm 2 Labeling Segments and their Planes for Connectivity and Safety

Require: Segment and plane list with neighborhood information.

Initialize by labeling all segments in the segment list as "Unexamined".
Label segments with planes with a slope greater than $\theta_{max}(=10^o)$ as "Non-ground".
Label small segments with less than $n_{small}(=6)$ cells as "Unknown".
Label thin segments as "Unknown". Thin segments are those for which: $\#interior\ cells < r_{thin} \times \#boundary\ cells$ where $r_{thin} = 0.1$. Small and thin segments are usually poorly observed and hence not considered.
Create an empty list L_S. L_S will contain segments to be examined further for safety.
Label the segment on which the robot is located as "Safe" and add it to L_S.
while $L_S \neq \phi$ **do**
    Pop L_S (i.e. remove segments from L_S) until a segment S is obtained that has not had its neighbors examined.
    Find neighboring segments, N_S, of S.
    for each $R \in N_S$ **do**
        Find neighboring boundary cells c_{RS} of S and R.
        $num_{RS} \leftarrow 0$
        for each cell $c \in c_{RS}$ **do**
            Compute the xy-coordinates (x_c, y_c) of the center of cell $c = (u, v)$ as follows: $x_c = u \cdot l_v$ and $y_c = v \cdot l_v$ (From Equation 5.12).
            Compute the expected ground height at cell c's center using the plane parameters of both segments R and S (Equation 6.1): $h_c^S = b_{S_1} x_c + b_{S_2} y_c + b_{S_3}$ and $h_c^R = b_{R_1} x_c + b_{R_2} y_c + b_{R_3}$.
            Compute the height difference, $h_c(S,R) = |h_c^S - h_c^R|$.
            if $h_c(S,R) < h_T$ (where $h_T = 0.05$m) **then**
                $num_{RS} \leftarrow num_{RS} + 1$
            end if
        end for
        if $num_{RS} \geq num_T(=5)$ **then**
            Segment R is reachable from S. Label R as "Safe" and add R to L_S.
        end if
    end for
    Mark S as having had all its neighbors examined.
end while
Re-label all remaining segments still labeled "Unexamined" as "Non-ground" as they are unreachable from the robot's current position.
Re-classify all segments labeled "Safe" into "Level" or "Inclined" segments depending on the slope of their planes. If the slope is greater than $\theta_T = 3$ degrees label the segment as "Inclined" else label it as "Level".
Return all segments with their labels.

2. When some of the cells neighboring a boundary cell are labeled "Unknown" it might indicate the presence of a drop-off edge. This condition exploits the fact that some drop-off edges occlude regions beyond them causing cells in those regions to be labeled as "Unknown".

Both criteria find cells that are drop-off edges but the second criteria also finds cells that are not drop-off edges. Therefore, we only label boundary cells satisfying the first condition as "Drop-off Edges". Many non drop-off segment boundary cells satisfy the second condition – this is specially true of boundary cells that are at the edge of the stereo sensing horizon since the cells beyond these are necessarily labeled "Unknown".

Therefore, we modify the second condition to lower the incidence of false positive boundary cells. In the modified condition only boundary cells that are next to "Unknown" cells and that are also close to cells that are lower in height than them are considered to be "Potential Drop-off Edges". Figure 6.5(a) shows the results of this method. We are able to identify correctly the only drop-off edge in this environment but we also get a few false positive drop-off edges in areas where stereo information is lacking. In Chapter 8 we omit labeling cells using the modified second condition to avoid cluttering the safety maps [5].

In the next chapter (Chapter 7) we look at a method that explicitly identifies drop-off edges for which the second condition, as originally stated, is satisfied. It uses motion and occlusion cues and its output can be used to enhance the safety maps created here.

Parameters. The parameters θ_{max}, h_T, num_T in Algorithm 2 are determined using the motion capabilities and dimensions of the robot for which the safety maps are being created. θ_{max} is determined by the maximum incline the robot can navigate (10 degrees in our implementation). h_T is the maximum height difference between two adjoining surfaces that the robot can navigate (5cm in our implementation) and num_T is the number of cells

[5] Unfortunately, in all of our datasets we do not get a single instance of a drop-off edge for which the first criteria is met. While we do not have a dataset, we did come across such a drop-off edge while testing our real-time safety mapper implementation – see Figure 2.2 in Chapter 2.

that make up the width of the robot (the robot is assumed to have a width of 5 cells in our implementation which corresponds to 0.5m).

Thus, the safety map as created, is weakly dependent on the robot's dimensions and navigation capabilities but independent of the trajectories that the mobile robot may take when navigating. We believe that it is the task of the path planner to generate and test the feasibility of different trajectories based on the safety map annotations. This clearly demarcates the goals of the mapping and path planning modules.

Results. Figure 6.5(a) shows a plane-based safety map created for the ACESRAMP environment. As we can see the region corresponding to the wheelchair ramp in the environment is correctly identified as "Inclined" and the below-ground region next to the ramp is correctly determined as non-reachable and labeled "Non-ground". In addition cells determined to be "Potential Drop-off Edges" are also shown.

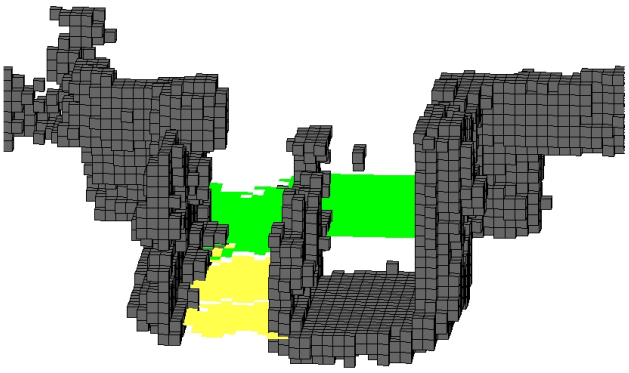
We can combine the 3D grid and the segments and their planes using the plane-based safety map to create a hybrid 3D model where traversable ground regions are represented using planes and non-ground regions are represented using voxels. Figure 6.5(b) shows such a hybrid 3D model for the ACESRAMP environment.

Computation Time. The average computation time for fitting planes to the grid segments and creating the plane-based safety map is around 35ms for an $8m \times 8m$ grid using unoptimized code written in *MATLAB*.

6.5 Related Work

In this section we review prior work in the robotics literature on fitting planes to point cloud data for the purpose of modeling a robot's surroundings. The process of fitting planes usually consists of two steps: (i) finding the assignments of points to planes, or a segmentation of the data such that all points in a segment belong to a single plane; and (ii) finding

(a)



(b)

Figure 6.5: (a) **Plane-based safety map** obtained after analyzing the segments and fitted planes for safety. The color scheme is the same as used in Figure 6.2(a) with the additional use of yellow to denote "Inclined" regions and blue for denoting "Potential Drop-off Edges". (b) **Hybrid 3D model.** Ground regions are represented using planes with "Level" planes shown in green and "Inclined" planes in yellow. "Non-ground" regions are represented using voxels (in grey).

the plane parameters themselves by fitting planes based on the assignments/segmentation. There are many different kinds of plane fitting algorithms. These include Expectation Maximization (EM) based methods [Thrun et al., 2004], methods that take an incremental approach [Poppinga et al., 2008], and RANSAC based methods [Fitzgibbon and Zisserman,

1998].

Of these methods, EM-based methods are amongst the more popular ones. Such methods iterate between finding the probabilities of point assignments and finding the plane parameters themselves based on the current best estimate of the probabilities [Thrun et al., 2004]. However, EM-based methods require that the correct number of planes in the environment be estimated in some manner beforehand and are subject to local minima that can be difficult to avoid. The local minima partly arise due to the fact that EM-based methods don't take point locality into account when assigning points to planes.

Incremental methods start by fitting a plane to a small collection of neighboring points and incrementally adding new points to the initial group followed by refitting to test the quality of the fit [Poppinga et al., 2008]. Since incremental methods start by fitting planes to small regions their results depend on the seed regions requiring that these be carefully chosen. Such methods are also sensitive to the thresholds used and are usually computationally expensive. Like incremental methods, RANSAC-based methods also start off by fitting planes to small collections of points and are iterative like EM.

If the range data is in the form of an "image", plane fitting can be formulated as a range image segmentation problem. Hoover et al. [1996] review and evaluate several methods for segmenting range image data into planes – these methods are fast and give good results [Jiang and Bunke, 1994]. Unfortunately, they cannot be applied to our 3D models since our range data is in the form of a unorganized point cloud.

Recently, Gaussian Processes [Plagemann et al., 2008] have become popular for fitting surfaces to laser range data for modeling terrain. However, GPs require heavy computation and in addition, fitting surfaces using Gaussian Processes also requires that the assignment/segmentation problem be solved. The hard part of plane or surface fitting is finding the number of planes/surfaces and determining the point assignments. Once the regions have been found it is straightforward to either fit planes using a least squares process or fit surfaces using a Gaussian Process or other method.

The plane fitting method we have introduced here consists of Algorithm 1 for segmentation and the least squares method of Section 6.3. By taking advantage of the fact that the robot can travel only on horizontal or gently inclined surfaces such as ramps, the Algorithm 1 segments the point cloud data very efficiently – albeit only finding a segmentation of points that belong to ground regions. Efficiency arises partly because the method is not iterative unlike EM or RANSAC-based methods, etc. The segmentation produced is very stable and produces fairly large segments that allow good fits to be found. Another strength of the segmentation method is that it is able to detect small changes in slope – the ramp found for the ACESRAMP environment has a slope of only about 6 degrees. We believe that this method is particularly suitable for the purposes of mobile robot navigation.

It might be possible to extend this method to vertical or sharply inclined surfaces. This will require that the normals of local patches in the point clouds be determined. Knowing the normals will allow the method to determine the directions along which to compute the "heights" in different regions which can then be used for segmenting those regions.

6.6 Summary

In this chapter we introduced three methods for building safety maps from 3D models. The threshold-based and traversability-based methods analyzed only the 3D grid whereas the plane-based method used both the grid and point cloud parts of the 3D model.

The threshold-based method was shown to be only suitable for environments that had no inclines. The traversability-based method was able to correctly identify inclined regions as being safe but was unable to identify them as being inclined. This is because the method could not distinguish between inclines and a series of steps. The plane-based method for building safety maps overcame this problem by analyzing planes fit to the 3D point cloud in addition to the 3D grid. The plane-based method successfully found traversable level and inclined regions and also identified drop-off edges.

We also introduced a fast method for fitting planes to level and gently inclined

ground regions in point clouds. The method produced a fast segmentation of the point cloud by overlaying a 3D grid on top of the point cloud and using the grid to find segments.

We presented representative results of all methods introduced in this chapter on the ACESRAMP environment. Chapter 8 presents more qualitative results obtained using the plane-based safety map method and also presents a quantitative analysis of the quality of the safety maps and the plane fits.

Chapter 7

Detecting Drop-offs using Motion and Occlusion Cues

We present a novel motion and occlusion based method that identifies drop-offs directly in front of the robot. Such drop-offs have an occluding edge and this method uses the relative motion (across several images) between this edge and other image features for detecting the drop-off.

The method works on a sequence of monocular images and does not require any stereo information. It can, however, be used in conjunction with the stereo-based methods for constructing safety maps. It provides redundancy and robustness in case the stereo method fails to detect a drop-off. Also unlike stereo methods, where drop-off edges have to be sometimes inferred (for example, the existence of an "Unknown" region next to a "Level" region can imply a drop-off edge), the motion and occlusion based method explicitly identifies drop-off edges. The method consists of two main steps:

1. In the first step, all occluding edges in front of the robot, that *might* be drop-off edges, are detected. The algorithm for doing this is purely image based and does not utilize any camera information, such as focal length.

2. In the second step, the identified edges are projected onto a horizontal 2D *motion grid*. The purpose of this step is threefold: it identifies the drop-offs edges amongst the occluding edges; it provides the 3D location of the edges in the world; and it also reduces the incidence of false positives.

Information about the drop-off edges from the 2D motion grid can be added to 2D metrical maps, such as the safety maps, to provide additional safety information to the robot.

7.1 Detecting Occluding Edges

Drop-offs on the ground have an occluding edge. If a robot moves towards an occluding edge, regions initially occluded by the edge come into view, and when the robot moves away from the edge, regions that were initially visible, disappear. If the robot were to track a fixed point (or feature) beyond the occluding edge and another point before the edge across several images, it would notice that the point beyond the edge moves "faster" than the point before, due to new regions coming into view beyond the edge (or going out-of view, depending on the robot's direction of motion). The same would not be true for a non-occluding edge – both points would appear to move at the same rate. We utilize this difference in the apparent "speed" of points relative to the edge to identify occluding edges and distinguish them from non-occluding edges.

The algorithm for detecting occluding edges, consists of matching edges across images and detecting invariant features around the edges. The average speeds of features above an edge and below an edge [1] are estimated and the difference thresholded to identify occluding edges. The threshold, δ_T, is set such that the method identifies occluding edges aggressively, resulting in a relatively high false positive rate (that is some non-occluding edges get incorrectly identified as occluding edges). This is preferred to the other scenario in which occluding drop-off edges are missed, as missing a drop-off carries a higher risk

[1] Features that are beyond an edge in 3D space appear above the edge in image space, and features that are before an edge in 3D space appear below the edge in image space.

for the robot.

The pseudo-code for the method is presented in Algorithm 3. Figure 7.1 shows the various stages of Algorithm 3 when applied to a pair of images separated by several frames in an outdoors environment. For ease of explanation we assume that the robot is moving towards a drop-off.

As mentioned, the drop-off detection method detects all occluding edges in the environment that have the potential to be drop-offs. Figure 7.1 provides an instance of this – in the last row of the Figure, the occluding edge of a pipe in the distance is also treated as a potential drop-off even though it is not one.

Matching Edges. One of the major steps in Algorithm 3 is matching edges across images. We propose a method for doing so in Algorithm 4. Matching edges is tricky because the lengths of edges change from image to image. This is partly due to the robot's motion, but mostly due to the edge detection process – an edge in one image can appear as two smaller edges in another image. Therefore, we need a matching method that is invariant to the length of the edge. The method we propose matches edges across images by comparing histograms of intensities in regions around the edges – since the number of bins in an intensity histogram is independent of the edge length. The histograms are compared using the Jensen-Shannon divergence [Wikipedia, 2009b] which measures how similar two histograms are to one another. We expect that the regions around matching edges should look similar and hence the histograms of the regions should be similar – so pairs of edges for which the Jensen-Shannon divergence is low are considered for matching.

Algorithm 3 Motion-based Detection of Occluding Edges.

Require: A pair of images, I_1, I_2 separated by $N = 5$ frames.

1: Initialize the set of occluding edges, O_1, in image I_1, to ϕ.
2: Find all edges in images I_1 and I_2 (we use code available online [Kovesi, 2008]). An edge is defined as a straight line segment.
3: Only consider edges that are, (i) nearly horizontal in the image (slope $\leq 30^o$), and (ii) not very far off (distance $\leq 20m$) – accomplished by discarding all edges above the horizon.
4: Match edges in I_1 with edges in the previous image, I_2 (Algorithm 4 describes the method we use). The separation of five frames ensures appreciable motion thereby increasing the signal-to-noise ratio. Let n_e = number of matched edges.
5: **for** i = 1 to n_e **do**
6: Denote matched edge i by e_i.
7: Find invariant features [Mikolajczyk and Schmid, 2004] above and below edge e_i in I_1 and I_2, and within a certain region around the edge.
8: Match the features across I_1 and I_2. Features above and below the edges are matched separately – the method used is a variation of the one in [Lowe, 2004]. Let n_a and n_b be the number of matched features above and below the edge respectively.
9: **for** j = 1 to n_a **do**
10: Let f_j be matched feature j above edge e_i
11: Compute the distance a_{j1} (in pixels), of f_j to e_i in image I_1. Similarly compute distance a_{j2}.
12: Compute the distance that f_j moves between images I_1 and I_2: $\delta_j = |a_{j1} - a_{j2}|$
13: **end for**
14: Similarly compute δ_k for all features, f_k, $k = 1..n_b$, below e_i
15: Histogram all distances, $\delta_j,\ j = 1..n_a$, and pick the bin with the highest count as the best estimate of the average distance, δ_a, moved by features above e_i relative to e_i.
16: Similarly compute δ_b.
17: **if** $\delta_{ab} = |\delta_a - \delta_b| > \delta_T$ **then**
18: Add e_i to the set of occluding edges, O_1.
19: **end if**
20: **end for**
21: Return O_1.

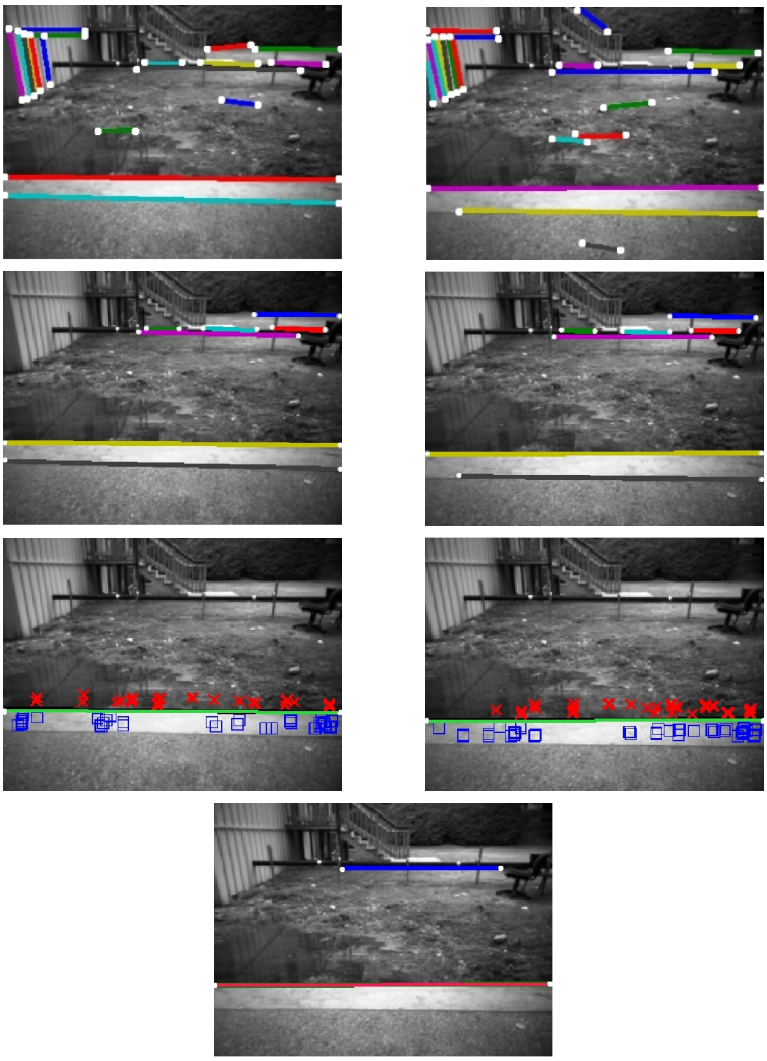


Figure 7.1: Images illustrating Algorithm 3 as applied to an outdoor environment. The robot is moving towards the drop-off edge. 1st Row: The edges detected in a pair of images separated by several frames. 2nd Row: Edges are filtered and matched across the image pair (matching edges have the same color in both images). 3rd Row: For a pair of matched edges, features are found both above (red crosses) and below (blue boxes) the edge and matched across the image pair. The distance moved by the matched features, above and below the edge, is computed separately. 4th Row: The occluding edges found – note that a pipe in the distance is also correctly identified as an occluding edge.

Algorithm 4 Matching Edges using Histogram Similarity

Require: A pair of images, I_1, I_2 and the ordered set of edges E_1 and E_2 in each.

1: Initialize the ordered set E_{12}, that will contain edges from E_2 that match edges in E_1, to ϕ.
2: **for** i = 1 to $|E_1|$ **do**
3: Denote by e_{1i}, edge i in E_1
4: Define an oriented rectangular search region, parallel to and centered on e_{1i}, in image I_2, whose size is based on the maximum expected amount of edge movement between the two images.
5: Create a set, M_i, containing those edges from E_2 that might match e_{1i}. M_i includes any edge in E_2, a substantial part of which falls within the above search region.
6: Define two oriented rectangular histogram regions around edge e_{1i} - one above it and the other below.
7: Compute normalized histograms h_{ia} and h_{ib} (i.e., probability distributions) of pixel intensities in the histogram regions above and below e_{1i}.
8: **for** j = 1 to $|M_i|$ **do**
9: Denote by e_{2j} edge j in M_i
10: Similar to that done for e_{1i}, compute normalized intensity histograms h_{ja} and h_{jb} above and below edge e_{2j}.
11: Compute the Jensen Shannon divergence, D_{JS}, between h_{ia} and h_{ja}:

$$D_{JS}(h_{ia}||h_{ja}) = (1/2)(D_{KL}(h_{ia}||h_a) + D_{KL}(h_{ja}||h_a)) \tag{7.1}$$

where $h_a = (1/2)(h_{ia} + h_{ja})$ and D_{KL} is the Kullback-Leibler divergence [Wikipedia, 2009c], defined as $D_{KL}(P||Q) = \sum_k P(k)(P(k)/Q(k))$, between two discrete distributions P and Q.
12: Similarly, compute $D_{JS}(h_{ib}||h_{jb})$.
13: Let $D_j = D_{JS}(h_{ia}||h_{ja}) + D_{JS}(h_{ib}||h_{jb})$.
14: **end for**
15: Find edge $j*$ in M_i for which D_j is smallest. We expect the sum of the Jensen Shannon divergences to be small for edges whose surrounding regions look similar.
16: **if** $D_{j*} \leq T(= 0.1)$ **then**
17: Add e_{2j*} to set E_{12}.
18: **else**
19: Add the empty edge e_ϕ to E_{12}. Essentially, if D_{j*} is greater than T, it means a good match was not found.
20: **end if**
21: **end for**
22: Return E_{12}

7.2 Projection onto a 2D Grid: Identifying Drop-off Edges

For a given image, Algorithm 3 finds a set of occluding edges that can potentially be drop-off edges. The next stage of the drop-off detection method, that of projecting the occluding edges onto a horizontal 2D grid, helps to identify true drop-off edges amongst these occluding edges.

If we assume the robot is travelling on level ground and that drop-off edges are also on level ground surfaces, we can compute the 3D location of edge pixels in the map coordinate frame using the equations in Section 4.1 – this is because the height of level ground is always known in the robot's coordinate frame. Section 7.2.1 describes the math behind computing drop-off locations. Knowing the 3D location of drop-off edge pixels allows us to project them onto a 2D horizontal grid in the map coordinate frame, called the motion grid. This motion grid is essentially a 2D map of the robot's local environment like the local safety maps, except that in this case the map only contains the locations of drop-off edges (cells in the grid are marked as drop-off edges or not)

The motion grid also helps to get rid of occluding edges that are not on the ground plane and hence are not drop-offs. For every image in which occluding edges are detected, we compute the 3D locations of those edges using the equations in Section 7.2.1. Then each cell in the motion grid through which a drop-off edge passes is incremented by a fixed amount. Only cells with values above a threshold are marked as being drop-off edges [2]. Hence, each cell has to accumulate evidence over several images for being marked as a drop-off edge.

The 3D locations of occluding edges not on the ground, computed using the equations in Section 7.2.1, are not consistent across images. Hence different cells in the motion grid get incremented each time eliminating such occluding edges. Thus the process of projecting drop-off edges onto a 2D grid, identifies drop-off edges and also handles the high number of false positives generated by Algorithm 3, in addition to providing the 3D loca-

[2] In our code we require that a cell be incremented at least 6 times before it is marked as a drop-off edge.

tions of drop-off edges. Figure 7.2 shows the drop-offs detected and the motion grid for the ACESRAMP environment (Figure 7.3 has additional results).

In the case when the robot travels on inclined surfaces of the sort we consider in this work, projecting the drop-off edge pixels using the equations in Section 7.2.1 should still work. This is because we only consider travel on surfaces with relatively low slopes and so the level ground assumption approximately holds. The locations of the drop-off edges won't be as precise as for level ground, but the drop-offs themselves should be detected.

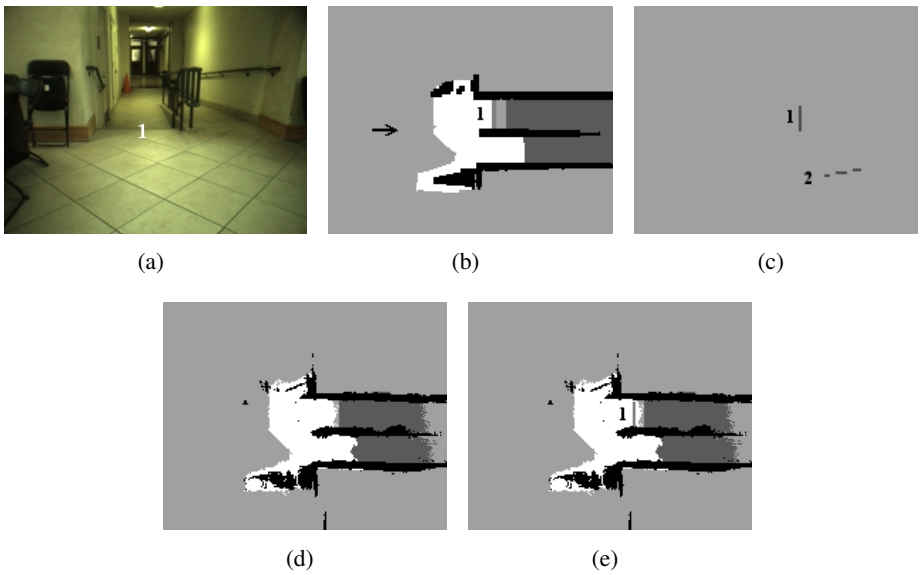
Computation Time: Both steps of the motion based method take a total of 6.7 seconds on average per image using un-optimized Matlab code. We believe the method can be made to run in real-time using optimized C++ code.

7.2.1 Computing Drop-off Edge Locations

For an image point (or pixel) $\mathbf{z} = (r, c, d)^T$, on a drop-off edge on level ground, let $\mathbf{x}^r = (x^r, y^r, z^r)^T$ be its location in the robot's coordinate frame. In our work, the ground plane is always at height zero in the robot's coordinate frame, and so we can set $z^r = 0$ for the point. Substituting this into Equation (4.3) in Section 4.1 we get,

$$\begin{aligned} \mathbf{x}^r &= \begin{pmatrix} x^r \\ y^r \\ 0 \end{pmatrix} = R_c \mathbf{x}^{cam} + T_c = \begin{pmatrix} r_1^T \\ r_2^T \\ r_3^T \end{pmatrix} \mathbf{x}^{cam} + \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix} \\ or, \quad 0 &= r_3^T \mathbf{x}^{cam} + t_3 \\ or, \quad -t_3 &= (r_{31}\ r_{32}\ r_{33}) \begin{pmatrix} x^{cam} \\ y^{cam} \\ z^{cam} \end{pmatrix} = r_{31} x^{cam} + r_{32} y^{cam} + r_{33} z^{cam} \end{aligned} \tag{7.2}$$

We can substitute Equation (4.1) for $\mathbf{x}^{cam}$ in the above equation to solve for a value of the



(a) (b) (c)

(d) (e)

Figure 7.2: Figures showing the 2D motion grid and its combination with a safety map. (a) An image showing a scene from the ACESRAMP environment with a drop-off edge (marked 1). (b) A ground truth safety map of the environment, obtained using height thresholds (Section 6.1), showing the location of the drop-off in the map. The arrow shows the direction from which the scene in the previous image was taken. (c) A 2D motion grid showing two drop-off edges found using the motion-based drop-off detection method described in this Chapter. The correct drop-off edge, marked 1, is identified in addition to a false positive drop-off edge marked 2. (d) A safety map of the environment obtained using a color-segmentation stereo method [Murarka et al., 2008] (similar to that described in Section 4.3.1). (e) The same safety map after being combined with the motion grid. The location of the drop-off edge marked 1, has now been added to the safety map. Drop-off edge 2 happened to fall on the right wall, allowing us to eliminate it. Thus combining the motion grid with the safety map led to a better map overall.

(virtual) disparity d of the point,

$$\begin{aligned} -t_3 &= r_{31}\frac{bf}{d} + r_{32}\frac{b(c_0-c)}{d} + r_{33}\frac{b(r_0-r)}{d} \\ or, \quad d &= \frac{-b}{t_3}(r_{31}f + r_{32}(c_0-c) + r_{33}(r_0-r)) \end{aligned} \tag{7.3}$$

Once we have d we can use Equations (4.1), (4.3), and (4.4) in order to solve for the 3D position of the edge point, $\mathbf{x}^m$, in the map coordinate frame.

7.3 Adding Detected Drop-offs to a Safety Map

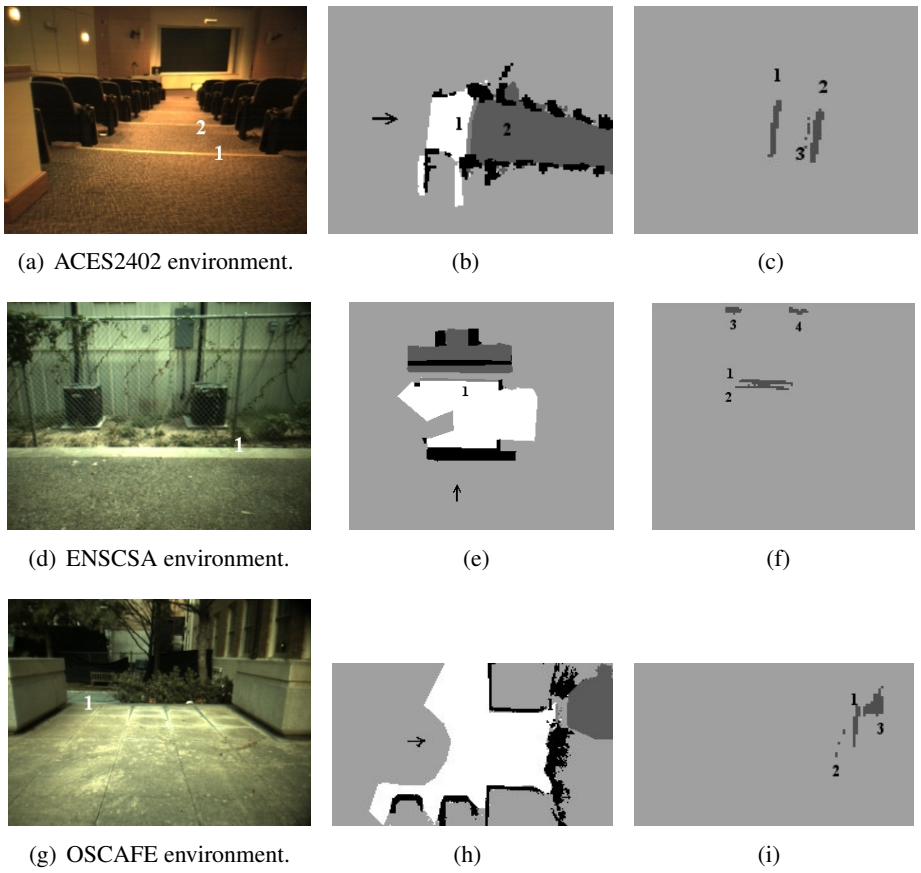
The motion-based drop-off method returns a horizontal 2D grid with drop-off cells marked. This grid can be combined with a safety map, created using one of the stereo methods, to obtain a combined stereo and motion safety map. In the combined safety map, cells in the stereo safety map marked "Level" or "Inclined" are further annotated as "Drop-off Edges" if the corresponding cells in the motion grid are marked as drop-off edges. If cells in the motion grid fall on any other type of cell in the stereo safety map it is assumed that the drop-off is a false positive and ignored. Figure 7.2 shows the results of combining a stereo safety map with a motion grid.

7.4 Evaluation and Results

The motion-based drop-off detection method was evaluated on four stereo video data sets (with 350-500 stereo image pairs in each dataset) collected by driving the robot through two indoor and two outdoor environments. Sample images from the environments and the results on those environments are shown in Figures 7.2 and 7.3.

The method was evaluated based on the drop-offs detected in the 2D motion grids that were created for all environments. The method was able to detect all 5 frontal drop-offs present in the four environments. This corresponds to a true positive rate of 100% (a drop-off edge correctly detected is a true positive) or a false negative rate of 0% showing that this method can be relied on for detecting drop-off edges in front of the robot.

However, the motion grids also contained a total of 7 false positive drop-offs. We do not report the corresponding false positive rate as it does not make much sense in this case. In order to calculate the false positive rate we need to divide the number of false positives by the number of safe edges in the environment (i.e., edges that are not drop-off edges) *that are considered by our algorithm*. There are several different plausible ways of finding the total number of safe edges that can give varying estimates of the total number. For example,



(a) ACES2402 environment. (b) (c)

(d) ENSCSA environment. (e) (f)

(g) OSCAFE environment. (h) (i)

Figure 7.3: Results of applying the motion-based drop-off detection method on three environments. The first column shows scenes from the environments with numbers indicating the location of drop-off edges in each image. The second column shows ground truth safety maps along with the locations of the drop-off edges identified in the first column. The third columns shows the motion grids built for the three environments and all drop-offs that were identified. In Figure (c), both drop-offs in the environment were identified successfully although a false-positive drop-off, marked 3, was also identified. In Figure (f), three false-positive drop-offs (marked 2, 3, and 4) were identified in addition to the correct one. In Figure (i), two false positive drop-offs (marked 2 and 3) were identified in addition to the correct one.

should we consider edges that are only seen once by the robot or do we only consider edges that have been observed several times? Also, what do we do about edges that are caused due to lighting conditions but are nevertheless considered by our algorithm? Due to these factors

we omit computing the false positive rate as it does not seem to convey much additional information. Instead a more useful number is the average number of false positive edges found per environment which happens to be 1.75 false positives per environment for our experiments.

How do these false positive drop-off edges affect motion planning? When the motion grid is combined with a stereo safety map most of the false positives go away since usually these are produced by occluding edges not on the ground. Examples of this include edge 2 in Figure 7.2. The edge disappears on combining with the stereo map as it corresponds to a hand rail on the wall. Similarly, in Figures 7.3(c) and 7.3(i) all false positive edges would disappear if we were to combine the motion grid with the stereo safety maps of the environments. However, in some cases the drop-offs edges do not disappear, for example, edge 2 in Figure 7.3(f) will appear as a drop-off edge. However, since the edge is close to an actual drop-off, in this case too the false positive should not prove to be a hindrance to motion planning.

Therefore based on the above examples, we believe that the majority of false positive edges found using this method will not be a significant hindrance to motion planning.

7.5 Related Work

Here we briefly discuss work that is most directly related to our method for detecting drop-offs. Other work on detecting drop-off edges using stereo vision and other sensors is discussed in Chapter 3.

We believe that the method presented in this chapter is the first time that drop-off edges have been detected in this manner. There has been work on finding occluding edges using optical flow in the computer vision community but our work differs from that work in the sense that we use "relative" optical flow as opposed to traditional "absolute" optical flow. In traditional optical flow the absolute motion of pixels between two images is computed. In our method we compute the motion of pixels relative to fixed features (edges)

in two images - hence the word “relative”.

An example of a work where “absolute” optical flow is used to detect occlusion boundaries is the work by Stein and Hebert [2006]. They use “absolute” optical flow to detect occlusion boundaries corresponding to *moving* objects in video sequences. The presence of motion makes it easier to detect the object boundaries. The “relative” optical flow method we use is more sensitive at detecting occlusions and works even when the occluding objects (drop-off edges in our case) in question are not moving.

7.6 Summary

In this chapter we have presented a novel motion and occlusion based method for detecting drop-offs edges in front of the robot. The method works by first detecting occluding edges in the environment and then projecting these edges onto a 2D motion grid to give the drop-offs. In experiments the algorithm has been demonstrated to detect all such drop-offs in four different environments. The method also finds several false positive drop-off edges. However, most of these edges disappear when the motion grid is combined with metrical maps of the environments, such as the local safety maps.

The motion based drop-off detection method does not detect lateral drop-offs. However, the stereo methods are good at detecting lateral drop-offs since the edges of such drop-offs are roughly parallel to the camera axis providing good parallax. On the other hand, drop-offs in front are not as easy for stereo methods to detect. So, in some sense, the motion method is complementary to stereo methods for detecting drop-offs. Thus in combination with the stereo local safety maps, the motion based algorithm gives us a fairly robust method for the detection of drop-offs.

Chapter 8

Evaluation and Results

Through Chapters 4, 5, and 6 we have presented several algorithms for the three main steps involved in building a safety map, namely,

(i) computing a stereo depth map,

(ii) building a 3D model, and,

(iii) analyzing the 3D model for safety.

The overall algorithm for building a safety map then consists of a combination of algorithms, one from each of the three steps. In this Chapter we evaluate and compare algorithms from steps (i) and (ii) based on the quality of the safety maps that are constructed using them [1]. When evaluating different algorithms from step (i), we keep the algorithms in the remaining steps constant. Similarly, when evaluating step (ii) algorithms, we keep the algorithms in steps (i) and (iii) the same.

In addition to providing us with an assessment (and comparison) of the performance of the individual algorithms involved in the three steps of building safety maps, the above

[1] Of the three algorithms presented for step (iii) in Chapter 6, only the third one which involves plane fitting is considered here. As shown in that chapter, the other two algorithms are not suitable for environments with ramps, such as the ones considered in this chapter.

evaluation also allows us to compare the overall performance of different combinations of algorithms. The best performing algorithm combination is then evaluated further. We also provide an analysis of the error rates that we compute for the safety maps and what these mean to the robot in terms of the safety of the paths planned.

We implement a real-time version of the best algorithm combination and provide qualitative results of the algorithm's performance when used on the intelligent wheelchair. We also demonstrate the algorithm's integration with path planning and the HSSH.

We begin this Chapter with an overview of our evaluation methodology and the stereo video datasets that we test our algorithms on. The evaluation methodology is general and should allow comparison with, and evaluation of, other vision-based mapping algorithms. Towards this end we plan to make our datasets and evaluation code publicly available [Murarka and Kuipers, 2009a].

8.1 Evaluation Methodology

In the first part of the evaluation in Section 8.2, we measure the accuracy of the algorithms from steps (i) and (ii) at constructing safety maps. We do this by running the algorithms on five stereo video datasets and comparing the constructed safety maps to ground truth safety maps and computing error rates. We also provide a qualitative evaluation of the algorithms by showing the constructed safety maps, the 3D models, and the planes that are found, for selected datasets.

In the second part of the evaluation in Section 8.3, we further evaluate the algorithm combination that is deemed to give the best overall performance as determined from the first part of the evaluation. In particular, we measure the following quantities.

- Plane accuracy: We measure the accuracy of the plane fitting process by comparing planes that are fit against ground truth planes.
- Frame latency: We measure latencies that arise due to building the 3D model and

(a) ACESRAMP environment.

(b) BIORAMP environment.

(c) ACES2302 environment.

(d) OSCAFE environment.

(e) NANORAMP environment.

Figure 8.1: Sample images from the five stereo video datasets on which the safety map algorithms are evaluated.

noise filtering. Information has to be accumulated over several image frames before the existence of an object is confirmed and it is shown in the safety map.

8.1.1 Datasets

As mentioned, the algorithms are evaluated on five stereo video data sets collected in a variety of environments using the Intelligent Wheelchair (Figure. 1.1). The datasets have between 350-500 stereo image pairs each. Sample images from the datasets are shown in Figure 8.1. Two of the datasets, ACESRAMP and NANORAMP, have a ramp and a drop-off. One dataset, BIORAMP, has two ramps, and the remaining two datasets, ACES2302 and OSCAFE, have drop-offs only. Most environments have large poorly textured regions, and both indoor and outdoor areas are represented. Lighting varies from good to fair.

Laser range data from both the horizontal and vertical laser range-finders on the wheelchair was collected simultaneously with video data and was used to provide ground

truth for our evaluation. Laser based maps of the environment were created and then manually annotated and corrected to give the ground truth safety maps used in the first part of the evaluation (the laser maps provided excellent starting points for the ground truth maps). We also manually fit planes to laser range points to give ground truth planes used in the second part of the evaluation.

8.1.2 Algorithms

The algorithms compared are as follows. Three algorithms from step (i) (computing stereo depth maps) from Chapter 4 are compared:

1. CS: The correlation stereo algorithm from Section 4.2.
2. SS: The color segmentation-based stereo algorithm from Section 4.3.1. In particular we evaluate the SS-Used version of this algorithm described in Section 4.4.
3. ES: The edge segmentation-based stereo algorithm from Section 4.3.2. We evaluate the ES-Used version of this algorithm described in Section 4.4.

Two algorithms from step (ii) (building 3D models) from Chapter 5 are compared:

1. DA: The data association method presented in Section 5.1.
2. OG: The occupancy grid based method presented in Section 5.2.

To compare the step (i) stereo algorithms, we have to use the same algorithms in the second and third steps of the overall algorithms that are used to build the safety maps. We use the occupancy grid algorithm, OG, in the second step and the plane fitting algorithm, denoted PF, algorithm in the third step giving the following overall algorithm combinations that are compared: CS+OG+PF versus SS+OG+PF versus ES+OG+PF.

Similarly, to compare the step (ii) 3D modeling algorithms, we use the correlation stereo algorithm, CS, in first step and the PF algorithm in the third step. This results in the following algorithm combinations being compared: CS+DA+PF versus CS+OG+PF. Note

that the algorithm combination CS+OG+PF is common to both step (i) and (ii) evaluations. *Thus we evaluate four algorithm combinations in all.*

Based on the evaluation of the above algorithms, we can pick the best algorithms from steps (i) and (ii), to give us the best algorithm combination for building maps [2], which turns out to be CS+OG+PF.

8.2 Evaluation Part 1: Comparison of Step (i) and Step (ii) Algorithms

The first part of the evaluation measures the accuracy of the algorithms by comparing the constructed safety maps against ground truth safety maps and computing error rates. For each constructed safety map we can identify the following types of cells.

- False Positives (FP): cells marked unsafe in the constructed safety map but labeled safe in the ground truth safety map.
- False Negatives (FN): cells marked safe in the constructed safety map but labeled unsafe in the ground truth map.
- True Positives (TP): cells marked unsafe in the constructed map that are also labeled unsafe in the ground truth maps.
- True Negatives (TN): cells marked safe in the constructed map that are also labeled safe in the ground truth maps.

For the purpose of evaluation, cells marked "Level" or "Inclined" in a safety map are considered safe and only cells marked "Non-ground" are considered unsafe. Cells marked "Unknown" or "Unexamined" are considered to be *un-classified* and hence play no role.

[2] Since steps (i) and (ii) are relatively independent of each other we don't need to test all possible algorithm combinations – simply using the best algorithm from each step should give us the best combination.

For a given dataset and algorithm, a safety map is created for every stereo image frame in the dataset [3]. For image frame k, we find the total number of FP cells, N^k_{fp}, and the total number of FN cells, N^k_{fn}, in the corresponding safety map. Also, of all classified cells in a constructed safety map, we find the total number of true safe cells present N^k_{safe} (the cells have to be classified by the ground truth map as well). Similarly, we find the total number of true unsafe cells in each safety map, N^k_{unsafe}.

We can then compute overall false negative and false positive rates for a dataset (for a particular algorithm) as follows,

$$fp_{rate} = \frac{\sum_k N^k_{fp}}{\sum_k N^k_{safe}} \tag{8.1}$$

$$fn_{rate} = \frac{\sum_k N^k_{fn}}{\sum_k N^k_{unsafe}} \tag{8.2}$$

where the sum is over all images k in the dataset. Once these error rates have been computed for each dataset, they are averaged across all five datasets to give the average false positive rate, fp_{avg}, and average false negative rate, fn_{avg} for the algorithm. In addition, the standard deviations fp_{std} and fn_{std} are also computed [4].

Tables 8.1 and 8.2 show these averaged error rates (and corresponding standard deviations) for each of the four algorithm combinations described in the previous section (8.1.2). The error rates are computed for three different settings of the parameter *decr* mentioned in section 5.2 for algorithms CS+OG+PF, SS+OG+PF, and ES+OG+PF; and for three different settings of the parameter *cmin* mentioned in section 5.1.3 for algorithm CS+DA+PF. Figure 8.2 shows graphically, as ROC curves, the fp_{avg} and fn_{avg} values presented in the tables (the ROC curves actually show $tp_{avg} = 1 - fn_{avg}$).

[3] In practice, safety maps are created for about 90-95% of all stereo frames in a dataset. Image frames that capture exactly the same scene as previous image frames (this happens when the robot is stationary) are discarded to avoid temporally correlated stereo range readings. See section 5.2 for more details.

[4] Since the standard deviations are computed over only five values they are large in some cases.

Error Rate (%) →	$fn_{avg} \pm fn_{std}$	$fp_{avg} \pm fp_{std}$
Parameter →	$decr = -1$	
CS+OG+PF	3.0 ± 1.8	9.3 ± 4.7
SS+OG+PF	14.0 ± 17.9	8.9 ± 5.2
ES+OG+PF	13.1 ± 16.7	8.6 ± 4.7
Parameter →	$decr = -3$	
CS+OG+PF	8.5 ± 5.3	6.2 ± 3.7
SS+OG+PF	22.1 ± 14.4	4.3 ± 2.7
ES+OG+PF	12.1 ± 6.7	8.7 ± 9.6
Parameter →	$decr = -5$	
CS+OG+PF	11.2 ± 7.3	5.2 ± 2.4
SS+OG+PF	17.4 ± 8.9	3.6 ± 1.6
ES+OG+PF	11.7 ± 8.8	11.6 ± 11.2

Table 8.1: **Error rates (%) for step (i) algorithms: CS+OG+PF, SS+OG+PF, and ES+OG+PF.** FN and FP error rates averaged across all five datasets, are shown for three values of the $decr$ parameter for the three algorithms. Standard deviations are also given.

Error Rate (%) →	$fn_{avg} \pm fn_{std}$	$fp_{avg} \pm fp_{std}$
Parameter →	$cmin = 3$	
CS+DA+PF	0.9 ± 0.7	27.1 ± 18.2
Parameter →	$cmin = 5$	
CS+DA+PF	1.4 ± 1.0	15.6 ± 10.2
Parameter →	$cmin = 7$	
CS+DA+PF	2.2 ± 1.2	21.3 ± 16.9

Table 8.2: **Error rates (%) for step (ii) algorithm: CS+DA+PF.** FN and FP error rates averaged across all five datasets, are shown for three values of the *cmin* parameter for the CS+DA+PF algorithm. Standard deviations are also given.

The algorithms were deemed to be sensitive to the $decr$ and *cmin* parameters and hence we measured their effect on the error rates. We expect this sensitivity because both parameters determine the weight we give to individual stereo range readings when building the 3D models and therefore control the amount of noise in the 3D models.

In addition to the quantitative evaluation, we also present qualitative results comparing the performance of the different methods. Figure 8.3 shows safety maps built for three datasets using all four algorithm combinations. These safety maps correspond to a

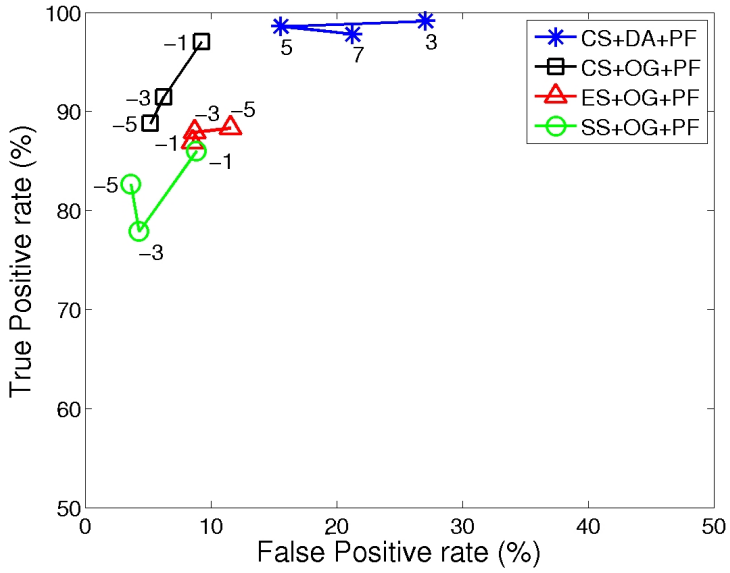


Figure 8.2: ROC curves showing the average true positive rate $tp_{avg} = 1 - fn_{avg}$ versus the average false positive rate fp_{avg} for different parameter values. The curve for CS+DA+PF is shown for three values of the *cmin* parameter. The curves for the remaining algorithms are shown for three values of the *decr* parameter.

particular image time/frame in the dataset [5]. Figure 8.4 shows the corresponding 3D hybrid models, while Figures 8.5, 8.6, and 8.7, show the corresponding planes that were found. Figure 8.8 shows 3D grids created using the CS+DA+PF and CS+OG+PF algorithms.

Of the two types of errors, false positive and false negative, false negative errors are of greater importance since the presence of false negative cells (unsafe cells marked safe) places the robot at risk of damage. It should be noted that a particular false negative rate does not translate directly into the chances of an accident, e.g., a value of 3 for fn_{avg} rate (such as that for CS+OG+PF in Table 8.1) does not mean the robot has a 3 percent chance of having an accident. False negative cells are usually at some distance from the robot, most likely arising because the robot has insufficient information about distant objects, and don't pose an immediate danger to the robot. Furthermore, robots usually have a margin of

[5] We would like to note that there is no "final" safety map for a dataset, e.g., a map obtained after processing all image frames in the dataset. Navigation is a continuous process and hence the safety map at any given timestep is the best and "final" map available to the robot at that time.

safety and so in general the effect of a particular false negative rate is expected to be less than what the numbers suggest.

False positive cells (safe cells marked unsafe), while not posing a danger to the robot, can be a hindrance to navigation because the robot might see obstacles where there are none. This would indeed be the case if false positives were distributed randomly across the safety map. However, based on the safety maps constructed (Figures 8.3 and 8.10) we see that most of the false positive cells occur adjacent to existing objects - that is most false positives are caused by objects "bleeding" into nearby regions. Furthermore, as the robot comes closer to the objects and gets more information about them, the false positives disappear.

In Section 8.4 we present a preliminary analysis of what false negative and false positive rates actually mean to a robot in terms of navigation and safety.

The ROC curves in Figure 8.2 show that we can trade-off the false positive and false negative rates by changing the values of the *decr* or *cmin* parameters. Usually increasing the false positive rate leads to a decrease in the false negative rate and vice versa. This is not true for SS+OG+PF and CS+DA+PF. For CS+DA+PF, what happens is that for a parameter setting of $cmin = 5$ the algorithm performs well on all datasets giving a lower false positive rate than that for the other parameter values. For SS+OG+PF for $decr = -5$, the algorithm performs very well on one dataset (on which the algorithm performs poorly for the other two parameter values) leading to a lower false negative rate even though on the other datasets SS+OG+PF gives poorer performance for $decr = -5$.

Comparison between step (i) stereo algorithms: CS, SS, and, ES. Based on the results (Table 8.1 and Figures 8.2 to 8.7) for algorithms CS+OG+PF, SS+OG+PF, and, ES+OG+PF , we make the following observations.

- CS has the lowest average FN rates (or highest TP rates).
- SS has the lowest FP rates but the highest FN rates.

- However, even the lowest FN rates of ES and SS are higher than the highest FN rates of CS.
- If we compare CS to ES, we can say that CS provides better performance because on the whole it has lower FN and FP rates.
- If we compare CS to SS, we see that SS has lower FP rates than CS although not by much. CS thus provides the best compromise between the FP and FN error rates of the three methods.
- SS and ES fail to detect the drop-off in the ACES2302 environment (Figure 8.3 and 8.4). This happens due to the presence of incorrect range readings at the drop-off edge that cause an inclined plane being found at the edge. This leads the robot to believe that there is a traversable incline where the staircase step is.
- Qualitatively the planes found using the CS method on the whole seem to be better than the planes found using the SS and ES methods (Figures 8.5, 8.6, and 8.7).

Based on these observations, it appears that CS has the best overall performance and provides the best compromise between the two error rates.

As mentioned in Chapter 4, these results are contrary to those obtained using the evaluation framework of [Scharstein and Szeliski, 2002] using which SS and ES were shown to perform better than CS. As discussed in Section 4.4, this suggests that we need to be careful when interpreting the actual performance of stereo algorithms based on their evaluation on very few images as done in [Scharstein and Szeliski, 2002]. In our video datasets, the algorithms are tested on several hundred images. We therefore believe that the evaluation of the three stereo methods done here is is a better reflection of the stereo algorithms capabilities than the evaluation of [Scharstein and Szeliski, 2002].

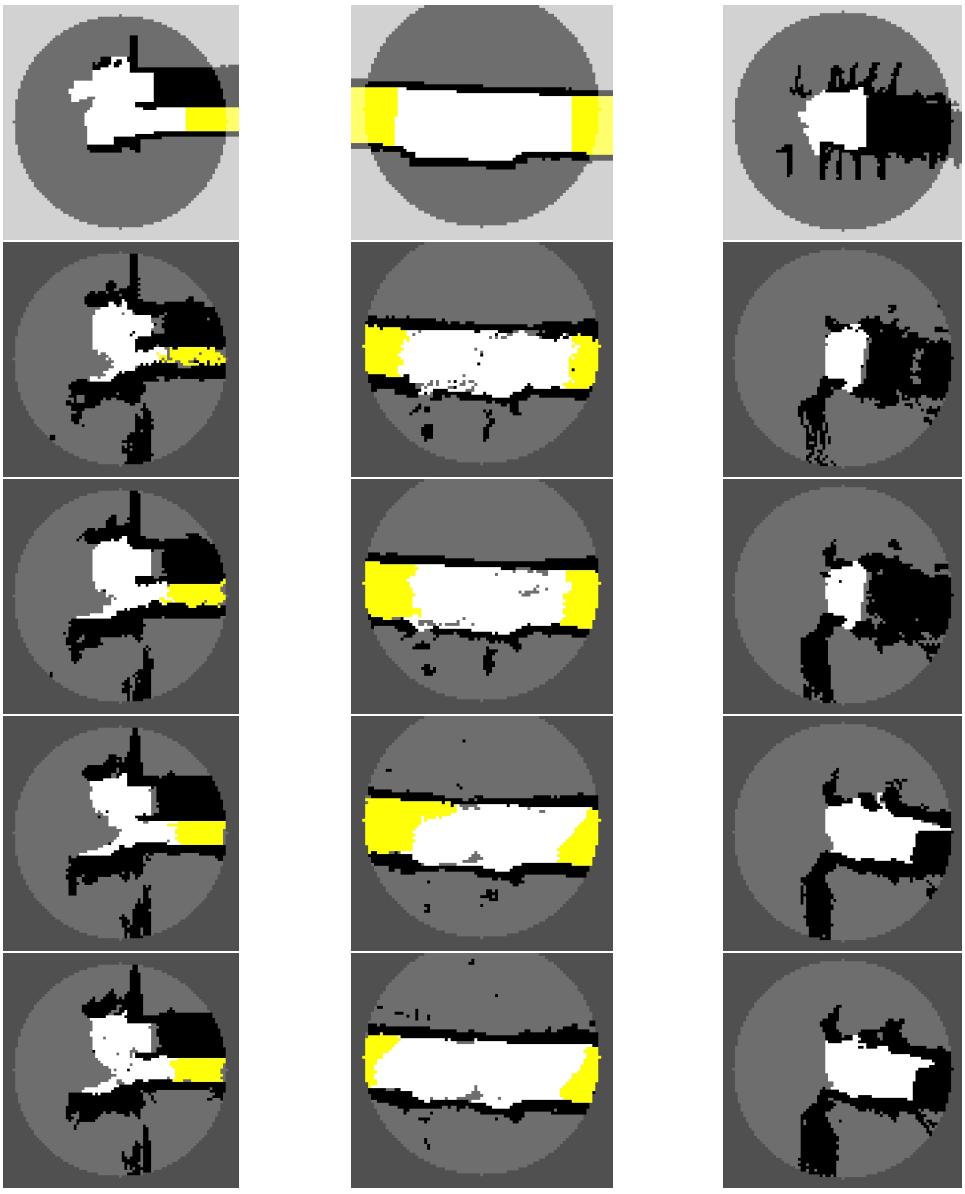


Figure 8.3: **Safety maps** of 3 datasets (columns) made by 4 algorithms (rows). Rows 2 to 5: Safety maps for CS+DA+PF ($cmin = 5$), CS+OG+PF, SS+OG+PF, and, ES+OG+PF (all for $decr = -1$) algorithms. Row 1 shows corresponding parts of the ground truth safety maps. Columns 1 to 3: Safety maps for ACESRAMP, BIORAMP, and, ACES2302 datasets. Color Scheme: white for "Level"; yellow for "Inclined"; black for "Non-ground"; and grey for "Unknown" regions. Outside circle regions are "Unexamined". SS+OG+PF and ES+OG+PF fail to detect the drop-off in the ACES2302 environment.

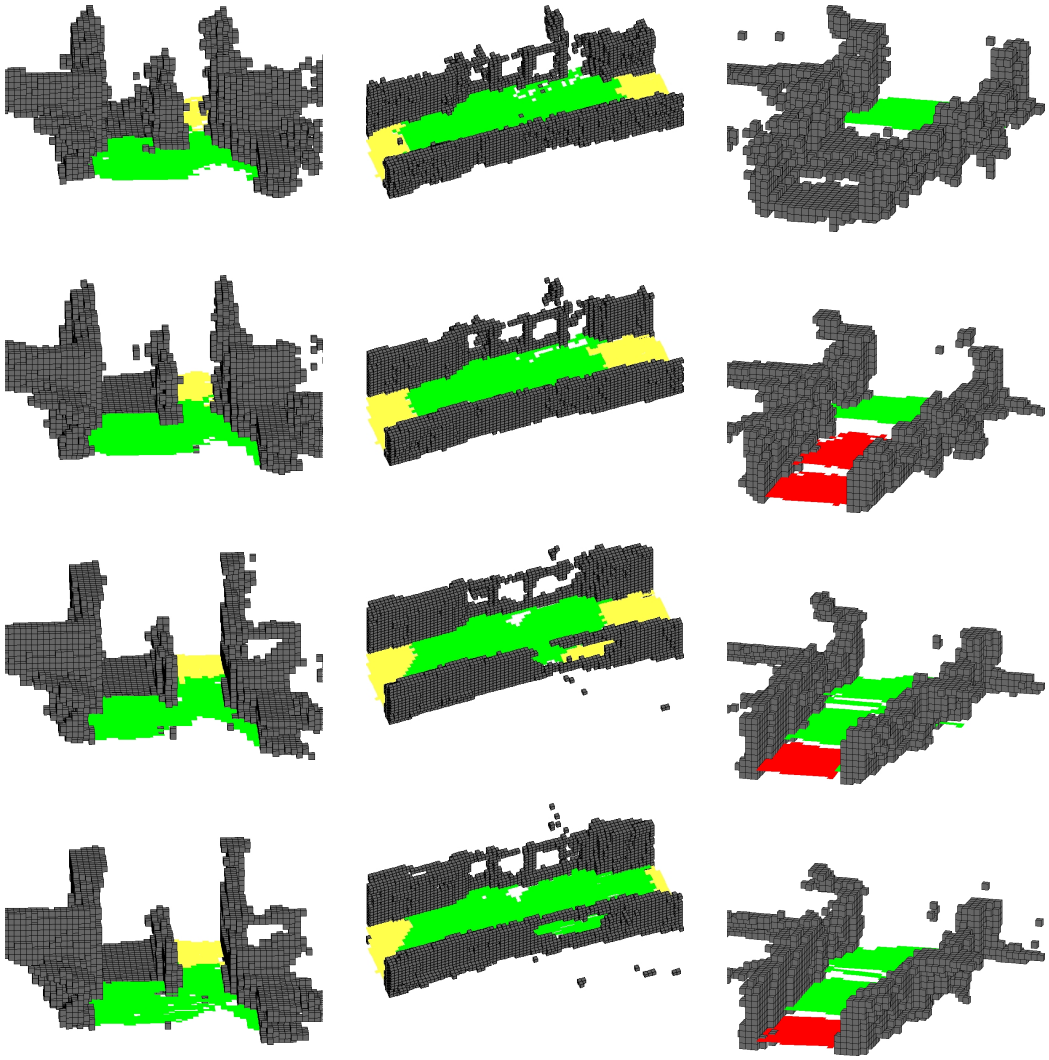


Figure 8.4: **3D Hybrid Maps** corresponding to the safety maps from Figure 8.3. Rows 1 to 4: Hybrid maps for CS+DA+PF ($cmin = 5$), CS+OG+PF, SS+OG+PF, and ES+OG+PF (all for $decr = -1$) algorithms. Columns 1 to 3: Safety maps for ACESRAMP, BIORAMP, and, ACES2302 datasets. Color Scheme: green for "Level" planes; yellow for "Inclined" planes; red and grey for "Non-ground" regions (where we use red for unsafe planes and grey for obstacles). SS+OG+PF and ES+OG+PF incorrectly detect unsafe planes as safe for the ACES2302 environment.

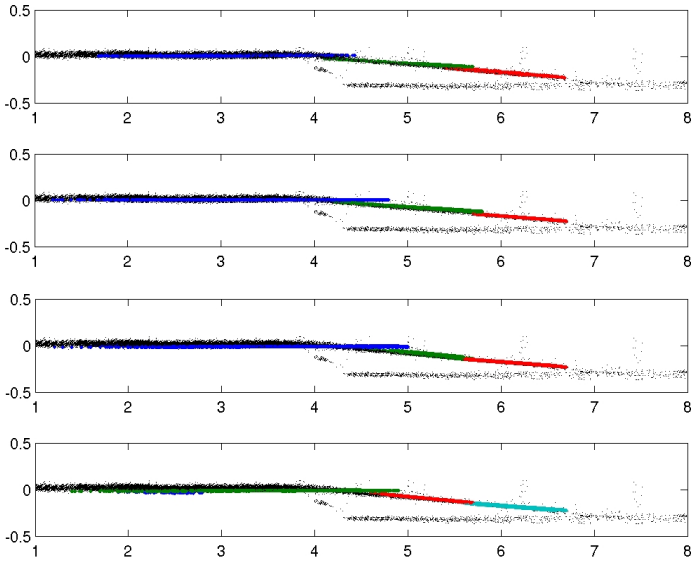


Figure 8.5: A cross-sectional view of the planes found (in color) for the ACESRAMP environment for the four algorithms, CS+DA+PF, CS+OG+PF, SS+OG+PF, and ES+OG+PF (in order from the top), corresponding to the safety maps in Figure 8.3. The planes found are compared to laser range data shown as black dots to show the quality of the fit.

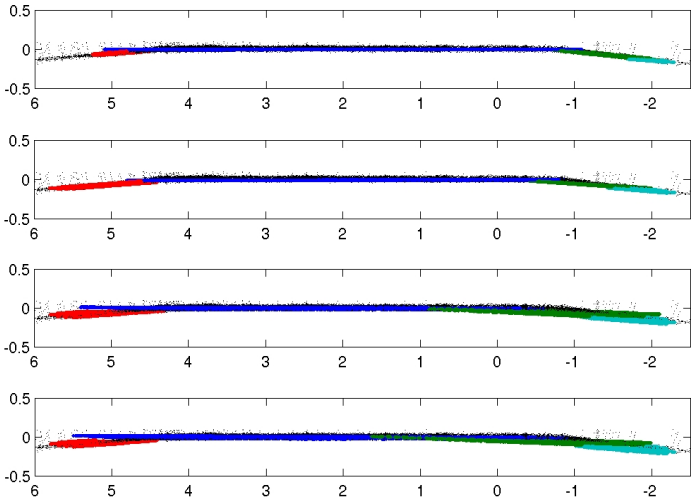


Figure 8.6: Planes (in color) found for the BIORAMP environment for the four algorithms, CS+DA+PF, CS+OG+PF, SS+OG+PF, and ES+OG+PF (in order from the top), corresponding to the safety maps in Figure 8.3.

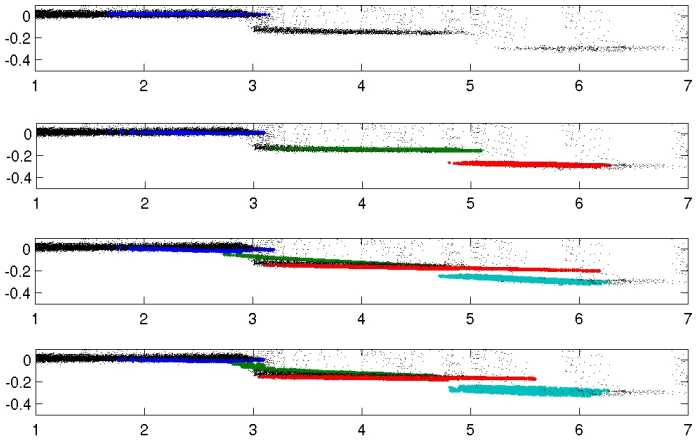


Figure 8.7: Planes (in color) found for the ACES2302 environment for the four algorithms, CS+DA+PF, CS+OG+PF, SS+OG+PF, and ES+OG+PF (in order from the top), corresponding to the safety maps in Figure 8.3. This figure shows why SS+OG+PF and ES+OG+PF fail to detect the drop-off in this environment. In both cases incorrect range readings just beyond the first drop-off edge on the left result in a incorrect inclined plane being found (in green). This plane causes the drop-off to appear as a traversable incline.

Comparison between step (ii) stereo algorithms, DA and OG. Based on the results (Tables 8.1 and 8.2 and Figures 8.2 and 8.8) of CS+DA+PF and CS+OG+PF we make the following observations.

- The data association method, DA, has a very low FN rate for all parameter values, lower than OG and in fact the lowest amongst all algorithms that we evaluate.
- However, DA has a fairly high FP rate.
- The 3D grid models built using the DA method look noisier than those built using the OG method in Figure 8.8. This is also true of the safety maps and hybrid 3D models.

The data association method is very promising in that it gives very low false negative rates for all parameter values. However, the relatively high false positive rates can be an issue in navigating narrow areas as they lead to thick walls that might make it impossible for the robot to move through such areas (see the hybrid map for the ACESRAMP environment in

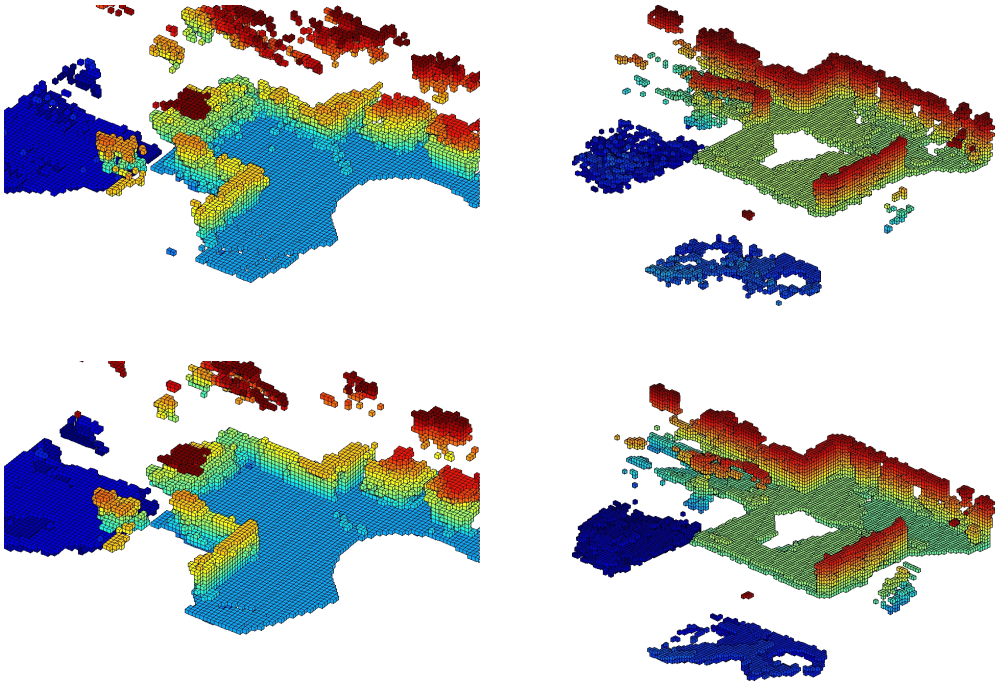


Figure 8.8: The 3D grids obtained as an intermediate step in the CS+DA+PF (top) and CS+OG+PF (bottom) algorithms for the OSCAFE (left) and NANORAMP (right) datasets. These images show qualitatively the difference between the data association, DA, and occupancy grid, OG, methods of Chapter 5 with the data association models appearing to be noisier. The hole in the center of the NANORAMP environment is due to the camera not seeing that area when exploring the environment.

Figure 8.4 created using CS+DA+PF). We believe the reason for the high FP rate of the DA method is the lack of a mechanism for incorporating negative (freespace) evidence while building 3D models (unlike the occupancy grid method OG).

In comparison, the CS+OG+PF method has higher FN rates than the CS+DA+PF method, but for one particular parameter setting ($decr = -1$) the CS+OG+PF method provides a fairly low FN rate while at the same time providing a significantly lower FP rate than the CS+DA+PF method. For this particular parameter setting, we believe that the CS+OG+PF algorithm provides the best compromise between competing requirements on the mapping system. We evaluate the CS+OG+PF method further in the next section.

A final point that we would like to make based on the constructed safety maps is the need for active sensing. The safety maps of Figure 8.3 have several cells in the middle of safe regions that are marked "Unknown". Such cells could be removed if the mapping algorithms were to have more information available about those areas. On a real robot it is possible to provide this extra information through the use of active sensing. The robot can move its camera to point at regions where it requires more information, thereby allowing it to build better maps and navigate more safely. The use of active sensing will also help in handling false positives, since they tend to evaporate as more information becomes available to the robot. In the real-time implementation of the CS+OG+PF method (Section 8.5) we simply have the camera pan left to right from time-to-time (a very primitive form of active sensing) which nevertheless helps the robot in building better maps.

8.3 Evaluation Part 2: Further Evaluation of CS+OG+PF

We provide two more quantitative evaluations of the CS+OG+PF algorithm, for the parameter setting of $decr = -1$, to help identify its capabilities and limitations better.

8.3.1 Plane Fitting Accuracy

We evaluate the accuracy of the planes that are found using the CS+OG+PF algorithm by comparing them against ground truth planes. The ground truth planes were obtained by manually segmenting the laser-range finder data and fitting planes to it. We compute two error measures.

1. *Angle between normals*: The angle between the normal of a detected plane and the normal of the corresponding ground truth plane.

2. *Average distance*: The average distance between points on the detected plane and the ground truth plane.

The average distance is computed as follows. The detected plane is obtained by fitting a plane of the form, $z = b_1 x + b_2 y + b_3$, to several stereo range data points. For each such range point, say $\mathbf{x}_i = (x_i, y_i, z_i)^T$, we re-compute the point's z-coordinate obtained using the fitted plane parameters: $\hat{z}_i = b_1 x_i + b_2 y_i + b_3$. We then compute the perpendicular distance of $\hat{\mathbf{x}}_i = (x_i, y_i, \hat{z}_i)^T$, to the ground truth plane for all range points i on the detected plane. The average of these distances gives the required distance.

From each of the five data sets, we randomly choose 5 image frames and compute the above two measures for all "Level" and "Inclined" planes detected in those frames. We average across all planes and all data sets to get the overall average and standard deviation values for the angle and average distances shown in Table 8.3.

Measure	Angle between normals (degrees)	Average distance (cm)
CS+OG+PF	1.2 ± 0.5	1.9 ± 0.8

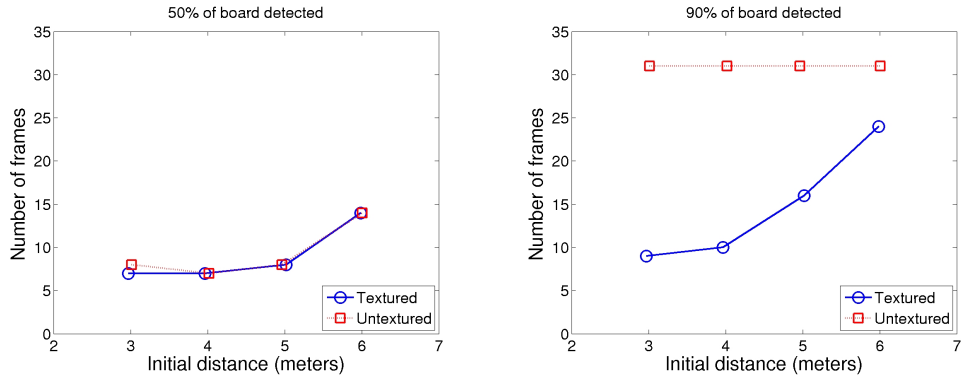
Table 8.3: Plane Accuracy: The overall averages and standard deviations of the angle between normals and of average distances are shown.

The low values of these error measures shows that plane fitting works very well and that we are able to accurately estimate both the normals and locations of safe planes. Figures 8.5, 8.6, and, 8.7 show examples of fitted planes.

8.3.2 Latencies in Detecting Objects

The use of the occupancy grid for noise filtering leads to latencies in hazard detection. However, such filtering is necessary because in its absence the robot may be too "paralyzed with fear" to move because of a high number of false positives. To evaluate the effect of filtering we compute the *frame latency*, defined as the number of camera frames between the appearance of an object in a video sequence and its detection by the robot. Knowing the frame latency is important for a mapping algorithm as it allows constraints on the robot's speed to be computed (using the robot's estimated stopping distance at various speeds).

The value of frame latency depends on object properties, in particular on the amount



(a) (b)

Figure 8.9: **Frame Latency**. (a) Plot showing number of frames that go by, before 50 percent of (the width of) a board is visible in the robot's occupancy grid, as a function of the initial distance to the board and the board's texture. Both textured and untextured boards are detected in about the same number of frames. (b) Plot showing frame latency for the case when we re-define detection as being when 90% of the board is visible. In this case the robot reaches the untextured board before 90% of it is visible - hence we plot the maximum value of frame latency for this case.

of texture present on the object, and the initial distance of the object from the robot. To measure frame latency as a function of these properties, we drive the robot towards two boards, one textured and one without texture, placed at various initial distances from the robot. For each board, we count the number of frames between the board's first appearance in the video sequence and its detection in the occupancy grid. To make the notion of detection concrete, we defined it to be the event when a certain percentage of the width of the board is visible in the occupancy grid. The results are plotted in Figure 8.9 for two different detection events, one when 50% of the board is visible, and the other when 90% of the board is visible.

As Figure 8.9(a) shows, when the board is at an initial distance of 5 meters or less, the robot is able to detect 50% of both textured and untextured boards within 8 frames. It is surprising that we get similar results for textured and untextured boards. However, when we re-define detection as being the event when 90 percent of the width of the board is visible

(Figure 8.9(b)), it takes significantly more time for the untextured board to be detected. In fact, the robot reaches the board before 90% of it is visible [6].

It is worth noting that as the initial distance to the board increases to more than 4 meters in Figure 8.9(b), for the textured board, the number of frames taken to detect the board jumps. This means that it can take a long time before objects at a distance of more than 4 meters are seen properly with our stereo camera. This provides an experimental justification for the use of a restricted planning radius in Section 6.1 as analysis of objects far away is going to be unreliable.

8.4 Relationship between Error Rates and Path Safety

In Section 8.2 we mentioned that a particular false negative (FN) rate does not translate directly into the chances of a robot having an accident. In this section we present a *preliminary* analysis of what false negative rates mean for the possibility of unsafe travel by a robot when it is planning paths using the safety map. In addition we also attempt to understand how much false positive (FP) errors can hinder path planning.

To do the analysis, we have a robot plan thousands of paths between several pairs of starting and goal points in simulated safety maps. Then we find the number of cases for which unsafe paths are found, no paths are found, and safe paths are found. A simulated safety map is obtained by sampling from a ground truth safety map: we randomly convert unsafe cells into safe cells and safe cells into unsafe cells in the ground truth safety map.

To create simulated safety maps that approximate real safety maps well enough, we need to model how FN and FP errors occur. Figure 8.10 shows a safety map for the BIORAMP environment with FN and FP errors highlighted. We can see that FN and FP errors mostly occur at the boundaries between safe and unsafe regions. Therefore when sampling we only flip cells that occur at the boundaries between safe and unsafe regions.

[6] This sounds like it can be a bit dangerous for the robot, but what happens is that different parts of the untextured board are detected, making it appear as an impassable obstacle nevertheless.

Figure 8.10: Safety map of the BIORAMP environment showing false positive cells in green and false negative cells in red. Most of these cells occur on the boundary between safe and unsafe regions.

Error Rates (%) → Algorithms ↓	FN	FNB	FP	FPB
CS+DA+PF ($cmin = 5$)	1.4	6.9	15.6	67.2
CS+OG+PF ($decr = -1$)	3.0	9.9	9.3	53.7
CS+OG+PF ($decr = -3$)	8.5	21.6	6.2	38.4

Table 8.4: False negative (FNB) and false positive (FPB) error rates in the boundary regions of safety maps for different algorithms. Also shown are the corresponding FN and FP error rates of the algorithms for the safety maps as a whole. The leftmost column lists the algorithm and parameter setting for which the FP and FN rates were obtained (see Tables 8.1 and 8.2).

Furthermore, since we only flip cells in boundary regions we cannot use the FN and FP rates that we have computed for the safety maps as a whole. We have to instead compute FN and FP rates for boundary regions. Table 8.4 shows the boundary FN and FP errors we get for the three pairs of FN/FP error rates that we consider in this analysis. The three pairs of error rates correspond to actual error rates that we get for some of our safety map algorithms. As we can see from Table 8.4, the FN error rates in boundary regions are roughly 2-5 times greater than the overall FN error rates. Similarly, the boundary FP rates are about 4-6 times greater than the overall FP rates.

Another observation we make from Figure 8.10 is that the FN and FP cells occur

in clumps indicating that their occurrences are correlated. However, in the analysis here, we assume that the FN and FP cells are independently and identically distributed (I.I.D) for simplicity. While this assumption may seem unrealistic it is in fact a somewhat reasonable assumption to make when FN and FP errors are either high or low. For the FN and FP errors that we consider in Table 8.4, we can see that the (boundary) FN errors are mostly low whereas the (boundary) FP errors are mostly high. Therefore, even with the I.I.D. assumption, we hope that we can still identify general trends – fulfilling the purpose of this preliminary analysis.

In the analysis, we create 200 simulated safety maps for a particular FN/FP error rate pair from the ground truth safety maps of each of our 5 datasets to give a total of 1000 simulated safety maps. We also randomly pick 5 starting and 5 goal points for each ground truth map. We then plan paths between every pair of start and goal points (25 pairs in all) for all simulated maps to get a total of 25000 (=1000x25) different start/goal pairs between which paths can possibly be found.

We assume the robots, for which we plan paths, are cylindrical. We plan paths for two different values of the robot's radii - 20cm and 30cm. We assume that both robots keep a 10cm margin of safety all around them. A robot with a radius of 30cm is a fairly large robot for the environments considered in our datasets. For example, the ramp in the ACESRAMP environment is about 80-90cm wide giving such a robot very little room to maneuver. As a result we expect FN and FP errors to have a greater effect on the safety of the 30cm robot as compared to the 20cm robot.

We use Dijkstra's shortest path algorithm [Wikipedia, 2009a] for planning paths. Once paths have been planned for all possible start/goal pairs we compute the following numbers: (i) **UF**: Percent of pairs for which unsafe paths are found. (ii) **NF**: Percent of pairs for which no paths are found. (iii) **SF**: Percent of pairs for which safe paths are found.

Tables 8.5 and 8.6 show the results we get for the three FN and FP rates that we consider for the two robot radii. In addition to the three pairs of FN/FP error rates we also

Error Rates (%) → Comments ↓	FN	FP	UF	NF	SF
Both false negative & false positive errors present	1.4	15.6	0.00	3.62	96.38
	3.0	9.3	0.00	3.00	97.00
	8.5	6.2	0.01	2.40	97.59
Only false negative errors present	1.4	0.0	0.00	0.00	100.00
	3.0	0.0	0.00	0.00	100.00
	8.5	0.0	0.04	0.00	99.96
Only false positive errors present	0.0	15.6	0.00	3.62	96.38
	0.0	9.3	0.00	3.00	97.00
	0.0	6.2	0.00	2.40	97.60

Table 8.5: Error rates and path safety for a 20cm radius cylindrical robot.

consider the cases when either FN or FP errors are zero. We do this to see the effect that FP errors can have on path safety. We make the following observations based on Tables 8.5 and 8.6.

- For the 20cm robot, the percent of unsafe paths is at least two orders of magnitude lower than the percent FN rate. The percent of cases for which no paths are found are several times lower than the percent FP rate.
- For the 30cm robot, the percent of unsafe paths found are about two orders of magnitude lower than the percent FN rate when FP errors are also present. However, when no FP errors are present, the percent of unsafe paths is about one order of magnitude lower. Thus FP errors actually help the robot by reducing the percent of unsafe paths. However, too many FP errors can be a hindrance to navigation as we see next.
- For the 30cm robot, the percent of cases for which no paths are found are quite high. This is expected because of the large size of the robot and also because the environments we consider have narrow regions. Since we randomly pick start and goal points, several of these points fall in the narrow regions meaning that paths cannot be found for the 30cm robot.
- FN errors have almost no effect on the percent of cases for which no paths are found.

Error Rates (%) → Comments ↓	FN	FP	UF	NF	SF
Both false negative & false positive errors present	1.4	15.6	0.00	35.04	64.96
	3.0	9.3	0.03	33.09	66.88
	8.5	6.2	0.07	29.10	70.83
Only false negative errors present	1.4	0.0	0.24	0.00	99.76
	3.0	0.0	0.49	0.00	99.51
	8.5	0.0	1.09	0.00	98.91
Only false positive errors present	0.0	15.6	0.00	35.04	64.96
	0.0	9.3	0.00	33.09	66.91
	0.0	6.2	0.00	29.10	70.90

Table 8.6: Error rates and path safety for a 30cm radius cylindrical robot.

While the model used in our analysis is not fully realistic it does make three trends stand out. First, for a low FN rate, the percent of unsafe paths are in general significantly lower (one to several orders of magnitude lower) than the percent FN rate. Second, a high FP rate can be a hindrance to navigation in narrow regions for large robots. Third, robot size has a significant effect on how safely and easily a robot is able to move about. Thus, the analysis gives more meaning to the FP and FN rates that we calculated for our various algorithms. For example, we now know that a FN rate of 1% means that a large robot has roughly a one in a thousand chance of planning a dangerous path.

There are numerous ways in which this preliminary analysis can be extended. The most obvious is having a better model for creating simulated safety maps. Another way to extend the analysis is by understanding how FP and FN errors change with distance. Qualitatively, we observe that FP and FN errors are lower in regions close to the robot. Thus, simply planning a dangerous path does not necessarily mean the robot will have an accident. As the robot follows the path and gets closer to dangerous regions, better stereo data will help the robot identify the regions as such and avoid them. We expect that if we take the effect of distance into account in the above analysis we should see significant reductions in both the percent of unsafe paths and percent of cases in which no paths are found.

8.5 Real-time Implementation of CS+OG+PF and Integration with the HSSH

A real-time version of the CS+OG+PF algorithm was implemented and integrated with the Hybrid Spatial Semantic Hierarchy [Murarka et al., 2009]. We present results showing safety maps of several environments built successfully by the real-time algorithm and the use of these safety maps for motion planning and extracting local topological structures.

The individual components of the real-time CS+OG+PF algorithm are as described in Chapters 4 to 6. The main differences between the CS+OG+PF algorithm tested in the previous sections and the real-time version are as follows.

1. The real-time version uses *mean height* in addition to the *indexed height* to segment the 3D model, as described in Section 6.2. This results in fewer and larger segments being found and increases the speed of the algorithm.

2. The real-time version uses an *optional* post-processing step whereby small "Non-ground" and "Unknown" regions that appear as *islands* within larger "Level" regions in the safety maps are re-labeled as "Level". This did not seem to unduly increase the instances of false negatives in the environments that we tested and helped get rid of random false positive cells at some distance from the robot [7].

3. In algorithm 2, we set $n_b = 1$ for the real-time CS+OG+PF version as opposed to 5 for the version evaluated above. We also used a parameter setting of $decr = -3$ for the real-time version.

The above changes (specially those in bullet 2) were necessitated by the limited field of view our stereo camera (about 45^o) and limited processing power available to the robot. To overcome the rather limited field of view of the camera, we occasionally had the camera pan from side-to-side to gather more information about the robot's surroundings.

[7] The false positive cells could have been taken care off in a more principled way through the use of active sensing.

The real-time algorithm was able to run at a minimum cycle rate of 4 Hz (with bursts of upto 9 Hz) when building local safety maps of size 10m$\times$10m on a 1.83 GHz dual-core laptop with several other components of the HSSH code base running simultaneously. The correlation stereo algorithm and the Player robot server [Player/Stage, 2009] ran on a separate computational backpack on the robot.

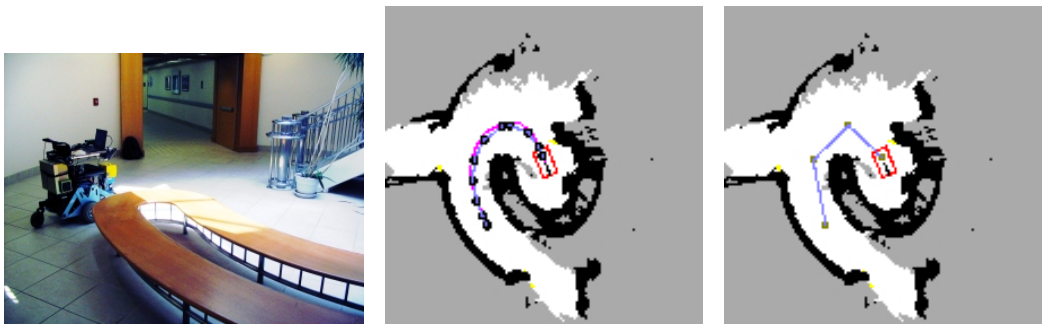
8.5.1 Motion Planning using Safety Maps

The HSSH motion planning module consists of a path planner and a trajectory generator. The planner used is a Rapidly-exploring Random Tree (RRT) planner [Kuffner and LaValle, 2000] that finds safe piecewise linear paths in space. The planner works with discrete representations such as 2D grids, allowing it to be used in a straightforward manner with the safety maps. The trajectory generator [Gulati et al., 2009] uses the output of the RRT planner to generate velocity and acceleration profiles (aka trajectories) for the wheelchair to follow. The trajectories smooth the paths returned by the planner while at the same time ensuring comfortable motion for the wheelchair user.

Figure 8.11 illustrates the successful integration between the HSSH motion planner and the safety map algorithm. Figure 8.11 also illustrates the advantage of using vision to build the safety maps – the bench in Figure 8.11(a) has an overhang that is too high to be seen by the horizontal laser range-finder on the wheelchair. However, the stereo camera is able to detect the bench and represent it in the safety map (Figure 8.11(b)) allowing the robot to avoid it. Figure 8.13 shows another instance of the motion planner being used successfully with safety maps.

8.5.2 Extracting Local Topology from Safety Maps

One of the main steps in the HSSH is that of extracting the local topology of *places* in the environment (Section 2.2.3). Knowing the local topology allows the HSSH to build large scale maps of the environment.



(a) (b) (c)

Figure 8.11: (a) The Intelligent Wheelchair following a trajectory in an indoor environment. (b) The safety map of the environment showing the wheelchair at its destination after having successfully followed the planned trajectory. The planned trajectory is shown in blue and black and the path followed by the wheelchair is shown in pink. (c) The same safety map showing the piecewise linear path planned by the RRT planner. We can see that the trajectory is a smoothed version of this path that also specifies velocity and acceleration profiles for the robot to follow.

Currently, the HSSH extracts local topologies from laser-based 2D grid maps called local perceptual maps (LPMs). Since the vision-based safety maps are also represented using 2D grids we are able to use them for extracting local topologies. The LPMs classify space into three categories: "Obstacles", "Free", and, "Unknown". We can convert safety maps into LPMs, by mapping "Level" and "Inclined" regions in the safety maps to "Free" regions, "Non-ground" regions to "Obstacles", and "Unknown" regions to "Unknown" regions. These vision-based LPMs can then be used by the HSSH to extract local topologies.

Figure 8.12 illustrates the *gateways* (Section 2.2.3) found using the vision-based LPM for a particular environment. The figure also shows how the *places* and their local topology determined using the vision-based LPM differ from the *places* and topologies determined using the laser-based LPM.

Lastly, Figure 8.13 shows an example of the integration of the safety mapping algorithm with both the motion planner and local topology algorithms. The figure shows an outdoors sidewalk that the wheelchair can navigate using the vision-based LPM but cannot

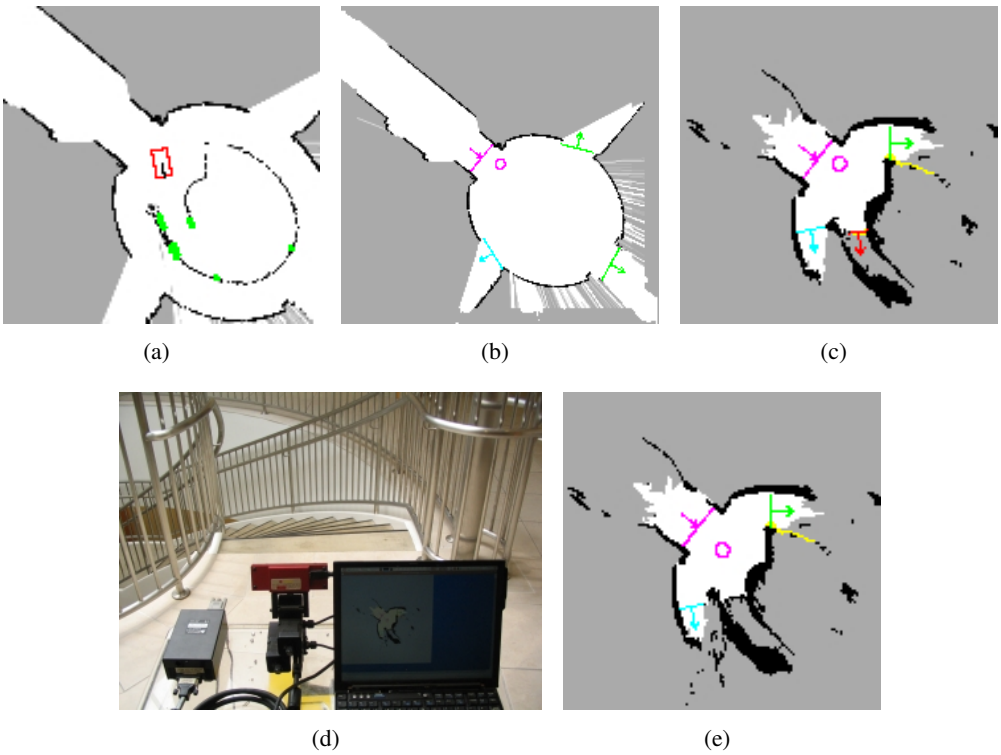
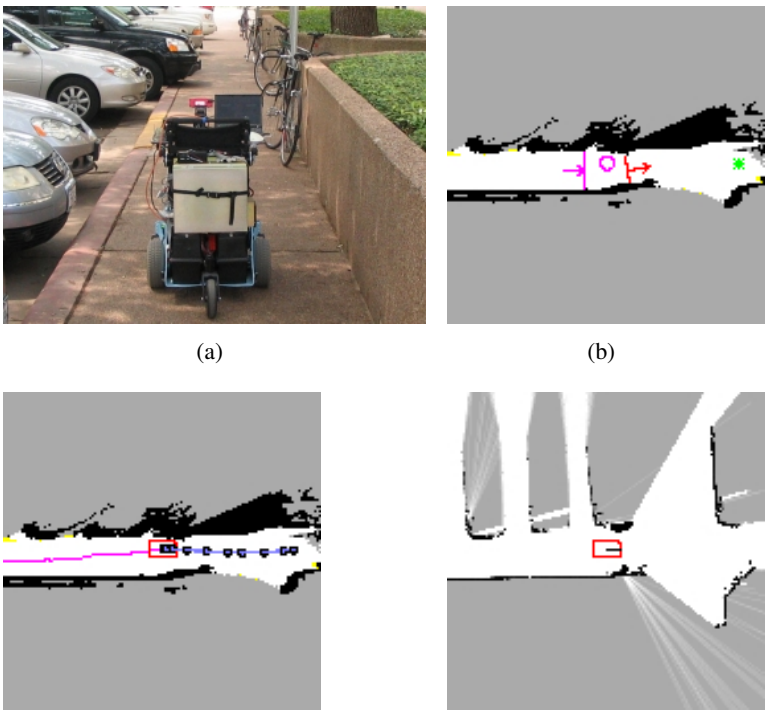


Figure 8.12: (a) A laser-based LPM showing the wheelchair robot in a large open intersection with a railing and a staircase in the middle (the green blobs represent dynamic obstacles, which occur due to the lasers not seeing the railing poles consistently). (b) The four *gateways* found using the laser-based LPM. The *gateway* algorithm removes "island" obstacles (in this case the railing poles) resulting in the gateways shown. The robot identifies the region as a *place* with a + intersection. (c) When using the vision-based LPM, the robot finds four very different *gateways* in the same region. This is because the stereo camera does a better job of detecting the railing poles than the laser range-finder. (d) On closer examination, the robot detects a drop-off due to a downward staircase using the stereo camera. (e) Upon detecting the drop-off, the *gateways* found using the vision-based LPM are updated and the robot identifies the place as a Y intersection. The laser-based LPM does not see the drop-off, which could be catastrophic.

navigate using the laser-based LPM.



(a) (b) (c) (d)

Figure 8.13: Integration of the vision-based LPM with motion planning and local topology. (a) The wheelchair travels down a cluttered sidewalk. (b) The wheelchair is able to detect the cars and the sidewalk drop-off edge on the left and represents these in its safety map. This allows the local topology algorithm to correctly identify the sidewalk as a *path* with two *gateways*. (c) The wheelchair plans safe trajectories using the motion planner and is able to successfully navigate the sidewalk. (d) The laser-based LPM misses the drop-off edge on the left and this results in an unsafe map for the robot.

8.6 Summary

In this chapter we evaluated several different algorithms for building local safety maps using stereo vision and provided a comparison of the algorithms. In particular we showed the following.

- Amongst the three algorithms for computing stereo depth maps, the correlation stereo algorithm, CS, gave the lowest false negative error rates whereas the color segmenta-

tion stereo algorithm, SS, gave the lowest false positive error rates.

- The data association method, DA, for building 3D models gave by far the lowest false negative error rates amongst all mapping methods. However, it also had the highest false positive rates.

- The safety mapping algorithm, CS+OG+PF, based on correlation stereo and the occupancy grid method, was deemed to give the best compromise between competing requirements on the mapping algorithms. The CS+OG+PF was also shown to find accurate planes.

We did a preliminary analysis to evaluate what the false positive and false negative rates mean to a robot in terms of planning unsafe paths or finding any paths at all. We showed that, at the least, the chance of a robot planning unsafe paths is about one order of magnitude lower than the corresponding percent false negative rate.

We also implemented a real-time version of the CS+OG+PF algorithm and showed it working successfully on a physical robot supporting safe motion planning and large scale map building. Finally we believe the evaluation methodology introduced in this chapter, consisting of finding error rates, measuring plane accuracy and detection latency, is very general and can be used for comparing the performance of various vision-based metric mapping systems.

Chapter 9

Conclusion

9.1 Summary

The major contribution of this work is the idea of the safety map for safe local robot navigation and ways to create the safety map from sensor input. The local safety map captures in 2D the information required by wheeled mobile robots to navigate safely and autonomously in 3D urban environments. We have focussed on using vision as the robot's sensory input because of the very large and varied amount of information that vision provides and also because it has the potential to be a ubiquitous robot sensor.

We have presented several vision-based algorithms for constructing safety maps that detect and model different hazards present in urban environments. The hazards we focus on include static obstacles, overhangs, drop-off edges, and inclines. Where possible we have drawn upon existing methods in the literature [Videre Design, 2006; Hong and Chen, 2004; Konolige, 1997] for the algorithms, and where necessary we have introduced our own algorithms [Murarka et al., 2006, 2008; Murarka and Kuipers, 2009b; Murarka et al., 2009].

The algorithms for constructing the safety maps consist of three main steps: (i) computing stereo depth maps, (ii) building 3D models, and, (iii) analyzing the 3D models for safety. The stereo depth computation algorithms that we have introduced in this work are

the color-based and edge-based segmentation stereo algorithms. We also discuss a standard correlation-based stereo algorithm.

For building 3D models we introduced a probabilistic data association method that works with both sparse and dense stereo range data. We also implemented a 3D occupancy grid method. The main algorithm we introduce for safety analysis consists of a fast segmentation method for identifying potentially traversable ground regions in a 3D grid, followed by fitting planes to the segments and then analyzing the planes and segments to produce the final safety map. We also construct a 3D hybrid map that uses planes to represent traversable ground surfaces and a grid to represent non-ground regions.

In addition to the above stereo based methods for detecting hazards, we also introduced a stand-alone motion and occlusion based method for detecting drop-off edges perpendicular to the robot's direction of motion. The method works on a stream of monocular images and its results can be used to augment the safety maps produced using stereo methods.

We have presented qualitative results for all the algorithms showing that they are able to detect obstacles, drop-offs, and ramps and produce good quality safety maps. We have also quantitatively evaluated the various algorithms on several datasets and provided a comparison of their performance. In particular we showed the following.

- Amongst the stereo depth computation algorithms, the correlation stereo based method gave the lowest false negative rate whereas the color segmentation stereo method gave the lowest false positive rate.

- Amongst the 3D model methods, the probabilistic data association method gave very low false negative rates whereas the occupancy grid method gave lower false positive rates.

- The correlation stereo algorithm combined with the occupancy grid method and the plane fitting algorithm gave the best overall algorithm, CS+OG+PF, providing the best compromise between the false negative and false positive rates.

We also do a preliminary analysis to understand what the error rates mean to a robot in terms of planning. We show that, at the least, the chance of a robot planning unsafe paths is about an order of magnitude lower than the corresponding percent false negative rate.

We implemented a real-time version of the CS+OG+PF algorithm and showed it working successfully with motion planning algorithms and a large scale mapping method (the Hybrid Spatial Semantic Hierarchy). We plan to make the evaluation code, datasets, and the corresponding ground truth data, publicly available so as to provide a common testing ground for other robot mapping systems [Murarka and Kuipers, 2009a].

We believe that with this work we have a taken a significant step towards achieving our goal of building robots that can navigate safely using vision-based sensors. Also by providing an evaluation framework to the community we hope to provide an objective way for comparing different mapping algorithms and helping the field move forward.

9.2 Future Work

There are several ways in which the work presented here can be extended. We discuss a few of these extensions.

Detecting Additional Hazards. To navigate successfully it is important that the robot learn to identify and track dynamic obstacles [Modayil and Kuipers, 2004]. A combination of image based and stereo based tracking will probably be required to detect dynamic obstacles. This is due to the presence of false positive range readings in stereo data that can be difficult to filter in the short time periods available when tracking dynamic objects.

An interesting challenge to navigating autonomously in urban environments is the presence of transparent glass doors and walls. It is extremely difficult to detect such glass surfaces using cameras and will require the use of sophisticated top-down processing. It seems that using additional sensors such sonar and bump sensors will be necessary for the near future.

Using Non-geometric Cues for Classification. In the current work we have mostly used geometric information about objects in the world to classify them as safe or unsafe (except in Chapter 7). As mentioned in Section 2.3.1, there are instances where geometric information is clearly not enough. As an example, consider a flat sidewalk with a dirt shoulder that is also flat but not traversable from the robot's perspective – geometry alone cannot distinguish between safe and unsafe surfaces in this instance. The use of additional visual cues such as texture and color [Ulrich and Nourbakhsh, 2000] will be important for handling such cases. Learning from experience will also play an important role in this kind of inference since it won't be possible to program-in this kind of knowledge beforehand.

Better Stereo Methods. The stereo methods that we have presented in this work are fairly good but it is clear that they can be improved upon. We believe that one direction that needs to explored further in the domain of stereo methods, are methods that are a mix of local and global techniques. While optimizing an objective function over the whole image makes sense, it seems that it would be better to optimize different objective functions over different regions of the image taking the characteristics of those regions into account. For example, in a region with good texture it makes sense to give less weight to the smoothing term whereas in regions of low texture the smoothing term should have a higher weight.

6-DOF Localization. There has been much progress in visual localization methods in recent times [Comport et al., 2007] and a useful extension to the current work would be the addition of a vision-based 6 DOF localization module.

Incorporating Negative Evidence. The probabilistic data association method of Chapter 5 does not use free space (or negative) evidence like the occupancy grid method does. This is probably why it has a high false positive rate, because, once a false positive range reading has been added to the map there is no way to remove it. On the other hand, the low false negative rates it provides consistently across different parameter settings makes it

an attractive method, worth extending to incorporate negative evidence. One way to do so might be to enforce a "soft" visibility constraint utilizing the uncertainties associated with landmark locations.

Learning Sensor Models. For both the occupancy grid method and the probabilistic data association method, we use hand tuned sensor noise models. Ideally, however, we would like to learn both the parameters and the form of the sensor models [Stronger and Stone, 2008].

Fitting Surfaces to Non-ground Regions. One way to reduce the effect of stereo noise would be to fit planes (or other surfaces) to non-ground regions in addition to traversable ground regions as we currently do. However, this is a much harder problem as obstacles and other objects in the environment can have complex topologies. For example, the rails next to the ramps in the ACESRAMP and TAYLORRAMP datasets can be very hard to model by fitting surfaces to 3D point cloud models.

Incorporating Stronger Constraints on Robot Capabilities. When analyzing planes and segments for safety, we look at height differences between adjacent planes to determine if one plane is reachable from another. We currently assume that if the robot is capable of going from plane A to plane B it is capable of going from plane B to A. However, this may not be necessarily true for all robots – some robots might find it easier to go downhill as opposed to travelling uphill – making it necessary to take into account such kinds of asymmetrical capabilities in robots when building safety maps. This might require extending our current safety map representation to include annotations indicating the direction of travel possible in different regions of the map. Thus, in the future, we would like to incorporate stronger constraints on robot capabilities, such as asymmetric motion capabilities, when building safety maps.

Active Sensing. Active sensing has the potential to further reduce false positive and false negative error rates and in general, the effects of stereo noise. The limited field of view of the camera and limited processing power available means that a robot needs to choose carefully where it should look (such as focussing attention on unknown regions that are in the robot's current path). Information theoretic techniques can be used to select pan and tilt actions that should be applied to the camera based on the robot's safety map and 3D model of its surroundings.

Understanding Error Rates. We would like to extend the analysis presented in Section 8.4 relating error rates to the safety of the paths planned. Some ways to do so, including the two ways mentioned earlier in Chapter 8, are as follows:

- Having a better model for creating simulated safety maps.
- Understanding and modeling how false positive and false negative errors change with distance to objects. We expect the error rates to decrease with distance meaning that the percent of expected unsafe paths for a given percent FN error rate (and percent of cases in which paths are not found for a given percent FP error rate) should be significantly lower than what we have computed.
- Doing the analysis for several different kinds of robots with different motion capabilities (holonomic vs. non-holonomic) and different dimensions.
- Doing the analysis for several path planning methods that work for the different kinds of robots. We would like to understand the true safety of paths planned by robots not only as a function of the error rates and but also as a function of the robot's navigation capabilities.

Bibliography

A. Akbarzadeh, J.-M. Frahm, P. Mordohai, B. Clipp, C. Engels, D. Gallup, P. Merrell, M. Phelps, S. Sinha, B. Talton, L. Wang, Q. Yang, H. Stewnius, R. Yang, G. Welch, H. Towles, D. Nister, and M. Pollefeys. Towards Urban 3D Reconstruction from Video. In *Third International Symposium on 3-D Data Processing, Visualization and Transmission*, 2006.

Y. Alon, A. Ferencz, and A. Shashua. Off-road Path Following using Region Classification and Geometric Projection Constraints. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2006.

S. Bagon. Matlab Wrapper for Graph Cuts, 2006. URL `http://www.wisdom.weizmann.ac.il/~bagon`.

P. Beeson, N. Jong, and B. Kuipers. Towards Autonomous Place Detection Using the Extended Voronoi Graph. In *IEEE International Conference on Robotics and Automation*, 2005.

P. Beeson, A. Murarka, and B. Kuipers. Adapting Proposal Distributions for Accurate, Efficient Mobile Robot Localization. In *IEEE International Conference on Robotics and Automation*, 2006.

P. Bellutta, R. Manduchi, L. Matthies, K. Owens, and A. Rankin. Terrain Perception for Demo III. In *Proceedings of the 2000 Intelligent Vehicles Conference*, 2000.

C.M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

M. Bosse, P. Newman, J. Leonard, M. Soika, W. Feiten, and S. Teller. An Atlas Framework for Scalable Mapping. In *IEEE International Conference on Robotics and Automation*, 2003.

Y. Boykov and V. Kolmogorov. An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision. *Pattern Analysis and Machine Intelligence*, 2004.

Y. Boykov, O. Veksler, and R. Zabih. Efficient Approximate Energy Minimization via Graph Cuts. *Pattern Analysis and Machine Intelligence*, 2001.

W. Burgard, A.B. Cremers, D. Fox, D. Haehnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. The Interactive Museum Tour-Guide Robot. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 1998.

J. Byrne, M. Cosgrove, and R. Mehra. Stereo Based Obstacle Detection for an Unmanned Air Vehicle. In *IEEE International Conference on Robotics and Automation*, 2006.

J. Canny. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.

H. Choset and K. Nagatani. Topological Simultaneous Localization and Mapping (SLAM): Toward Exact Localization Without Explicit Localization. *IEEE Transactions on Robotics and Automation*, 17(2):125–137, 2001.

D. Comaniciu and P. Meer. Mean Shift: A Robust Approach Toward Feature Space Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:603–619, 2002.

A.I. Comport, E. Malis, and P. Rives. Accurate Quadrifocal Tracking for Robust 3D Visual Odometry. In *IEEE International Conference on Robotics and Automation*, 2007.

H. Dahlkamp, A. Kaehler, D. Stavens, S. Thrun, and G. Bradski. Self-supervised Monocular Road Detection in Desert Terrain. In *Robotics: Science and Systems*, 2006.

A.J. Davison. Real-Time Simultaneous Localisation and Mapping with a Single Camera. In *IEEE International Conference on Computer Vision*, 2003.

G.N. DeSouza and A.C. Kak. Vision for Mobile Robot Navigation: A Survey. *IEEE Transactions of Pattern Analysis and Machine Intelligence*, 24(2):237–267, 2002.

J. Diebel, K. Reutersward, S. Thrun, J. Davis, and R. Gupta. Simultaneous Localization and Mapping with Active Stereo Vision. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3436–3443, 2004.

M.W.M.G. Dissanayake, P. Newman, S. Clark, H.F. Durrant-Whyte, and M. Csorba. A Solution to the Simultaneous Localisation and Map Building (SLAM) Problem. *IEEE Transactions of Robotics and Automation*, pages 229–241, 2001.

R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*. Wiley-Interscience, 2000.

P.F. Felzenszwalb and D.P. Huttenlocher. Efficient Graph based Image Segmentation. *International Journal of Computer Vision*, 59(2):167–181, 2004.

P.F. Felzenszwalb and D.P. Huttenlocher. Efficient Graph based Image Segmentation (code), 2007. URL `http://people.cs.uchicago.edu/~pff/segment`.

M.A. Fischler and R.C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications of the ACM*, 24(26), 1981.

A. Fitzgibbon and A. Zisserman. Automatic 3D Model Acquisition and Generation of New Images from Video Sequences. In *Proceedings of the European Signal Processing Conference*, 1998.

D.A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach*. Prentice-Hall, 2002.

A. Gelb. *Applied Optimal Estimation*. MIT Press, 1974.

S. Geman and D. Geman. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *Pattern Analysis and Machine Intelligence*, 6:721–741, 1984.

B. Georgescu and C.M. Christoudias. Edge Detection and Image Segmentation System (EDISON), 2009. URL `http://www.caip.rutgers.edu/riul/research/code/EDISON/index.html`.

S. Gulati, C. Jhurani, B. Kuipers, and R. Longoria. A Framework for Planning Comfortable and Customizable Motion of an Assistive Mobile Robot. In *IEEE International Conference on Intelligent Robots and Systems*, 2009.

J.-S. Gutmann, M. Fukuchi, and M. Fujita. 3D Perception and Environment Map Generation for Humanoid Robot Navigation. *International Journal of Robotics Research*, 2008.

R. Hadsell, P. Sermanet, J. Ben, A.N. Erkan, J. Han, B. Flepp, U. Muller, and Y. LeCun. Online Learning for Offroad Robots: Using Spatial Label Propagation to Learn Long-Range Traversability. In *Robotics: Science and Systems*, 2007.

R. Hadsell, A. Erkan, P. Sermanet, M. Scoffier, U. Muller, and Y. LeCun. Deep Belief Net Learning in a Long-range Vision System for Autonomous Off-road Driving. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 628–633, 2008.

C. Harris. Geometry from Visual Motion. *Active Vision*, pages 263–284, 1993.

R.I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.

N. Heckman, J.F. Lalonde, N. Vandapel, and M. Hebert. Potential Negative Obstacle Detection by Occlusion Labeling. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007.

D. Hoiem, A.A. Efros, and M. Hebert. Automatic Photo Pop-up. In *ACM SIGGRAPH*, 2005.

D. Hoiem, A.A. Efros, and M. Hebert. Recovering Surface Layout from an Image. *International Journal of Computer Vision*, 75(1), 2007.

L. Hong and G. Chen. Segment-based Stereo Matching using Graph Cuts. In *Computer Vision and Pattern Recognition*, 2004.

A. Hoover, G. Jean-Baptiste, X. Jiang, P.J. Flynn, H. Bunke, D.B. Goldgof, K. Bowyer, D.W. Eggert, A. Fitzgibbon, and R.B. Fisher. An Experimental Comparison of Range Image Segmentation Algorithms. *IEEE Transactions of Pattern Analysis and Machine Intelligence*, 18(7), 1996.

A. Howard. Real-time Stereo Visual Odometry for Autonomous Ground Vehicles. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008.

A. Huertas, L. Matthies, and A. Rankin. Stereo-Vision Based Tree Traversability Analysis for Autonomous Off-Road Navigation. In *Proceedings of the IEEE Workshop on Applications of Computer Vision*, Breckenridge, CO, 2005.

K. Iagnemma and M. Buehler. Editorial for Journal of Field Robotics - Special Issue on the DARPA Grand Challenge. *Journal of Field Robotics*, 23(8):461–462, 2006.

L. Iocchi, K. Konolige, and M. Bajracharya. Visually Realistic Mapping of a Planar Environment with Stereo. In *International Symposium on Experimental Robotics*, 2000.

L.D. Jackel, E. Krotkov, M. Perschbacher, J. Pippine, and C. Sullivan. The DARPA LAGR Program: Goals, Challenges, Methodology, and Phase I Results. *Journal of Field Robotics*, 23(11-12):945–973, 2006.

X.Y. Jiang and H. Bunke. Fast Segmentation of Range Images into Planar Regions by Scan Line Grouping. *Machine Vision and Applications*, 2(2):115–122, 1994.

D. Kim, J. Sun, S. M. Oh, J. M. Rehg, and A. Bobick. Traversability Classification Using Unsupervised On-Line Visual Learning for Outdoor Robot Navigation. In *IEEE International Conference on Robotics and Automation*, 2006.

V. Kolmogorov and R. Zabih. Computing Visual Correspondence with Occlusions Using Graph Cuts. In *IEEE International Conference on Computer Vision*, 2001.

V. Kolmogorov and R. Zabih. What Energy Functions can be Minimized via Graph Cuts? *Pattern Analysis and Machine Intelligence*, 2004.

K. Konolige. Improved Occupancy Grids for Map Building. *Autonomous Robots*, 4(4), 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*, 2006.

K. Konolige, M. Agrawal, and J. Sola. Large Scale Visual Odometry for Rough Terrain. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008.

P. Kovesi. Peter Kovesi's Image Processing Matlab Functions, 2008. URL `http://www.csse.uwa.edu.au/~pk/research/matlabfns`.

J.J. Kuffner and S.M. LaValle. RRT-Connect: An Efficient Approach to Single-query Path Planning. In *IEEE International Conference on Robotics and Automation*, pages 995–1001, San Francisco, California, April 2000.

B. Kuipers. The Spatial Semantic Hierarchy. *Artificial Intelligence*, 119:191–233, 2000.

B. Kuipers. Modeling Spatial Knowledge. *Cognitive Science*, 2:129–153, 1978.

B. Kuipers, J. Modayil, P. Beeson, M. MacMahon, and F. Savelli. Local Metrical and Global Topological Maps in the Hybrid Spatial Semantic Hierarchy. In *IEEE International Conference on Robotics and Automation*, 2004.

R. Labayrade, C. Royere, D. Gruyer, and D. Aubert. Cooperative Fusion for Multi-Obstacles Detection with use of Stereo Vision and Laser Scanner. *Autonomous Robots*, 19(2), 2005.

S. Li. *Markov Random Field Modeling in Computer Vision*. Springer-Verlag, 1995.

D.G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

L. Matthies and S. Shafer. Error Modeling in Stereo Navigation. *IEEE Journal of Robotics and Automation*, 3(3):239–248, 1997.

J. Michels, A. Saxena, and A. Ng. High-Speed Obstacle Avoidance Using Monocular Vision and Reinforcement Learning. In *International Conference on Machine Learning*, 2005.

K. Mikolajczyk and C. Schmid. Scale and Affine Invariant Interest Point Detectors. *International Journal of Computer Vision*, 60(1):63–86, 2004.

J. Modayil and B Kuipers. Bootstrap Learning for Object Discovery. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004.

J. Modayil, P. Beeson, and B. Kuipers. Using the Topological Skeleton for Scalable Global Metrical Map-Building. In *IEEE International Conference on Intelligent Robots and Systems*, 2004.

B. Morisset, R.B. Rusu, A. Sundaresan, K. Hauser, M. Agrawal, J.C. Latombe, and M. Beetz. Leaving Flatland: Toward Real-Time 3D Navigation. In *IEEE International Conference on Robotics and Automation*, 2009.

A. Murarka and B. Kuipers, 2009a. URL `http://eecs.umich.edu/~kuipers/research/pubs/Murarka-iros-09.html`.

A. Murarka and B. Kuipers. A Stereo Vision Based Mapping Algorithm for Detecting Inclines, Drop-offs, and Obstacles for Safe Local Navigation. In *IEEE International Conference on Intelligent Robots and Systems*, 2009b.

A. Murarka, J. Modayil, and B. Kuipers. Building Local Safety Maps for a Wheelchair Robot Using Vision and Lasers. In *3rd Canadian Conference on Computer and Robot Vision*, 2006.

A. Murarka, M. Sridharan, and B. Kuipers. Detecting Obstacles and Drop-offs using Stereo and Motion Cues for Safe Local Motion. In *IEEE International Conference on Intelligent Robots and Systems*, pages 702–708, 2008.

A. Murarka, S. Gulati, P. Beeson, and B. Kuipers. Towards a Safe, Low-Cost, Intelligent Wheelchair. In *3rd Workshop on Planning, Perception and Navigation for Intelligent Vehicles*, 2009.

D. Murray and J. Little. Using Real-time Stereo vision for Mobile Robot Navigation. In *Workshop on Perception for Mobile Agents at CVPR*, Santa Barbara, CA, 1998.

P. Newman, D. Cole, and K. Ho. Outdoor SLAM using Visual Appearance and Laser Ranging. In *IEEE International Conference on Robotics and Automation*, 2006.

D. Nister. An Efficient Solution to the Five-Point Relative Pose Problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(6):756–770, 2004.

C. Plagemann, S. Mischke, S. Prentice, L. Kersting, N. Roy, and W. Burgard. Learning Predictive Terrain Models for Legged Robot Locomotion. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008.

Player/Stage, 2009. URL `http://playerstage.sourceforge.net`.

J. Poppinga, N. Vaskevicius, A. Birk, and K. Pathak. Fast Plane Detection and Polygo-

nalization in Noisy 3D Range Images. In *IEEE International Conference on Intelligent Robots and Systems*, 2008.

E. Prassler, J. Scholz, and P. Fiorini. Navigating a Robotic Wheelchair in a Railway Station during Rush Hour. *International Journal of Robotics Research*, 18(7):711–727, 1999.

A. Rankin, C. Bergh, S. Goldberg, P. Bellutta, A. Huertas, and L. Matthies. Passive Perception System for Day/Night Autonomous Off-Road Navigation. In *Proceedings of the SPIE Defense and Security Symposium: Unmanned Ground Vehicle Technology VI Conference*, pages 343–358, Orlando, FL, 2005a.

A. Rankin, A. Huertas, and L. Matthies. Evaluation of Stereo Vision Obstacle Detection Algorithms for Off-Road Autonomous Navigation. In *32nd AUVSI Symposium on Unmanned Systems*, 2005b.

A.L. Rankin, A. Huertas, and L. Matthies. Nighttime Negative Obstacle Detection for Off-road Autonomous Navigation. In *SPIE Defense and Security Symposium: Unmanned Systems Technology IX Conference*, 2007.

C. Rasmussen. A Hybrid Vision + Ladar Rural Road Follower. In *IEEE International Conference on Robotics and Automation*, 2006.

D.B. Reid. An Algorithm for Tracking Multiple Targets. *IEEE Transactions on Automatic Control*, 24(6), 1979.

R.B. Rusu, A. Sundaresan, B. Morisset, M. Agrawal, and M. Beetz. Leaving Flatland: Realtime 3D Stereo Semantic Reconstruction. In *International Conference on Intelligent Robotics and Applications*, 2008.

K. Sabe, M. Fukuchi, J.-S. Gutmann, T. Ohashi, K. Kawamoto, and T. Yoshigahara. Obstacle Avoidance and Path Planning for Humanoid Robots Using Stereo Vision. In *IEEE International Conference on Robotics and Automation*, 2004.

A. Saxena, S. Chung, and A. Ng. Learning Depth from Single Monocular Images. In *Neural Information Processing Systems*, 2005.

A. Saxena, J. Schulte, and A. Ng. Depth Estimation using Monocular and Stereo Cues. In *International Joint Conference on Artificial Intelligence*, 2007.

D. Scharstein and R. Szeliski. A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. *International Journal of Computer Vision*, 47(1):7–42, 2002.

D. Scharstein and R. Szeliski. Middlebury Stereo Vision Page, 2009. URL `http://vision.middlebury.edu/stereo`.

S. Se and M. Brady. Ground Plane Estimation, Error Analysis and Applications. *Robotics and Autonomous Systems*, 39:59–71, 2002.

S. Se and P. Jasiobedzki. Photo-realistic 3D Model Reconstruction. In *IEEE International Conference on Robotics and Automation*, 2006.

R. Sim and J.J. Little. Autonomous Vision-Based Exploration and Mapping Using Hybrid Maps and Rao-Blackwellised Particle Filters. In *IEEE International Conference on Intelligent Robots and Systems*, 2006.

S. Singh, R. Simmons, T. Smith, A. Stentz, V. Verma, A. Yahja, and K. Schwehr. Recent Progress in Local and Global Traversability for Planetary Rovers. In *IEEE Conference on Robotics and Automation*, 2000.

D. Stavens and S. Thrun. A Self-Supervised Terrain Roughness Estimator for Off-Road Autonomous Driving. In *Uncertainty in Artificial Intelligence*, 2006.

A.N. Stein and M. Hebert. Local Detection of Occlusion Boundaries in Video. In *British Machine Vision Conference*, 2006.

A. Stentz. The Focussed D* Algorithm for Real-Time Replanning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1652–1659, 1995.

D. Stronger and P. Stone. Maximum Likelihood Estimation of Sensor and Action Model Functions on a Mobile Robot. In *IEEE International Conference on Robotics and Automation*, 2008.

J. Sun, H. Y. Shum, and N. N. Zheng. Stereo Matching using Belief Propagation. *Pattern Analysis and Machine Intelligence*, 2003.

H. Tao, H.S. Sawhney, and R. Kumar. A Global Matching Framework for Stereo Computation. In *International Conference on Computer Vision*, 2001.

S. Thrun. Robotic Mapping: A Survey. In G. Lakemeyer and B. Nebel, editors, *Exploring Artificial Intelligence in the New Millennium*. Morgan Kaufmann, 2002.

S. Thrun, C. Martin, Y. Liu, D. Hahnel, R. Emery-Montemerlo, D. Chakrabarti, and W. Burgard. A Real-time Expectation Maximization Algorithm for Acquiring Multi-planar Maps of Indoor Environments with Mobile Robots. *IEEE Transactions on Robotics*, 2004.

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. 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.

L.A. Torres-Mendez and G. Dudek. Statistical Inference and Synthesis in the Image Domain for Mobile Robot Environment Modeling. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004.

O. Trullier, S. Wiener, A. Berthoz, and J. Meyer. Biologically-based Artificial Navigation Systems: Review and Prospects. *Progress in Neurobiology*, 51:483–544, 1997.

I. Ulrich and I. Nourbakhsh. Appearance-based Obstacle Detection with Monocular Color Vision. In *AAAI National Conference on Artificial Intelligence*, 2000.

U.S. Department of Justice. ADA Guide for Small Towns, 2000. URL `http://www.usdoj.gov/crt/ada/smtown.pdf`.

U.S. Department of Justice. ADA Standards For Accessible Design, 1994. URL `http://www.usdoj.gov/crt/ada/stdspdf.htm`.

Videre Design, 2006. URL `http://www.videredesign.com`.

C. Wellington and A. Stentz. Online Adaptive Rough-Terrain Navigation in Vegetation. In *IEEE International Conference on Robotics and Automation*, 2004.

C. Wellington, A. Courville, and Stentz A. Interacting Markov Random Fields for Simultaneous Terrain Modeling and Obstacle Detection. In *Robotics: Science and Systems*, June 2005.

Wikipedia. Dijkstra's Algorithm, 2009a. URL `http://en.wikipedia.org/wiki/Dijkstra's_algorithm`.

Wikipedia. Jensen-Shannon Divergence, 2009b. URL `http://en.wikipedia.org/wiki/Jensen-Shannon_divergence`.

Wikipedia. Kullback-Leibler Divergence, 2009c. URL `http://en.wikipedia.org/wiki/Kullback-Leibler_divergence`.

Vita

Aniket Murarka was born in Kharagpur, West Bengal in 1978. He received a B.Tech. in Mechanical Engineering from the Indian Institute of Technology, Bombay in 1999, and an M.S. in Mechanical Engineering from the University of Texas at Austin in 2001. Since then he has been working on his doctorate in Computer Science at the University of Texas at Austin.

Permanent Address: C-402, Hans Marg, Malviya Nagar
Jaipur, Rajasthan 302017
India

This dissertation was typeset by the author.