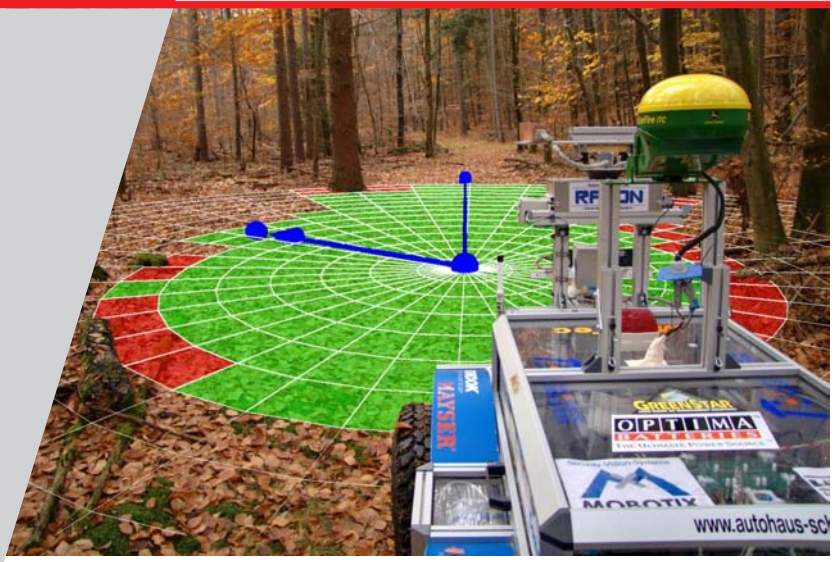
Tim Braun

Technische Universität Kaiserslautern

Cost-Efficient Global Robot Navigation in Rugged Off-Road Terrain



Tim Braun

Cost-Efficient Global Robot Navigation in Rugged Off-Road Terrain

RRLAB
The Robotics Research Lab

Tim Braun

Cost-Efficient Global Robot Navigation in Rugged Off-Road Terrain

This PhD thesis aims at finding a global robot navigation strategy for rugged off-road terrain which is *robust* against inaccurate self-localization, *scalable* to large environments, but also *cost-efficient*, e.g. able to generate navigation paths which optimize a cost measure closely related to terrain traversability.

In order to meet this goal, aspects of both metrical and topological navigation techniques are combined. A primarily topological map is extended with the previously lacking capability of cost-efficient path planning and map extension. Further innovations include a multi-dimensional cost measure for topological edges, a method to learn these costs based on live feedback from the robot and a set of extrapolation methods to predict the traversability costs for untraversed edges. The thesis presents two sophisticated new image analysis techniques to optimize cost prediction based on the shape and appearance of surrounding terrain.

Experimental results indicate that the proposed global navigation system is indeed able to perform cost-efficient, large scale path planning. At the same time, the need to maintain a fine-grained, global world model which would reduce the scalability of the approach is avoided.

Verlag Dr. Hut
ISBN 978-3-86853-135-0
9 783868 531350

Cost-Efficient Global Robot Navigation in Rugged Off-Road Terrain

Dipl.-Inform. Tim Braun

Vom Fachbereich Informatik der

Technischen Universität Kaiserslautern

zur Verleihung des akademischen Grades

Doktor-Ingenieur (Dr.-Ing.)

genehmigte Dissertation.

Zur Begutachtung eingereicht am:	18. November 2008
Datum der wissens. Aussprache:	27. März 2009
Vorsitzender:	Prof. Dr. phil. nat. Markus Nebel
Erster Berichterstatter:	Prof. Dr. rer. nat. Karsten Berns
Zweiter Berichterstatter:	Prof. Dr.-Ing. Klaus-Dieter Kuhnert
Dekan:	Prof. Dr. rer. nat. Karsten Berns
Zeichen der TU im Bibiliotheksverkehr:	D 386

Acknowledgments

Building and maintaining the complex mechanics and control software of a large outdoor robot with the size of a small car is a formidable challenge, which would have been simply impossible to take on without the collaborative effort of many dedicated co-workers at the Robotics Research Lab in Kaiserslautern. At the same time, these people also created a highly enthusiastic and inspiring atmosphere, which was vital to maintain good spirits and creativity, in spite of the countless work hours that we all put in to create something that can actually deal with the humblingly complex reality. Without such great colleagues and friends, the last five years would have been diminuished greatly.

First of all, I'd like to thank my supervisor Prof. Dr. Karsten Berns for enabling my work in the fascinating area of outdoor robotics in the first place. The great deal of faith and optimism he put in every one of us helped a lot to focus again on the 'global vision' behind one's work and the already obtained achievements, especially at times when one was overly deep immersed into technical details and all the things that don't work yet. A special thanks to Prof. Dr. Klaus-Dieter Kuhnert, who was willing to take on the duty of this thesis' second referee on such short notice.

Also, I'd like to thank Carsten Hillenbrand for his great work on the electronic components and low-level controllers that were and are of immeasurable value for everyone in the research group. He is also responsible for the impression of some students that 'someone is always laughing in this working group'. Without the competent system administration of Tobias Luksch and his uncountably many perl scripts, the implementation of a working navigation system would have been much more difficult and time-consuming. Also, I'd have missed out on many great board gaming sessions and beach volleyball plays. In this respect, I'd also like to commemorate the hosting skills of Jan Koch, whose cooking and fine wines made the late night Wii sessions at his place a great thing to remember.

Special thanks are extended to my two colleagues Martin Proetzsch and Helge Schäfer which formed the other two-thirds of the initial 'Team Ravon'. As already stated, the creation of such a complex soft- and hardware system would have been impossible, if one person had tried to do it all alone. Instead, the possibility to build on top of Helge's obstacle avoiding pilot and Martin's behavior-based vehicle control was a central prerequisite which allowed me to approach the task of global navigation in the first place. I'll keep the simultaneous work on a highly integrated software system in memory as an example of successful and effective software engineering. Also, the four competitions which we attended together have been a great experience because of their dedication and the 'code it right until the dawn breaks' attitude, which saved the day more than once. In this context, I'd also wish to thank Christopher Armbrust, Alexander Renner and Tobias Föhst which joined the team shortly before the ELROB 2008 event and now form the next

generation of researchers that continue to work in the area of outdoor robotics. Good luck, guys!

A significant contribution to this thesis was made by the students who opted to do their project or diploma thesis under my supervision. I'd like to especially thank Florian Faust, Gregor Zolynski, Henning Bitsch and Benjamin Seidler for the great deal of effort they put into their respective works. They exceeded the limit of what would have been sufficient to simply achieve a good grade by far. In this context, I'm especially grateful to Henning Bitsch, who continued to improve upon his already excellent work out of personal interest long after he received his university degree.

Further acknowledgments go to the other people in the lab which allowed me to see other interesting areas of their research and carried a part of the group's basic workload. Thanks to my roommate Jens Wettach, who beared with me throughout the last years, and whose dry sense of humor was delightful on numerous occasions. Thanks to Norbert Schmitz for maintaining and fixing many things in our software framework, Daniel Schmidt for putting up with the chores of teaching and undergraduate education, Jochen Hirth for his work on the literature database, and Thomas Wahl for taking up Carsten's work on the electronics as well as the tedious task of laying out new circuit boards. Furthermore, I'd like to thank Rita Broschart for dealing with the administrative chaos that we generated with our many trips to different conferences and various other institutions. Our technician Lothar Gauss needs to be thanked for acquiring even the most obscure materials and tools in record time, even though we consistently found to require these very urgently at the most unsuitable times. I applaud his excellent inter-personal skills, which awarded me the fame of having my favorite bun named officially as 'Tim's Brötchen' in the cafeteria.

Last but not least, I want to express my gratitude to my sister Heike and her spouse Ahmad, who both spent a lot of their free time proof-reading initial versions of this thesis. Their outside perspective provided me with numerous suggestions to improve the written presentation of my work. And finally, I am very thankful for the unwavering support received from my girlfriend Christiane, who tolerated the many late nights at the lab and constantly provided comfort during difficult times when nothing seemed to work out right. It would have been much harder to pull through this without you, Chrissy.

This thesis has been funded by the PhD program of the Department of Computer Sciences of the University of Kaiserslautern. Their support is gratefully acknowledged.

Abstract

This thesis addresses the problem of finding a global robot navigation strategy for rugged off-road terrain which is *robust* against inaccurate self-localization and *scalable* to large environments, but also *cost-efficient*, e.g. able to generate navigation paths which optimize a cost measure closely related to terrain traversability.

A novel navigation methodology is proposed to meet this goal. The approach combines aspects of both metrical and topological techniques in order to exploit the strengths of both classes. It extends a primarily topological map with new aspects that add the previously lacking capability to perform cost-efficient path planning and map extension on the topological level. The introduced additions include a multi-dimensional cost measure for topological edges which records the most relevant aspects of terrain traversability, a technique to learn consistent cost values from scratch based on feedback from the robot during operation and a method to use the gained information for path planning with user-selectable priorities.

To allow the prediction of traversability costs for topological edges which have not been traversed yet, a set of methods to extrapolate edge costs from existing cost information is developed. Furthermore, a new metrical 'local traversability map' is proposed which stores cost modifiers for possible exploration directions in the vicinity of topological nodes in a compact form. These cost modifiers are key to predict the traversal costs of edges that lead into up-to-now unknown terrain. The thesis presents two new, sophisticated image analysis techniques to fill these traversability maps. They use a long range stereo camera system to estimate terrain traversability based on surface shape and visual appearance. A temporary terrain model is constructed and immediately abstracted into light-weight cost modifiers which are stored in a local traversability map. This ensures the maintenance of a minimal world model, leading to a scalable navigation approach which can be used in very large environments.

In order to validate the performance of the proposed methodology, a series of experiments is conducted in both a three-dimensional simulation environment and the real world. The results indicate that the global navigation system proposed in this thesis is indeed able to perform cost-efficient path planning using the introduced hybrid map structure. At the same time, the need to build and maintain a fine-grained, global world model which would reduce the scalability of the approach is avoided. This strongly supports the claim that a scalable yet cost-efficient navigation system can be designed by striking a balance between the two well established types of purely metrical or topological navigation systems.

Contents

1. Introduction

1.1 Motivation

Mobile robots capable of autonomous navigation in unstructured off-road terrain have received a considerable amount of research interest throughout the last decades. Aside from the scientific challenges posed by such systems, interest is spurred by the fact that these robots have a broad range of potential applications. At first, usage scenarios focused on security-related tasks, such as performing border patrols, guarding large corporate estates (airports, fabrication plants), or the scouting of non-urban terrain during a military crisis. More recently, researchers have started to envision the use of robotic systems for different types of outdoor tasks. This includes agricultural services like autonomous harvesting, repetitive transportation duties on construction sites and the ongoing monitoring of environmental conditions within large territories. Mobile off-road robots are nowadays also considered as additional tools which can support human personnel working in disaster areas. This scenario includes the deployment of robots for scouting, cleaning or mapping tasks after natural catastrophes such as earthquakes or forest fires. Likewise, the systems could be used after incidents that caused nuclear or chemical contamination.

In order to navigate *efficiently* through rugged, off-road terrain, the robot must be able to reason about the varying degrees of traversability found in the environment. Consequently, it needs to quantify the traversability of each considered path using a *traversal cost measure*. Only such a metric allows to compare available travel options algorithmically and select the optimal alternative. The definition of an expressive cost measure and its accurate correlation with the properties of the surrounding terrain are thus central issues for the formulation of an efficient navigation strategy in off-road robotics.

Many established approaches which compute non-trivial cost measures deduce traversability costs based on an accurate, global metrical model of the load bearing terrain surface. This allows to detect both slopes of varying degrees and obstacles. However, the construction and maintenance of such a detailed world model is problematic in large off-road terrain for a variety of reasons.

On the one hand, the huge amount of data that has to be handled imposes a high burden on a mobile robot's computational resources. On the other hand, new sensor data must

be incorporated into the existing world model in a metrically consistent fashion, which requires *accurate* robot localisation. Especially in forested areas with much vegetation, this requirement can be difficult to meet. After all, global localization sensors such as GPS work with reduced efficiency in these places, and vegetation is not overly well suited for landmark or point cloud based relocalization techniques due to issues like perceptual aliasing, the unreliable detection of foilage or its viewpoint-dependent penetrability.

To address the scalability and consistency problems that arise with the use of a detailed metrical model, some researchers have proposed to employ a more abstracted, mainly topological representation for large-scale navigation. Such a topological model focuses on representing navigation-relevant places and the possible driving options between them. Usually, it does *not* contain metrical information about obstacles anymore. Instead, topological navigation approaches rely upon a subordinated piloting system which is capable of detecting and circumventing obstacles at close ranges by itself, using current sensor readings and possibly, a spatially limited, local world representation.

In general, these approaches scale much better than those which perform extensive metrical modeling of the environment. However, the topological methods currently in use for global off-road navigation also abstract from most of the information needed to define a cost measure which accurately reflects the characteristics of the corresponding path. Thus, the resulting navigation system loses a great deal of efficiency: first by using the abstracted topological model instead of a metrical map, and second by replacing realistic traversability cost measures with a substitute metric (such as the distances between topological map nodes) which does not accurately reflect the properties of the underlying terrain.

It seems that the two predominant ways to build a world representation for robot navigation have some drawbacks when it comes to enabling large-scale, cost-efficient navigation in off-road terrain. On the one side, techniques which use a detailed, metric world model can perform cost-efficient planning, but fail to scale because they don't abstract enough from details. On the other side, topological approaches scale well, but suffer from too much abstraction – since they also abstract from terrain traversability, they cannot support cost-efficient path planning. Thus, the question is: Is there a way to combine the strengths of both approaches and derive a scalable, yet cost-efficient system for large-scale navigation in rugged outdoor terrain? How can a robot represent, learn, predict and sense appropriate cost measures for efficient path planning, without entailing the considerable drawbacks of the expensive, 'full detail' world model?

The goal of this thesis is to help answering these questions.

1.2 Goal of this Thesis

The goal of this thesis is to define and evaluate a *robust* and *scalable* methodology for *cost-efficient*, *global* robot navigation in rugged and vegetated outdoor terrain.

In this context, 'global navigation' is understood as the task of generating navigation decisions that have a spatial extent significantly larger than the immediate sensor horizon of the robot. Thus, the researched navigation system, called the **navigator**, shall concentrate on robot navigation tasks starting above a range of about 10 meters, going up to several kilometers. An expressive *cost measure* needs to be formulated in order to select a path with optimal cost from the set of available options. However, the cost estimation shall not require the construction of a highly detailed, global world model. Instead, a *minimal world model* has to be found which still allows reasoning about traversability costs. This includes the selection of the optimal path at a global scale as well as the cost-efficient exploration of untraversed terrain. At the same time, the model shall abstract from local aspects of the environment or the robot's exact trajectory. These issues are to be handled by a piloting subsystem that maintains a metrically accurate, but spatially limited world representation.

The existence of a local piloting system, called the **pilot**, is presumed in this thesis. The pilot is responsible for 'local navigation'. This entails steering the robot around obstacles and towards a goal using a local world model with a size comparable to the robot's sensor range, e.g. a size of about 10 meters. Therefore, the global navigation strategy that is to be found does *not* need to cope directly with the task of local obstacle avoidance or physical robot control.

The main hypothesis underlying the thesis at hand is the assumption that cost-efficient global navigation can be accomplished in a scalable way by building a minimal world model and refining it continuously based on experience and self-observation of the pilot. With this approach, the additional knowledge invested into the global navigation layer can be kept at a minimum and a high degree of robustness against sensor noise or inaccurate self-localization can be achieved. Instead of building separate cost estimation rules from scratch, the global navigator can benefit from the world representation already constructed in the pilot, reuse the domain knowledge invested there, and abstract it to optimize path planning on a large spatial scale.

The design of a global navigation methodology that verifies this hypothesis and meets the formulated goal raises a whole set of questions. The following aspects are deemed especially relevant in this context and will be investigated thoroughly in the thesis:

Appropriate World and Travel Cost Representation

As stated in the motivation, a combination of metrical and topological map paradigms might yield a suitable world model for cost-efficient global navigation in rugged off-road terrain. This intuitive notion needs to be subjected to scientific analysis and matured into the formulation of both a concrete *world representation* and an expressive *cost metric* that are indeed well suited for global navigation.

Interaction of Global and Local Navigation Layers

Once a suitable type of global world model / map is formally defined, the *interaction* between this model (managed by the navigator) and the pilot responsible for local navigation has to be examined. How can an appropriate sequence of driving commands be

generated for the pilot on the basis of the global map? Which techniques allow to detect navigation successes and failures robustly? And how can the global world model be *adapted* in response to these events in order to improve the overall map quality?

Consistent Travel Cost Assessment

The separation of global and local navigation leads to the effect that the physical trajectory implemented by the robot is not pre-planned accurately by the global navigation system, but emerges in situ through interaction of the piloting subsystem and the terrain. This raises the problem of *cost consistency*, e.g. the question how to ensure that the navigator works with path costs that reflect the costs which are actually incurred by the pilot. Can consistent path costs for global navigation be determined at all if the driven path is not known beforehand? And if not, can the *transfer* of cost experience from completed travels to the navigator at least help to approximate the true costs in the long run?

Prediction of Travel Costs and Map Extension

In order to evaluate new travel options for global navigation, it is not sufficient to consider only existing cost information. It is also required to *predict* the traversal costs of new map connections as accurately as possible. This requires research into methods that allow to transfer cost data between established and new paths within the global map. Based on the capability to extrapolate costs, strategies to *extend* the map with cost-efficient new paths shall be investigated. How can this be used to reach a previously unreachable goal in unknown terrain?

Improved Large-Scale Travel Cost Prediction using Sensor Information

To improve the quality of cost prediction on the global navigation level, algorithms shall be devised which evaluate *sensor information* over longer ranges than considered by the pilot. However, the methods shall be focused on extracting only information needed for the tasks performed by the navigator, e.g. for cost-efficient path planning. Likewise, a representation shall be found that exclusively stores the information relevant to these tasks for future use in order to keep the world model minimal.

1.3 Document Structure

The remainder of this document is structured as follows:

Chapter 2 presents several basic components that constitute the foundation for the techniques derived in the subsequent parts of this thesis. It presents the physical robotic system which is used to validate the developed methods in the real world and describes an existing piloting system that fills the role of the robot's 'pilot' for local navigation. This component is regarded as a given and will be used by the global navigation strategies which are the actual focus of this thesis. The chapter also introduces coordinate system conventions that will be required in the following chapters to specify locations in the world representation.

Chapter 3 deals with the definition and utilization of an appropriate world representation for the envisioned task of global navigation in rugged terrain. It contains a survey of

existing map types which represent navigation related knowledge and formulates both a suitable world model and cost measure for global navigation. It also introduces algorithms that facilitate the interaction between the navigator and the pilot. This includes the generation of driving commands for the pilot using the navigator's global map and the adaptation of the map structure using success or failure feedback from the piloting layer. The chapter also presents a learning mechanism that derives consistent global path costs through observation of the local piloting layer.

The extrapolation of costs stored in the global map onto new connections and the use of this capability for map extension is treated in chapter 4. A set of methods is introduced which allows to transfer existing cost measures between connections in the global map. The influence of using cost information with different degrees of locality on the accuracy of the resulting cost prediction is evaluated and a precedence relation for the different methods is defined. In the second part of the chapter, a method to exploit the cost prediction capability for goal-directed and cost-efficient exploration of previously untraversed terrain is proposed. Its performance is empirically tested in a real-world scenario.

Chapters 5 and 6 present two novel approaches to estimate terrain traversability costs based on information from long range visual sensors. Algorithms are proposed to use the derived information in order to improve the quality of cost prediction on the global navigation level. Chapter 5 describes a technique which uses *geometrical* aspects of the terrain to build a piecewise model of the surface slope and reason upon its traversability. *Appearance* based criteria are applied to the sensed visual information in chapter 6 in order to allow the detection of vegetation which appears solid but may actually be passable. The performance of both sensor analysis methods is evaluated using a series of real world image recordings.

Chapter 7 summarizes the obtained results and the contributions to the field of outdoor robotics. The thesis concludes with a discussion of the achieved results and an outlook on future work.

2. Fundamentals

As outlined in the introduction, the global navigation methodology examined in this thesis presumes the existence of a local navigation layer capable of controlling the employed off-road robot on a local scale. This foundation has a substantial influence on the work done in the scope of this thesis. Therefore, the off-road robot RAVON which is used to test the proposed global navigation algorithms is briefly introduced in the following. Afterwards, the existing piloting system which has been selected to fill the role as the local 'pilot' for the global navigator is presented. The pilot is based on a behavior-based framework and implements basic maneuvering capabilities such as approaching a local target position and obstacle avoidance. After addressing these two topics, two coordinate systems are formalized to allow the future specification of world positions in defined frames of reference.

2.1 The Experimental Platform RAVON

In order to experimentally validate the developed global navigation methodology, a robot capable of traversing rugged off-road terrain is required. Such a robot needs a powerful actuation system to provide the required maneuverability across uneven ground, and therefore needs to be larger and heavier than many other robots (e.g. the Pioneer series) used in other areas of mobile robotics. While this causes many additional technical problems, it also allows the integration of several fast computers with high processing power. Thus, the developed algorithms can be more demanding than what is typically seen as the limit of real time robotics.

Figure 2.1 shows the experimental wheel-based vehicle RAVON (*Robust Autonomous Vehicle for Off-road Navigation*), which has been developed at the Robot Research Laboratory of the University of Kaiserslautern in order to serve as a testing platform for off-road navigation. RAVON has a length of 2.35 m, a width of 1.4 m, and weighs about 650 kg. It is propelled by four electrical DC motors and can climb slopes up to 45° with a maximum velocity of 2 m/s. The motor currents of each motor are measured using hall sensors. The currents are monitored continuously by two motor controllers employing digital signal processors in order to prevent motor overloads. Additionally, the current

(a) Front view

(b) Side view

(c) Typical operation terrain: Forest

(d) Typical operation terrain: Rugged off-road

Figure 2.1: The experimental platform RAVON

measurements can be accessed by higher control level software and used e.g. to estimate the robot's current energy consumption. Front and rear axis of RAVON can be steered independently, allowing advanced driving maneuvers such as tight turning radii or parallel steering. The robot is powered by a set of 8 spiral cell lead batteries which results in an operation time of about 4 hours in grassy terrain. Four industrial PCs running Linux form the main processing core of RAVON. They are equipped with dual core CPUs to provide a more advantageous power/performance ratio and connect with the sensors and actuators via controller area network (CAN) buses. Two computer systems are additionally equipped with graphical processing units (GPUs), as parts of the developed algorithms have been ported to this architecture for speed reasons. The robot estimates its world position by fusing the output of two separate Global Positioning Systems (GPS), an inertial measurement unit (IMU), an electronic compass and wheel encoders using a specialized kalman filtering technique [Schmitz 06]. The global localization accuracy ranges from below 20 cm in open space up to about 8 m in dense foilage. However, due to the use of highly sensitive receivers and the redundancy of the employed systems, localization is virtually never lost completely in natural forests.

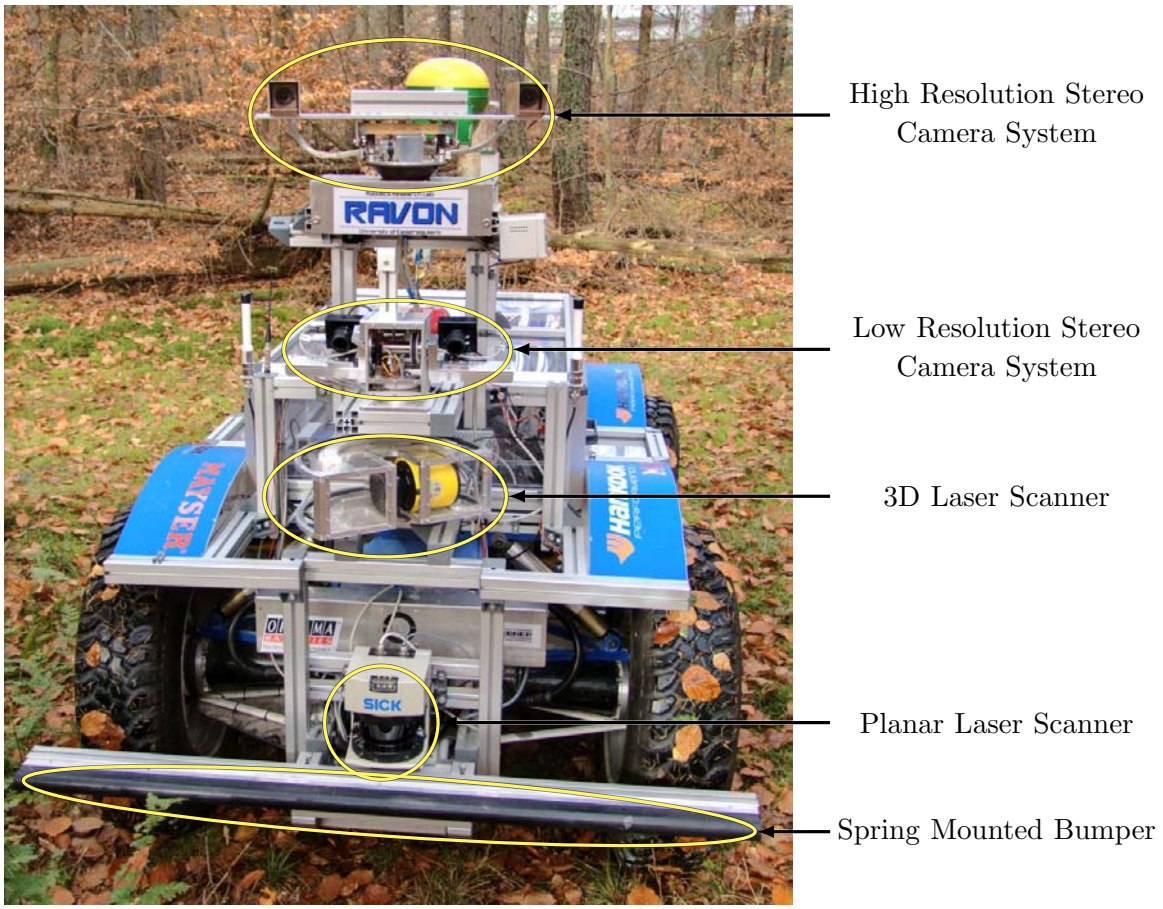


Figure 2.2: RAVON's sensor systems

RAVON features a variety of sensor systems for obstacle detection (see figure 2.2). At closest range, a spring-mounted bumper system is used to detect hindrances on a tactile basis and provide both protection and a safety shutdown in case of collisions. Additionally, it can be used to try pushing soft obstacles at very low speed and allows to detect flexible vegetation upon contact using a depression sensor at the spring mount. Two horizontally aligned 2D SICK laser range finders monitor close range safety zones (5 m front and rear) with a 180° field of vision and an angular resolution of 0.5°. Obstacle detection up to a distance of about 10 m relies on a wide angle stereo camera system with a resolution of 640x480 pixels and a horizontal field of view of 60° and a 3D laser scanner system (implemented by continually panning an upright mounted 2D scanner along the vertical axis). The obtained distance measurements are used to classify the terrain ahead into ground, vegetation/solid obstacles and overhangs. The gained information is then integrated into a short term obstacle memory, extending 8 meters around the robot [Schäfer 07] [Schäfer 08]. This extends the obstacle avoidance capabilities to objects that pass into the blind spots at the sides of RAVON. The obstacle avoidance sensors and their detection areas are depicted in figure 2.3.

In the scope of this thesis, the sensor suite has been extended with a turnable high resolution stereo camera system mounted on top of RAVON's chassis. This camera system is used for shape- and appearance-based terrain traversability estimation in a radius of up to 30 m. Further technical details for this system are given in section 5.2.1.

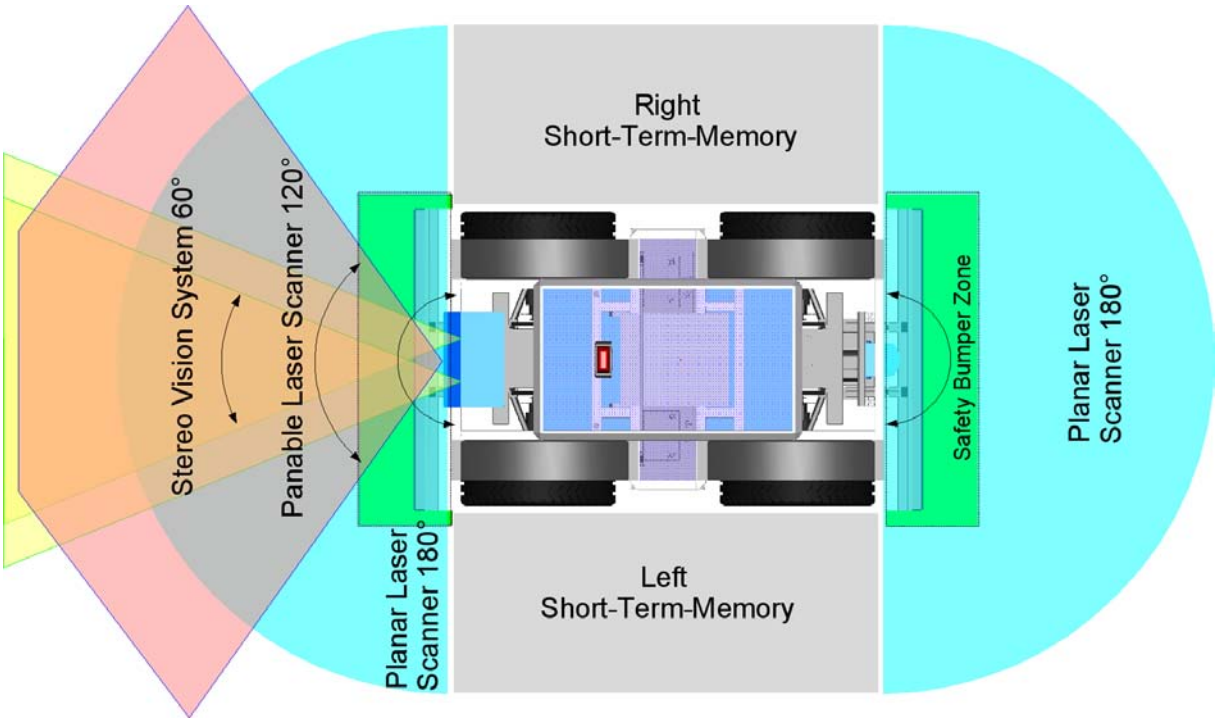


Figure 2.3: RAVON's visual sensor ranges ([Schäfer 07])

2.2 The Behavior-Based Piloting System

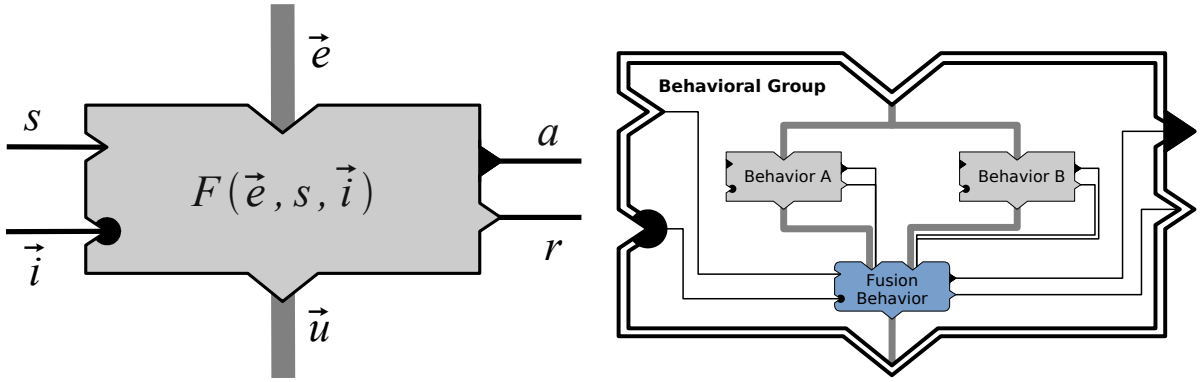
The introduction has listed some reasons why control systems that plan actions based on complex physical models of the robot and the environment are not ideally suited for rugged off-road terrain. Therefore, a different approach has been researched in previous work in order to build a piloting system able to guide RAVON safely around obstacles while approaching a local goal position.

The pilot that results from this effort is modeled as a **behavior-based system** according to the classification scheme of [Matarić 02] and consists of a set of interconnected behaviors instead of a single, monolithic control system [Proetzsch 07a]. Each single behaviour in the pilot network has a clearly defined and typically very simple 'goal' which it wants to accomplish, and considers *only* the aspects of the world that are relevant for this task. These aspects are often derived from current sensor data, although the use of more persistent information is also permitted. Complex motions are not pre-planned, but *emerge* through interaction of multiple behaviors. In this way, the control depends less on an accurate, global world model and is therefore more robust against false sensor interpretations.

2.2.1 The iB2C Architecture

The formal basis of the piloting system is constituted by the iB2C behavior-based system architecture presented in [Proetzsch 07b] and [Proetzsch 08]. As some insight into the internal workings of the pilot is required for the later exposition, the framework is now introduced shortly.

The fundamental building blocks in iB2C are behavior modules, such as depicted in figure 2.4a. Each behavior module receives a vector of input data $\vec{e}$ and produces an output



(a) A behavior module (b) A behavior group

Figure 2.4: The building blocks of iB2C

vector $\vec{u}$. The semantic function of the behavior is modeled by the transfer function F, which describes how the output of the module depends on the current input values. F can express a purely reactive response to the input values, but it is also possible to formulate a more complex function which uses the behavior's internal state (such as a state machine or a pattern recognition algorithm). This way, both reflex-like sensor-actor couplings and highly deliberative behaviors can be implemented (as postulated for behavior-based architectures by [Matarić 97]).

Aside from the input vector $\vec{e}$ and the behavior's internal state, the transfer function is modulated by two control inputs which coordinate the interaction between different behavior modules. Each behavior has a **stimulation** input $s \in [0, 1]$ which represents the external stimulation requested by other (hierarchically superior) behaviors. The inverse effect is induced by the **inhibition** input vector $\vec{i}$, which contains a set of inhibition inputs $\in [0, 1]$ from which the maximum inhibition value is taken into account at every given time. As output, each behavior generates two control signals that allow to deduce information about its current state or influence other behaviors. The **activity** $a \in [0, 1]$ represents the amount of influence that the behavior wants to achieve in the behavior network and is typically connected to the stimulation input of another (fusion) behavior. The **target rating** r is a measure which specifies how *unsatisfied the behaviour is with the current situation* given the behaviour's 'goal'. It ranges from 0 for totally satisfied behaviors to 1 in the totally unsatisfied case.

Competing behaviors are coordinated by fusing two or more of them using standardized fusion behaviors (figure 2.4b) which provide the very same interface as basic behavior modules. The input vector $\mathbf{e}$ of fusion behaviors is composed of the activities a_i, the target ratings r_i, and the output vectors $\mathbf{u_i}$ of the particular competing behaviors B_i. The transfer function F is the fusion function (weighted or maximum) processing the input vector value-wise to a merged output vector u. The behavior set $B = \{B_0, \ldots, B_n\}$ and a corresponding fusion behavior can be put into a single behavior group which implements the same interfaces as a single behavior. In this way, a hierarchical network can be created by combining atomic behaviors into more and more complex behaviors.

2.2.2 Layout of the Pilot

As an example for the use of the iB2C components in the actual implementation of RAVON's piloting layer, the Forward Obstacle Stop behavior group is shown in figure 2.5.

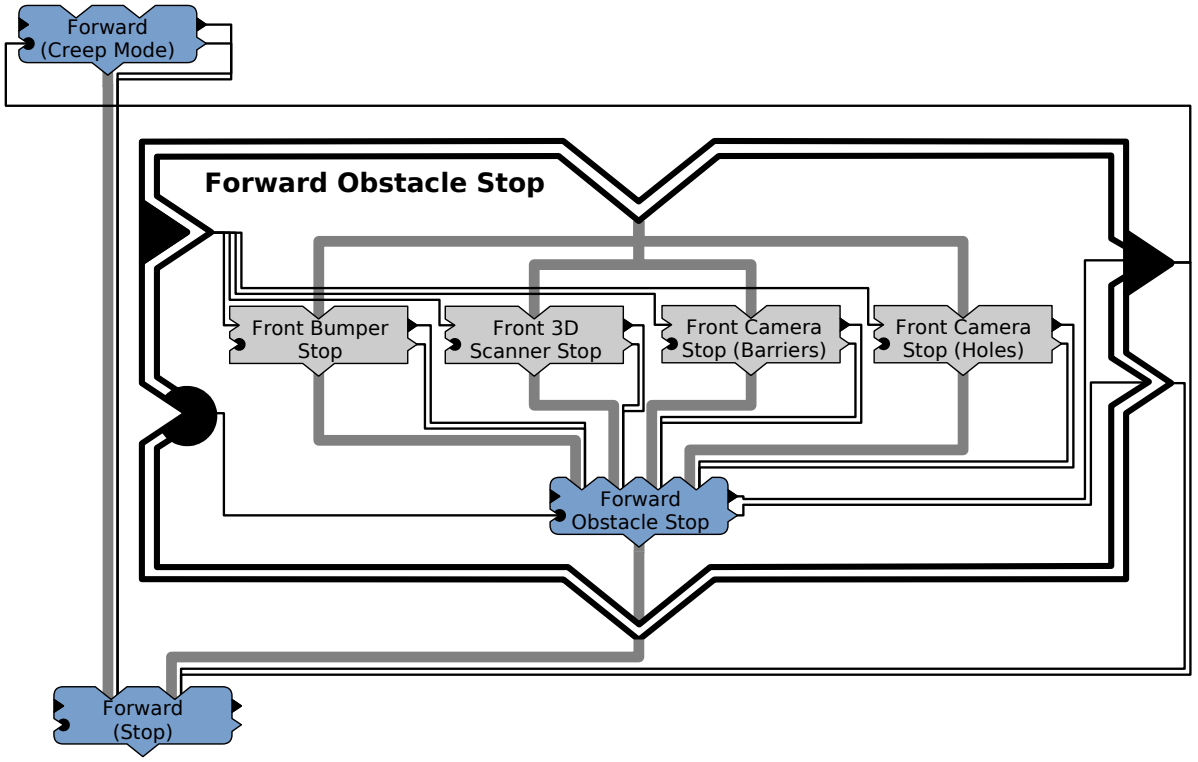


Figure 2.5: An exemplary behavior group of the piloting layer

Each of the contained behaviors wants to stop the robot's forward motion in case an untraversable obstacle is detected in close proximity. However, each behavior considers a different aspect of the available sensor data. The Front Bumper Stop behavior, for example, becomes active and unsatisfied (both the activity a and the target rating r rise) in case the sensors detect pressure on the frontal bumper. Via the fusion behavior, this activity is ultimately fed back as an inhibition into the Forward (Creep Mode) behavior, reducing its activity and hence, the forward speed of the robot. In combination with the other basic behaviors, the Forward Obstacle Stop behavior group realizes a more complex behavior which reacts to multiple sensor modalities that require a vehicle stop.

Moving towards a more high-level view and considering the arrangement of the pilot's main behavior groups only, the complete behaviour system is laid out in three main layers, as depicted in the simplified overview in figure 2.6.

The topmost interface layer provides the exterior interface to the services of the robot pilot. The pilot is able to either receive manual steering commands from a human operator using a joystick, or it can approach a given target position or pose specified in its working coordinate system WCS (see the next section). The latter two actions are initiated if the Approach Target Position or Approach Target Pose behaviors are activated (receive an activation input > 0) by the user or, as depicted in the figure, by the global navigator. In

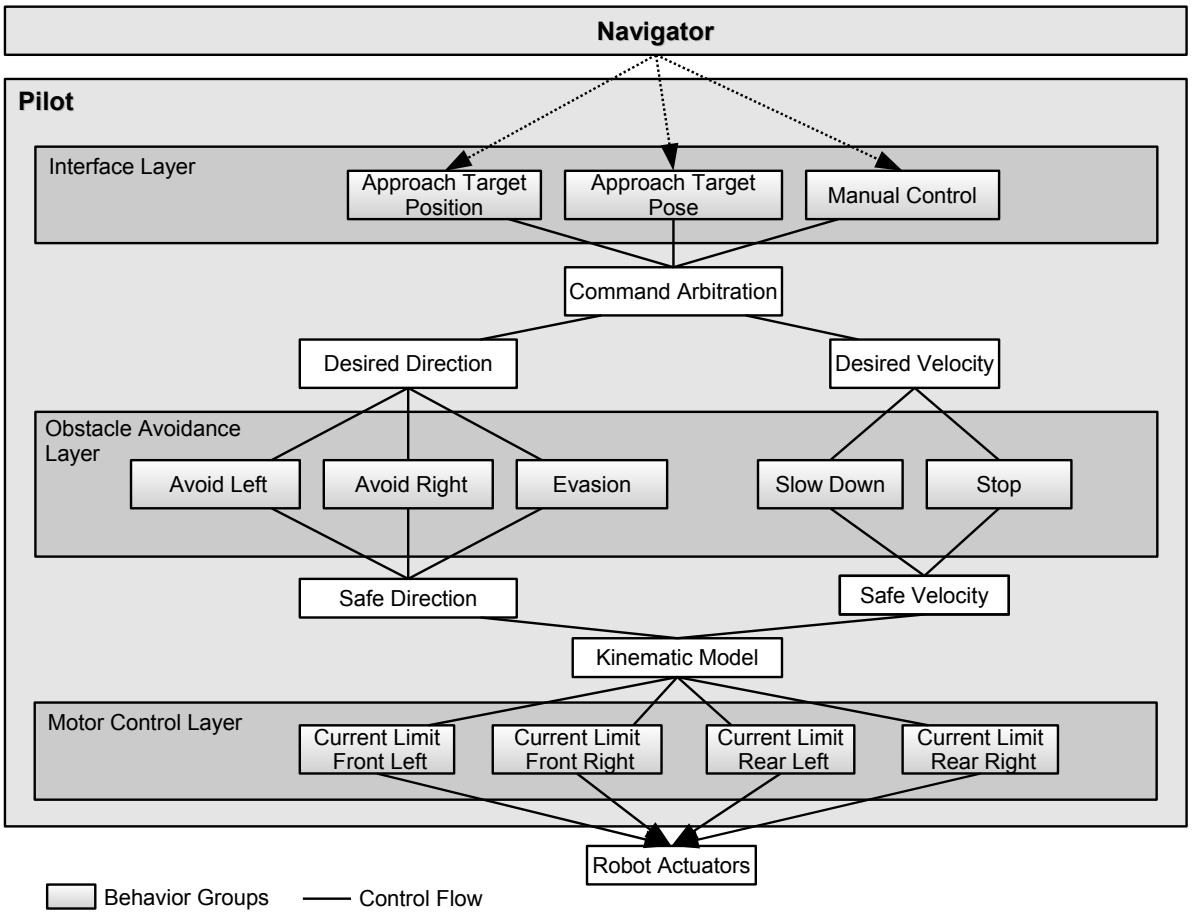


Figure 2.6: The three main behavior layers of the pilot

both cases, the behaviors generate a driving command that depends on the current robot pose and the relative orientation to the given goal position. In case of the Approach Target Pose behavior, the behavior tries to reach the given goal position *and* to achieve a given orientation at this point. This is internally accomplished by a whole group of behaviors which is responsible for lunging out before the goal position is reached in order to turn the robot appropriately. However, due to the hierachical composition of the iB2C behaviors, this complex movement sequence appears to be implemented by just one visible behavior group.

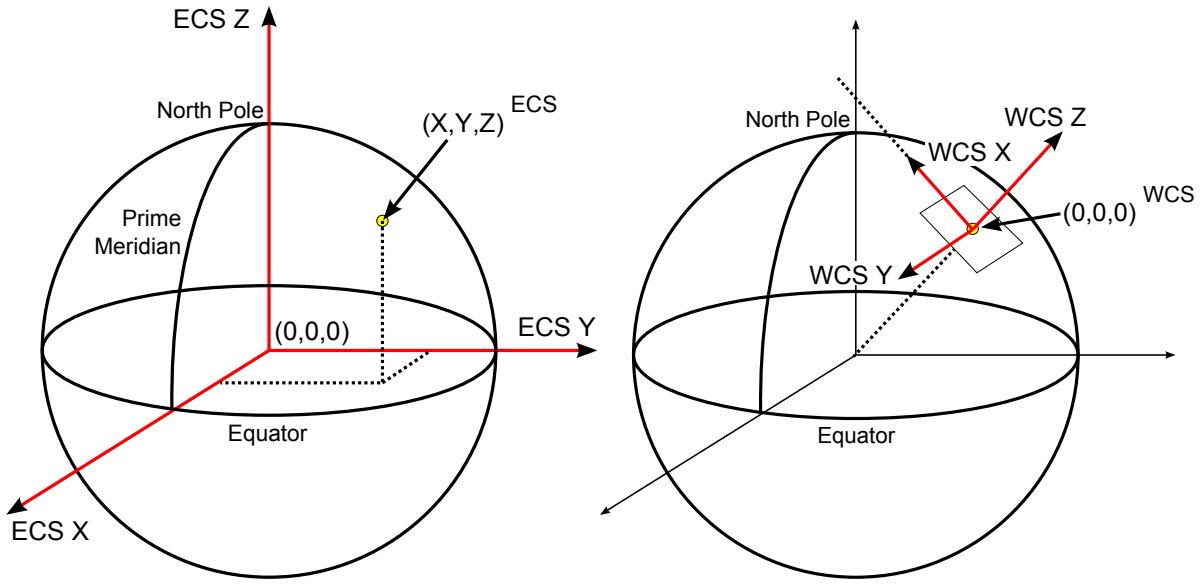
The driving direction and speed set by the interface layer is subsequently modified according to the judgement of the behaviors in the obstacle avoidance layer. This layer contains two behavior groups (named Avoid Left/Right) that turn the robot left or right if obstacles are detected on either side by any of the available sensor systems. If obstacles are detected directly ahead, the Evasion group is responsible for deciding on which side the obstacle shall be passed. This decision remains valid until the obstacle situation has been negotiated completely, avoiding oscillations of the Avoid Left/Right behaviors due to sensor noise. The behavior groups Slowdown and Stop decelerate or stop the vehicle close to obstacles. Further behaviors, which are omitted here for clarity, are capable of effecting ranking maneuvers and retracing to previous positions in case a deadlock between two or more behavior groups occurs and would stop the vehicle for indefinite amounts of time.

As a result of the influence of the obstacle avoidance layer, the requested driving direction and speed is potentially altered in order to avoid obstacles and ensure the safety of the vehicle. Based on this 'safe' driving command, the actual motor currents are computed by a kinematic model and fed into the motor control behavioral layer. This layer includes (among others) four behaviors that limit the motor currents to avoid overload situations.

It is important to stress that the actual trajectory driven by the robot after the pilot's Approach Target Position / Pose behaviors are activated *emerges* as a result of complex interactions between the subordinated behaviors and is thus not pre-planned or completely predictable in advance. This poses a significant problem for any high level control that tries to perform the tasks of path planning and traversal cost estimation. One of the main contributions of this thesis is the development of a navigation system which is capable of consistent cost estimation, in spite of the unsteady foundation of the behavior-based pilot.

2.3 Coordinate Systems for Outdoor Environments

The creation and use of a persistent map for path planning requires a coordinate system which remains valid even if the robot is turned off temporarily or reinitialized. The system therefore needs an origin and coordinate axes which are independent of any robot related information (such as heading or position during startup). In the scope of this thesis, a cartesian ECEF (earth-centered-earth-fixed) coordinate system has been selected as a suitable convention which exhibits these properties. This coordinate system is subsequently called the **Earth Coordinate System ECS** to conform to the naming scheme of the other coordinate systems introduced later. The ECS is a three-dimensional, cartesian coordinate system which originates at earth's center of mass as shown in figure 2.7a.



(a) Earth Coordinate System ECS (b) Working Coordinate System WCS

Figure 2.7: Coordinate systems used by navigator and pilot

The ECS Z axis is chosen to be parallel to the earth's rotational axis and thus pokes through the geographic north pole. The X axis is defined to intersect the equator exactly at

the prime meridian (the intersection point has latitude 0° and longitude 0°in the commonly used latitude / longitude coordinate system). The ECS Y axis is completely determined by the two already defined axes and the requirement that ECS is a cartesian coordinate system with three perpendicular axes and a positive (right-handed) orientation. Under these conditions, ECS Y must be equal to ECS Z × ECS X, where × denotes the cross product in $\Re^3$. Here, the property is exploited that the cross product of two vectors yields a third vector which is perpendicular to both. The presented definition of the ECS ensures that the ECS is always in sync with the earth's rotation. Therefore, the coordinates of any point that is fixed with respect to the earth's surface always remain constant. In comparison to the geodetic WGS 84 system commonly used for GPS receivers, ECS coordinates have the benefit that they can be converted to other cartesian coordinate systems without using non-linear, trigonometric functions. ECEF also does not require the introduction of multiple mapping zones as done in the Universal Transverse Mercator (UTM) system.

Although the ECS is stable across robot initializations and can be used to designate any place on earth with fixed coordinates, the ECS coordinates of such a place are typically not very convenient to use. For example, the ECS coordinates of the University of Kaiserslautern are (4119033.7 m, 560825.9 m, 4821521.8 m). So, in order to provide a coordinate frame which is easier to handle and which allows to specify goal positions by hand, the **Working Coordinate System WCS** is introduced in addition to the ECS. The WCS is more convenient to use, has a more accessible value range and more intuitive coordinate axes. However, it depends upon the initial position of the robot and is therefore not suited to mark a position with a unique set of coordinates in a persistent fashion.

The WCS origin is set to the point where the robot control system is first initialized after system startup. The WCS Z axis is chosen to point 'upwards', e.g. to have exactly the opposite direction as the earth's gravitational force vector. Thus, the WCS Z vector can be expressed easily in the ECS frame as the normalized vector running from the center of the earth to the WCS origin (see figure 2.7b). The WCS X axis is constructed to point 'north', e.g. point towards the north pole. However, as the WCS Z axis is already fixed, the WCS X axis can only *approximately* point to the north pole while still remaining orthogonal to WCS Z. Thus, a line running along the WCS X axis is required to intersect a line running along the ECS Z axis, but the intersection point does not lie at the north pole.

2.3.1 General Coordinate System Conversion

In the following chapters, it will frequently be required to transform coordinates between different coordinate systems. In order to describe such conversions mathematically, the use of **homogeneous** coordinates is very convenient. As this is a standard mathematical technique used frequently in many areas of robotics and computer graphics, only a short recapitulation is presented. Further information can be found in [Craig 89].

Informally speaking, a point or vector $\vec{p}$ given in three-dimensional coordinates $(x\ y\ z)^\mathsf{T}$ can be projected into homogeneous space by 'adding' another dimension and setting the fourth coordinate to 1, leading to the coordinates $(x\ y\ z\ 1)^\mathsf{T}$. The main benefit of using this representation in robotics is that it allows to specify both rotation and translation

of coordinates in one single matrix. Since the conversion from one right-handed cartesian coordinate system into another can be expressed as a rigid motion, a single transformation matrix therefore suffices to completely describe the conversion between any two such coordinate systems.

Before adressing how conversion matrices can be determined, some notational terms are fixated to allow precise discussion of coordinates in different reference systems.

Definition 2.1 *The following notational conventions are defined:*

- *The notation $\vec{p}^{\,A}$ is used to express that the coordinates of the vector or point $\vec{p}$ are referenced on the coordinate system A.*
- *A matrix M that converts coordinates given in a coordinate system A into coordinates referenced on another coordinate system B is denoted as M_A^B.*

There are two frequently used ways to formulate a conversion matrix M_A^B. First, it is possible to transform a set of coordinates given in the source system A step by step into the system B using a sequence of basic translation and rotation matrices. M_A^B can then be computed by multiplying all of the elementary transformation matrices together in the right order. This is an intuitive way to specify the transformation and is frequently used to define the local coordinate system of a robot in relation to its environment. Often, three rotation angles called *roll*, *pitch* and *yaw* are used to specify the robot's **orientation** with respect to the three coordinate axes of B. A subsequent translation step with the three coordinates x, y, z then specifies the robot's **position**. It is common to refer to the combination of the orientation transform and the position transform as the (6D) **pose** transformation of the robot with respect to the original reference system, or simply the pose of the robot. The mathematical technique to convert between a 6D robot pose $(x, y, z, roll, pitch, yaw)$ specified in the $WORLD$ coordinate system and the corresponding transformation matrix M_{ROBOT}^{WORLD} is documented in [Craig 89].

The second way to define the conversion from a coordinate system A into another system B is to use the known coordinates of A's three *canonical basis vectors* in the destination coordinate system and an additional translation vector that relates the origins of A and B. Let $\vec{x}^{\,B} = (x_0\ x_1\ x_2)^\mathsf{T}$ be the coordinates of A's X axis base vector ($\vec{x}^{\,A} = (1\ 0\ 0)^\mathsf{T}$ by definition) and $\vec{y}^{\,B}$ and $\vec{z}^{\,B}$ be defined likewise for A's Y and Z base vectors. Also, let $O^{\,B} = (O_0,\ O_1,\ O_2)^\mathsf{T}$ be the position of A's origin in the coordinate system B. Then, the transformation matrix M_A^B which transforms coordinates from the source coordinate system A into B is simply defined as:

$$M_A^B = \begin{pmatrix} x_0 & y_0 & z_0 & O_0 \\ x_1 & y_1 & z_1 & O_1 \\ x_2 & y_2 & z_2 & O_2 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.1}$$

According to the notational conventions stated in definition 2.1, this leads to

$$\vec{p}^{\,B} = M_A^B \cdot \vec{p}^{\,A} \tag{2.2}$$

It can also be shown that the inverse of M_A^B transforms coordinates back from the destination system B into A. Therefore,

$$\vec{p}^{\,A} = \left(M_A^B\right)^{-1} \cdot \vec{p}^{\,B} \tag{2.3}$$

which automatically forces

$$M_B^A = \left(M_A^B\right)^{-1} \tag{2.4}$$

according to definition 2.1.

2.3.2 Conversion between WCS and ECS

Equation 2.1 offers a convenient way to determine the coordinate conversion matrix that leads from the previously introduced working coordinate system WCS to the global ECS. The calculation requires that the ECS coordinates of the robot's initialization point O_W are known. This can be easily accomplished by using the GPS sensor information available to the robot.

Given O_W, the ECS coordinates of the WCS Z axis base vector $\vec{z}^{\,ECS}$ can be calculated by normalizing the vector running from the ECS Origin O_E (with ECS coordinates (0,0,0) by definition) to O_W:

$$\vec{z}^{\,ECS} = \frac{\overline{O_E O_W}}{\left|\overline{O_E O_W}\right|} \tag{2.5}$$

To determine the WCS X axis vector, one can exploit the fact that the WCS X axis must lie in the plane spanned by the WCS Z axis and the ECS Z axis in order to intersect the elongated ECS Z axis as postulated in the WCS definition.

Using the constraint that the WCS Y axis must be orthogonal to the plane in which the WCS X axis resides (and thus, both the WCS Z and ECS Z vectors), the ECS coordinates of the WCS Y axis base vector are given by

$$\vec{y}^{\,ECS} = \frac{\vec{z}^{\,ECS} \times \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}}{\left| \vec{z}^{\,ECS} \times \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \right|} \tag{2.6}$$

$\vec{x}^{\,ECS}$ is now easily determined using

$$\vec{x}^{\,ECS} = \frac{\vec{y}^{\,ECS} \times \vec{z}^{\,ECS}}{\left|\vec{y}^{\,ECS} \times \vec{z}^{\,ECS}\right|} \tag{2.7}$$

With the ECS coordinates of all three WCS base vectors and the origin position O_W available, M_{WCS}^{ECS} can be computed according to equation 2.1.

Algorithm 1: SetupCSWithZAxis(Point3D *origin*, Vector3D *x_direction*, Vector3D *z_axis*)

Data: Point3D *origin*, Vector3D *x_direction*, Vector3D *z_axis*
Result: Coordinate Conversion Matrix M

Vector3D $z \leftarrow \text{normalize}(z\_axis)$
Vector3D $y \leftarrow \text{normalize}(z \times x\_direction)$
Vector3D $x \leftarrow \text{normalize}(y \times z)$

$$\text{Matrix M} = \begin{pmatrix} x_0 & y_0 & z_0 & origin_0 \\ x_1 & y_1 & z_1 & origin_1 \\ x_2 & y_2 & z_2 & origin_2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

return M

2.3.3 Conversion Functions

The approach that has been used to determine the M_{WCS}^{ECS} matrix in the last paragraph can be also employed to calculate the transformation matrices between other coordinate systems. Because the capability to set up local coordinate systems will be used extensively later, the computation of the transformation matrix is packaged into algorithm 1.

As can be observed, the function must be supplied with the origin of the source coordinate system SCS (parameter **origin**), the coordinates of the SCS Z axis vector (parameter **z_axis**) and the direction into which the SCS X axis is supposed to point (parameter **x_direction**). As before, it cannot be guaranteed that the X axis runs exactly along the vector given as parameter **x_direction**. All coordinates of the three input parameters need to be specified in the destination coordinate system DCS. Upon invocation of the SetupCSWithZAxis function, a transformation matrix M_{SCS}^{DCS} is produced. In the actual implementation, M_{WCS}^{ECS} is generated by executing $M_{WCS}^{ECS} = \text{SetupCSWithZAxis}(O_W, (0\ 0\ 1)^\mathsf{T}, O_W)$.

Analogous to algorithm 1, a second algorithm is formulated in algorithm 2, which again constructs a conversion matrix M given three parameters. This time however, the **x_axis** parameter determines the X axis base vector *exactly* and the **z_direction** parameter indicates the desired direction of the Z axis. Although this function has no immediate use for converting between WCS or ECS, it will be required later to setup other local coordinate systems.

2.4 Conclusion

In this chapter, several prerequisites for the following exposition of the researched global navigation methodology have been presented. The mobile off-road robot RAVON was introduced along with its physical parameters and the available sensor equipment. This test platform will be used in the following to validate the performance of the proposed methods in the real world. Furthermore, the robot's behavior-based 'pilot' for local navigation was described as far as this will become relevant for the interaction with the researched global navigation system. The chapter ends with the formalization of an earth coordinate system ECS and a temporary working coordinate system WCS. The ECS allows to specify place

Algorithm 2: SetupCSWithXAxis(Point3D *origin*, Vector3D *x_axis*, Vector3D *z_direction*)

Data: Point3D *origin*, Vector3D *x_axis*, Vector3D *z_direction*
Result: Coordinate Conversion Matrix M

Vector3D $x \leftarrow \text{normalize}\,(x\_axis)$
Vector3D $y \leftarrow \text{normalize}\,(z\_direction \times x)$
Vector3D $z \leftarrow \text{normalize}\,(x \times y)$

$$\text{Matrix } M = \begin{pmatrix} x_0 & y_0 & z_0 & origin_0 \\ x_1 & y_1 & z_1 & origin_1 \\ x_2 & y_2 & z_2 & origin_2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

return M

coordinates that remain unaltered even if the robot is reinitialized and is thus suitable for use in a global map. The WCS changes between initializations and is thus unusable for persistent maps, but allows to specify goal positions in a much more intuitive, local frame of reference. The mathematical tools required to transform between WCS and ECS are introduced as well as two general functions that will be used later to construct additional local coordinate systems.

3. Cost-Efficient Large-Scale Navigation

Whenever a mobile robot is required to navigate beyond its sensory horizon, it must either rely on potentially ineffective or misleading local search strategies (like the 'bug algorithms' [Lumelsky 87]) or use some kind of world model to store cues for navigation. Such a world model is normally referred to as a 'map' and can be provided a priori or build from experience. Many different map representations have already been proposed in literature.

Starting from a survey of existing map types in section 3.1, section 3.2 evaluates the inherent strengths and weaknesses of the most prominent map concepts with respect to the task of outdoor navigation. Section 3.3 then presents the basic map representation that has been proposed on the basis of this evaluation and formalizes it up to the degree required for precise discussion. This basic map concept will be extended further in chapter 4 in order to incorporate information for exploration. The current chapter remains restricted to navigation within a given map.

Since existing literature does not provide a suitable cost measure for the envisioned application in off-road terrain, a novel cost measure is developed and a methodology to learn increasingly accurate estimates for it through experience is proposed. Finally, the obtained benefit and flexibility of this learning scheme is experimentally validated using simulations and real-world experiments. The employed cost measure, the learning approach and the experimental validation results have been published in [Braun 08a]. The chapter ends with a conclusion and a critical review of the presented approach that points out avenues for future work.

3.1 A Survey of Existing Map Types

The map types for mobile robot navigation found in literature can be broadly divided into the three classes of **metrical**, **topological** and **hybrid** maps. In order to highlight the common benefits and weaknesses inherent in these different data representations, the focus of this section is placed on the map structures themselves rather than the algorithms

used for their creation. For a good survey and comparison of different mapping *algorithms* (at least for indoor applications), the reader is referred to [Thrun 02].

3.1.1 Metrical Maps

Purely metrical maps are probably the most common type of maps used for mobile robot navigation today. The distinction between metrical and topological representations is often somewhat vague. An attempt to provide a set of defining characteristics of what is considered a metric map in the scope of this thesis is made in definition 3.1.

Definition 3.1 *A map is considered to be* metric*, if it exhibits the following three characteristics:*

- *The map content is stored in one* global *frame of reference which defines a mathematical distance metric between any two map locations.*
- *The positional data is* unabstracted*, i.e. the accuracy of the internal map is comparable to the accuracy of the sensor data available to the mapping system.*
- *All locations are* equally important*. The mapping system does not model special places explicitly.*

The two most prominent metrical map types are grid maps and feature maps. Both are introduced below.

3.1.1.1 Grid Maps

Grid maps were among the first widely known and successfully used metrical map types for mobile robots and are still popular due to their simplicity and intuitive representation. Grid maps divide space into adjacent portions of equal metrical sizes. For a two-dimensional map, this results in a square grid, while the three dimensional grid map resemble a rubics cube. Both dimensionalities have been used [Elfes 89] [Moravec 96], but three-dimensional grid maps are rare due to their excessive storage requirements. Two major variants of grid maps are occupancy grids and elevation maps.

Occupancy Grids

Occupancy Grids, originally proposed by Moravec [Moravec 88] and Elfes [Elfes 89] are grid maps where each grid cell stores a probability value that specifies the estimated probability of this cell being occupied by an obstacle (see figure 3.1 for an example).

In the original approach [Elfes 89], a single, global occupancy grid was iteratively filled and updated using range measurements indicating the amount of free space in a certain direction up to the nearest obstacle. The measured distance was conditioned on a probabilistic measurement error model before being used to update the grid cell occupancy values (figure 3.2). This accounted for the typical noisy performance characteristics of the used sonar sensor and provided some degree of robustness against erroneous data. While the original probabilistic sensor model was derived by hand, later approaches learned it

Figure 3.1:
Occupancy grid map [Thrun 96b]
Each pixel represents one grid cell, darker colors indicate higher occupancy probabilities. Thus, white pixels denote free space, gray pixels represent unknown areas and black pixels stand for obstacles.

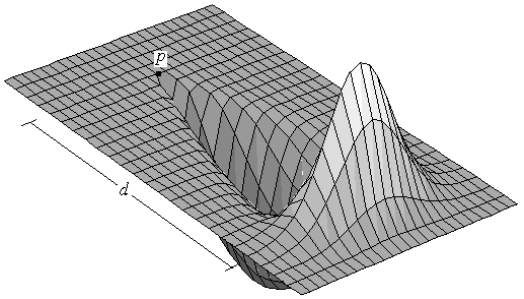


Figure 3.2:
Sonar sensor model [Moravec 96]
The drawing shows the quantitative distribution of obstacle probability as a function of distance, given a sonar distance measurement d taken from point p.

from examples in order to improve robustness and simplify adaption to different types of equipment [Moravec 93] [Thrun 96b].

Regardless of the employed sensor model, occupancy grids require accurate robot localization in order to incorporate new sensor data at the right location. Elfes proposed a correlation-based *map matching* algorithm for this - a locally built occupancy grid map surrounding the robot was iteratively matched with the global occupancy grid. The best match (based on a correlation of the grid cells occupancy values) provided a translational and rotational correction vector that could be applied to the robot pose estimate in order to compensate position errors. Although this approach can cope with small deviations, it is not able to localize the robot if it has no previous information on its approximate position without conducting a computationally intractable full search. Furthermore, the achievable relocalization accuracy depends upon the metrical size of the grid cells.

Since their invention, a lot of variations have been developed for occupancy grids and their related processing algorithms. A rather minor modification changes the type of sensor used as data source. While the original work obtained the range measurements using sonar sensors, other researchers opted to use laser range finders [Yguel 06] or stereo vision [Jennings 97] [Thrun 96b]. Further modifications have been spurned by the facts that the map matching procedure can only compensate local errors and is computationally very expensive. Hierarchical methods that use occupancy grids of multiple scale levels and perform a coarse-to-fine correlative map matching have been proposed to reduce the time required for map matching [Elfes 89]. Another speedup approach that has been proposed more recently harvests the parallel computational power of programmable graphical processing units (GPUs) [Yguel 06]. Yguel et. al. use the GPU as a specialized hardware component to apply the basic map matching scheme on high-resolution grids and multiple laser range sensors in real time.

Other approaches enhance the local grid map correlation scheme to allow global localization and to recover from catastrophic failures. [Fox 99] presents a global localization scheme based on the Markov assumption which effectively uses a 3D grid map of the robots configuration space (x, y, θ). Each cell stores the estimated likelihood that the robot has

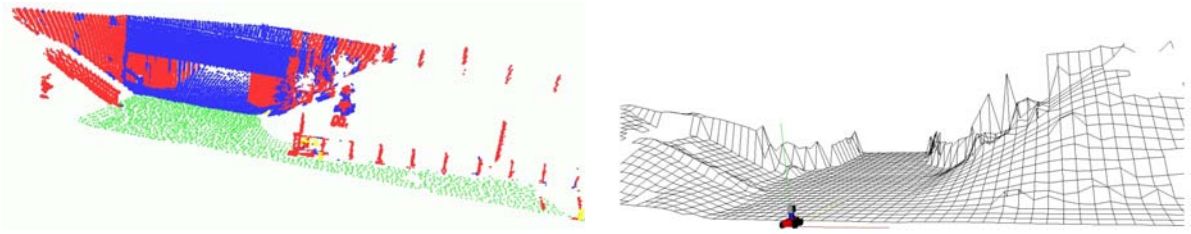
the corresponding pose. Sensor readings are then used to iteratively update these pose probabilities using Bayes theorem, making poses where the sensor readings agree with the stored (occupancy grid *or* geometrical primitives) map more likely, while poses inconsistent with the sensed data become penalized. Although this localization scheme can solve the global localization problem and is based on a sound mathematical foundation, it is even more computationally intensive than the original map matching, which only considers the local vicinity of the robot. Therefore, an alternative formulation based on a monte-carlo approach has been proposed [Fox 01] [Thrun 01], which does not explicitly generate a 3D grid map of all possible robot poses, but rather generates a finite amount of samples in the robots configuration space. The sample distribution thereby approximates the a posteriori probability of the robot pose, conditioned on the robots actions and its perceived sensor data.

Elevation Maps

Digital Elevation Maps (DEMs) are a variant of grid maps that share the regular spatial grid layout, but store *terrain heights* instead of occupancy probabilities per cell. They are often used as a replacement for occupancy grids in outdoor scenarios, because one can easily derive terrain slope from adjacent elevation map cells, which is a key factor for traversability analysis of outdoor terrain. Impassable obstacles are typically modeled (somewhat implicitly) as patches with very great heights, resulting in untraversable steep slopes. Most navigation approaches that use this representational form compute navigation commands in a two-step fashion; first an elevation map is produced based on a 3D point cloud generated using stereo-vision [Singh 99] [Thrun 06] [Hadsell 07a], laser range finders [Moorehead 01] [Ye 04] or both [Kweon 92] and second, the elevation map is analyzed to yield a grid map containing traversal costs. This traversability grid map can then be employed to compute the actual robot paths using A*-type algorithms [Ferguson 04]. Since the estimation of terrain height from point clouds requires several 3D points per grid cell, elevation maps often consist of rather large grid cells with side lengths ranging from 20 cm [Pfaff 05] up to 1 m.

A benefit of using elevation maps is the early abstraction from the full 3D information contained in the sensed point clouds and the resulting computational savings. However, the storage of a single height value per grid cell can only express *one* surface per grid cell. Therefore, terrain with overhanging geometry or multiple levels, such as large trees or bridges, cannot be modeled correctly by standard elevation maps. In order to counter this well-known problem, [Pfaff 05] recently proposed to classify single points of 3D point clouds into the four categories *traversable*, *non-traversable*, *vertical* and *vertical-gap* based on their context and compute an **extended elevation map** based only on the lowest *traversable* or *vertical-cap* points of each patch (see figure 3.3). [Triebel 06] extends this approach further and proposes **multi-level surface maps**, which store multiple surfaces per grid cell instead of a single one. The segmentation of 3D distance measurements into multiple surfaces is based on the classification scheme of [Pfaff 05] combined with a distance heuristics.

While both approaches eliminate overarching points from the sensor data and thus extend the applicability of elevation map to terrain containing overhangs, they need to classify each sensed range point prior to map generation, resulting in increased computational effort and an additional source of error. Both classification schemes are furthermore biased



(a) Point classifications (b) Extended elevation map

Figure 3.3: Computing an extended elevation map in front of a bridge ([Pfaff 05])
(a) shows the classification of a 3D point cloud into traversable (green), vertical (red) and vertical-gap (blue) points. (b) shows an elevation map computed using only traversable points from (a).

toward vertical structures, which are typical for man-made constructions such as bridges, but might be too restrictive for complex natural environments like cluttered forests.

Since elevation maps are dense metrical maps like occupancy grids, they too require accurate localization in order to incorporate new data at the right location. However, published approaches typically do not use a map matching procedure based on height values, which would be analogous to the occupancy grid method. Instead, some rely on absolute sensor systems available outdoors such as GPS combined with wheel odometry and accept the positional error that cannot be eliminated by this approach. Others use an additional feature-based map (see section 3.1.1.2) for localization. The reasons for not using elevation maps directly for localization are not clearly stated, but two possible causes might be that the terrain height estimates are either too unstable under different viewpoints or the grid map resolution is simply not fine enough for acceptable relocalization.

3.1.1.2 Feature Maps

In contrast to grid maps, feature based maps do not represent space uniformly with equal precision, but store explicit *features* and their spatial locations instead. Since feature positions can be stored with arbitrary accuracy and are not blurred by any discretizing grid, feature maps support precise robot (re-)localization well. Typical feature based localization approaches proceed in three phases. After an initial *feature identification* step, which detects relevant features from current sensor data, a *data association* phase establishes correspondences between currently sensed features and those already stored in the map. Using these correspondences, *pose estimation* of the own position relative to the map can be performed.

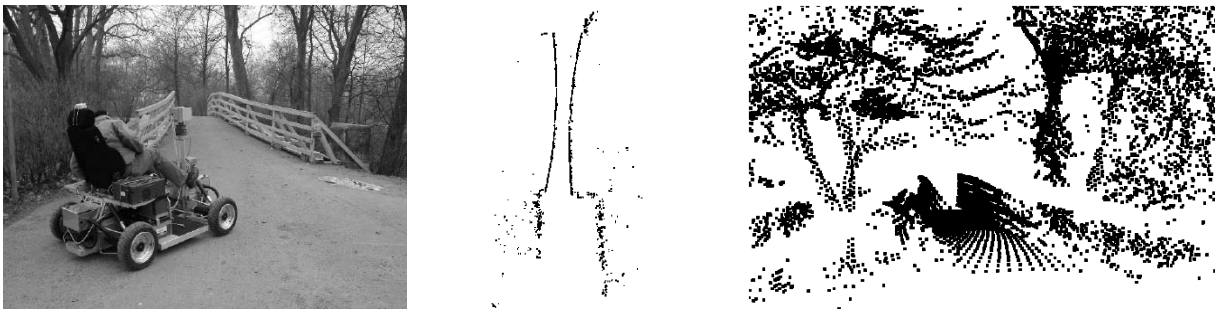
Many different types of features have been proposed as building blocks for feature based maps. They vary broadly in distinctiveness and range from simple, indistinguishable **basic features** up to highly expressive **landmarks**. Since the level of feature distinctiveness also influences the applicable range of algorithms and suitable data structures, both subclasses are treated separately in the following paragraphs.

Basic Features

Arguably the simplest entities usable for feature based maps are surface points stemming from range sensors such as laser range finders or stereo cameras. These sensors quickly provide range measurements in larger numbers and with much higher angular precision

than ultrasonic sensors. Due to the increased accuracy, the measured distances can easily be projected onto points in three-dimensional space, yielding point clouds outlining terrain and obstacle surfaces. These *point features* (also referred to as *point obstacles* by [Thrun 02]) can be directly used to construct a **point feature map**. It is also possible to construct a basic feature map from an existing occupancy grid map by constructing point features for all grid cells which exceed a given occupation probability threshold.

Figure 3.4 shows two examples taken from [Brennecke 03] of a real outdoor scene plus two and three dimensional point clouds created by using a panning laser scanner.



(a) Original scene (b) 2D point cloud (c) 3D point cloud

Figure 3.4: Examples for point clouds usable as basic feature maps

A great benefit of point feature maps is that they do not require *any* feature identification or classification stage, since each sensor measurement is directly used as a feature. The downside of this is that each sensor quickly produces a lot of features and consequently, computational operations on these maps are typically slow and memory taxing. Also, association between stored maps and fresh data can only rely on location-based heuristics to assign point correspondences. This leads to the requirement of a good initial pose estimate for robot relocalization and complicates global relocalization approaches based on point feature maps.

Nevertheless, point feature maps are quite often utilized for robot localization in both indoor and outdoor environments. In order to determine the pose of a robot given such a map, approaches such as [Nuechter 06] and [Brennecke 04] match the stored map M containing 3D point feature coordinates m_i in a global coordinate frame with the data set D of currently sensed point obstacles d_i, projected into global coordinates using a robot pose estimate p. The goal is to find the rotation R and translation t, which maximizes the overlap between the point clouds of D and M. The computed affine transformation can then be applied to correct the robot pose estimate p in a fashion quite similar to the map matching procedure proposed for occupancy grids (Section 3.1.1.1).

Mathematically, R and t are computed using a *scan matching algorithm* that computes a rotation R and translation t which minimizes the summed distances of corresponding point features:

$$\arg\min_{R,t} \sum_{i=1}^{|M|} \sum_{j=1}^{|D|} w_{i,j} \|m_i - (Rd_j + t)\|^2, \tag{3.1}$$

where the association weights $w_{i,j}$ are equal to 1 if the features m_i and d_j correspond to each other and 0 otherwise.

In 1992, [Besl 92] proposed a two-step iterative method now called the Iterative Closest Point (ICP) algorithm to perform both the data association and the minimization of equation 3.1 on the basis of two point feature maps. To fix the values of $w_{i,j}$, the algorithm heuristically assumes that the feature m_i corresponds with the *spatially closest* feature d_j, if their mutual euclidean distance remains below a threshold d_{max}. Then, R and t can be computed using a variety of different methods. While [Besl 92] used the Levenberg-Marquardt numerical optimization technique for this, other researchers have proposed spring-relaxation models [Eggert 98] or closed-form analytic solutions summarized and compared in [Lorusso 95] [Eggert 97].

Experience has shown that the concrete method used for the computation of the affine transformation (R, t) has little impact on the final localization result [Eggert 97]. In contrast, the selection of correct correspondences between point features is crucial for both convergence speed and successful convergence. Therefore, improved heuristics based on the basic ICP approach have been proposed. [Lu 94] presents the 'Iterative Dual Correspondence' (IDC) heuristics, which alternates between finding point feature correspondences using the ICP rule and a new rule called iterative matching-range-point (IMRP), which selects feature pairs which both lie close together *and* have the same distance from their respective feature sets center of gravity. As motivation behind this approach, Lu et. al. state that IMRP does not tend to produce correspondences which cancel out the rotational component R of the affine registration transformation as ICP commonly does. Therefore, the IMRP quickly converges to the right rotation, while the alternating use of ICP takes care of calculating the correct translation.

Data structures for point maps are required to handle both a large number of features and support efficient lookup of the closest neighbors for each candidate point within a maximum distance. The obvious, brute-force search through a complete list results in a complexity of $O(n^2)$ for n data points and is inapplicable for real-time navigation. Consequently, point feature maps are preferably stored in tree-like data structures that simplify neighborhood searches instead of plain feature lists. [Nuechter 06] presents a summary of different implementations and concludes that a kD-tree which generalizes binary search trees to multiple dimensions is a suitable representation. kD-trees recursively partition space along one coordinate of the k-dimensional space and can quickly exclude whole subtrees from neighbor searches, if a candidate is sufficiently far away from nearby partition boundaries. To exploit this characteristic to the fullest, [Nuechter 06] proposes to partition space along the dimension with the largest extent and provides some speed bounds for this case.

Landmarks

While basic features are practically indistinguishable from each other except by their spatial position, **landmarks** can be defined as highly distinctive sensor 'anomalies' which can be detected *and* uniquely identified from a large collection of sensorial data. Landmark detection and identification typically requires the use of an information-rich sensor, such as a color camera, and the implementation of sophisticated pattern recognition methods.

Distinct landmarks are much rarer than basic features and cluster around places with special environmental characteristics, so they are distributed rather irregularly. This property can be an advantage if the mapped features are also relevant for the navigation

task at hand, because in this case, the resulting maps focus the robots computational effort onto relevant data quite naturally. On the other hand, landmark maps typically lack coverage of sensorially poor parts of the environment, which makes path planning or traversability analysis in these areas difficult. Indeed, their primary application is robot localization, but in contrast to basic feature maps, landmark maps are much more compact and less affected by the data association problem. This extends the range up to which successful localization can be performed and allows the application of algorithms that spend more computational effort per feature than ICP-type approaches.

Early uses of landmark based maps for robot localization include geometrical triangulation and trilateration techniques. [Sugihara 88] presents several localization approaches based on known landmark positions and angles between them observed by a single camera. [Avis 90] extends this algorithm to take uncertainty into account, while [Betke 94] uses a complex number representation to reduce triangulation runtime complexity to $O(n)$ for distinguishable landmarks.

Another prominent class of methods that relies on landmark based maps employs the Extended Kalman Filter (EKF) for simultaneous localization and mapping (SLAM). The Kalman Filter is a general, statistically motivated technique for the estimation of a system's state using both an (approximative) model of the system's future behavior and potentially noisy measurements of some observable aspect of the system. Building upon initial work from [Smith 90], most kalman-based techniques such as [Dissanayake 01], [Tardos 02] or [Andrade-Cetto 01] choose the current position of a robotic vehicle and the estimated location of all known landmarks as the system state and use measurements of relative distances between the robot and landmarks as measured input (see figure 3.5).

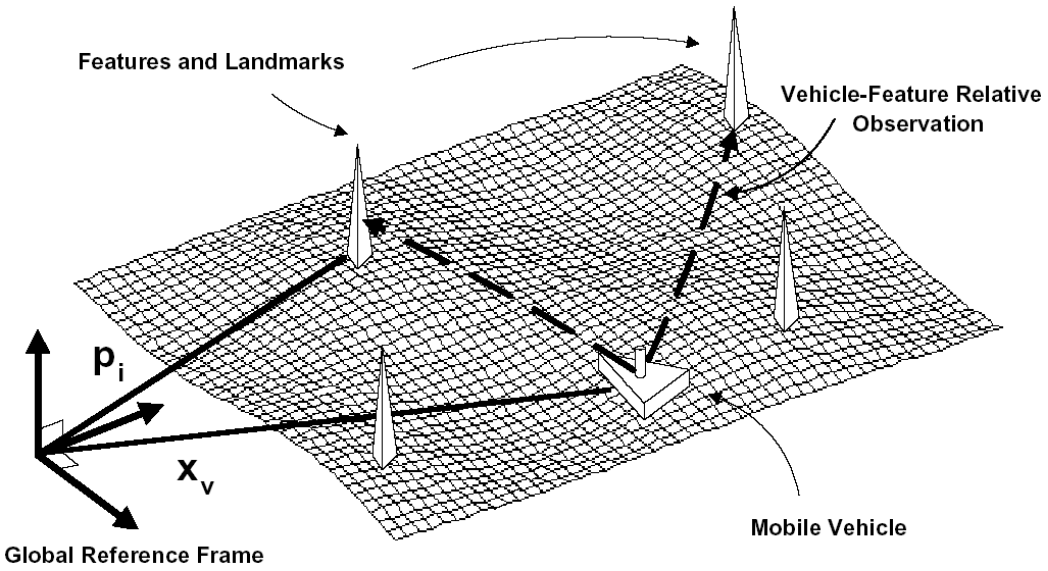


Figure 3.5: SLAM with landmark feature maps (adapted from [Dissanayake 01])
[Dissanayake 01] provides theoretical proof that both robot pose x_v and landmark positions p_i can be determined accurately (down to the initial pose uncertainty with respect to a given global frame of reference) using relative robot-landmark distance measures.

Based on theoretical considerations, [Dissanayake 01] presents a proof that both the robot pose and the position of all landmarks can be completely and accurately determined (down

to the initial pose uncertainty with respect to a given global frame of reference) given only such relative distance measurements. [Dissanayake 01] validates the theoretical claims in a practical experiment using a radar sensor mounted on an off-road car and special reflectors as landmarks. In contrast, the approach of [Andrade-Cetto 01] is based on natural, visual landmarks. The presented approach adds some precautions against dynamic changes in the environment, as each landmark is annotated with an *existence state* quantifying the frequency and stability of the landmark recognition. Using an exponential decay law, vanished or unstable landmarks with a low existence state are eventually deleted from the feature based map, while newly detected landmarks are incorporated.

More recently, research effort has focused on computational improvements for kalman-based localization techniques. In order to reduce the complexity from $O(n^2)$, where n is the number of all known landmarks, several researchers have proposed to exploit locality of landmark groups and restrict the application of the kalman filter to nearby landmarks. [Guivant 01] considers the special, block-diagonal form of the kalman covariance matrices and presents a new 'compressed kalman-filter' that reduces the computational requirements to $O(n_a^2)$, where n_a is the number of landmarks in the local area around the robot. [Knight 01] approaches the complexity problem by postponing full matrix updates until the robot has sufficient computational time available and updates a constant-sized data set based on current measurements in the meantime.

3.1.2 Topological Maps

The different metrical map types presented in the last section support precise robot localization in a global frame of reference and accurate path planning. Unfortunately, the highly detailed world representation requires a lot of memory space and leads to algorithms with high computational demands. Also, the incorporation of new sensor information depends highly on the quality and accuracy of the current position estimate. If the robot's pose estimate is bad during a map update or if incorrectly sensed or outdated information is inserted without precaution, the metrical world map can become seriously corrupted. These properties limit the scalability of purely metrical mapping techniques [Brooks 87].

Motivated by these drawbacks, researchers aiming at robust, large scale navigation have started early to look at the **topological** type of world model, which represents the environment in a more compact, qualitative fashion. Topological approaches focus on representing navigation-relevant places and their connections on an abstract level rather than the exact metrical layout of the surroundings. Thus, imprecise localization is less of a problem for topological approaches and algorithms can run faster because they have to cope with much less data.

Topological maps commonly use graphs as underlying data structure. Graph nodes identify locations of interest and their characteristical features, while knowledge about travel between nodes is encoded in the connecting graph edges. Analogous to definition 3.1, definition 3.2 tries to clarify what is considered a topological map in the context of this thesis.

Definition 3.2 *A map is considered to be* topological*, if it exhibits the following two characteristics:*

- *The map explicitly models locations having navigational significance and omits places that have not.*
- *The stored information is too abstract to fully support local navigational capabilities such as obstacle avoidance or trajectory generation.*

Apart from robot navigation, topological maps are also interesting from the perspective of cognitive science, as findings from psychology indicate the presence of similar maps in higher vertebrates [Chown 95] [Steck 99]. Therefore, a body of research on topological maps already exists in this area from which roboticists can draw some inspiration. Interestingly, researchers from psychology have also started to use robots in order to test the plausibility of new hypotheses about how animals develop navigational competences. This interdisciplinary area of 'developmental robotics' has already obtained interesting results [Lungarella 03].

An influential, early topological mapping procedure is presented in a series of papers by Kupiers and Byun [Kuipers 91] [Kuipers 00]. Spatial knowledge is modeled as a hierarchy of increasingly abstract and powerful representations, called the Spatial Semantic Hierarchy (SSH). The lowest level of the SSH consists of *distinctive states*, which are isolated fixed-points of hill-climbing control laws available to a robot. This property guarantees that the robot can 'home-in' on the distinctive location using the appropriate control law and thus allows the compensation of positional errors upon arrival at a distinctive state. Travel between states can be effected by a sequence of different control laws, which is abstracted to single travel *actions*. The recognizability of distinctive states is verified using a *rehearsal procedure* [Remolina 04] and appropriate signatures for their sensor-based detection are learned automatically [Kuipers 02]. Then, a topological map is created using stable distinctive states as graph vertices and the corresponding travel actions as topological edges.

The success of topological navigation techniques depends largely on the reliable detection of proximity to stored topological nodes. Since this issue of *place recognition* is so important (see [Duckett 00] for a comparison of various landmark detection techniques), most authors opt to use an approach similar to kuiper's distinctive states and place map vertices at locations easy to detect and 'home-in' to. Some researchers select suitable places manually and use for example wall corners or corridor intersections as distinctive locations for graph nodes [Matarić 92] [Kuipers 91]. This of course restricts the applicability of their navigation strategies to surroundings that exhibit such configurations.

To avoid this problem, other scientists let the robot decide on its own about what constitutes a suitable place. This decision is based on an appropriate measure of distinctiveness. For example, the method presented in [Nehmzow 00] uses sonar sensor readings as characteristical place fingerprints and inserts new topological nodes whenever the similarity of the current sensor impression to all known signatures drops below a preset threshold. Similarity is calculated using a Reduced Coulomb Energy (RCE) classifier, which in contrast to other popular classification techniques such as self-organizing maps, has the capability to grow on demand and thus incorporate new place signatures without limit. Upon detection of a known place, a *centering behavior* is activated, which tries to minimize any residual differences between the place signature and the current sonar readings

by moving the robot toward the place center. This technique corresponds directly to the hill-climbing control laws presented by [Kuipers 00].

Another approach which selects suitable topological nodes autonomously based on their distinctiveness is presented by Zimmer [Zimmer 96]. Here, an adaptive neural network is used in which each cell represents a situation consisting of odometry position and current sensor impressions. New places are added to the network if the difference between the current situation and its closest match exceeds some threshold. This approach simultaneously guarantees the addition of new places with sufficiently distinctive sensor data *and* establishes an upper bound on the metrical distance between nodes, as the node position itself is part of the place signature. [Yamauchi 96] even goes a step further and reduces the place signature only to its metrical position. This precludes any sensor-based homing behavior and thus requires another means for correcting positional errors, but relieves the robot from interpreting exterioceptive sensors for the purposes of topological navigation altogether.

Aiming at a biologically plausible navigation method, [Franz 98] and [Franz 00] propose a view based algorithm where topological places are identified using a *visual* fingerprint obtained by an omnidirectional camera. Whenever the current visual fingerprint is sufficiently distinct from the last recorded one, a new place node is instantiated and connected to the last node via a new edge. Exploration is conducted by driving in the direction of the angle bisector of the current nodes' two edges having the largest opening angle. The constructed topological map consists of nodes defined by the characteristic visual fingerprint and edges only defined by their exit angles. Upon reaching a known node, a visual homing strategy is initiated to minimize positional drift. This approach exhibits strong parallels to some navigational capabilities observed with desert ants [Lambrinos 00] [Wehner 03], which have been found to home in onto their nest entrance using approximately omnidirectional visual cues. Unfortunately, the proposed approach cannot cope with the problem of *perceptual aliasing*, which occurs whenever two distinct places have identical sensor signatures. If this happens, a erroneous transition to the place already stored in memory is detected and inserted into the world model.

3.1.2.1 Probabilistic Approaches

The problem of perceptual aliasing and erroneous place recognition has been identified as a key challenge for topological navigation approaches. One obvious possibility to improve the robustness against such errors is to increase the discriminative power of the robot sensors. [Kortenkamp 94] presents one of the first methods aiming in this direction. The publication describes a fusion algorithm which combines sonar readings with visual cues using a simple Bayesian network. Experiments in indoor environment demonstrate that this multi-modal place recognition approach can significantly improve recognition rates of distinct places. However, as persuasively argued in [Nehmzow 00], more sensors can only reduce perceptual aliasing, but never eliminate it entirely. After all, there really *are* places that look similar, even to humans.

Another strategy to cope with perceptual aliasing is to accept the fact that some topological places cannot be distinguished from each other, but try to use previous positional knowledge to remain localized even if perceptual aliasing occurs. To implement this, several researchers have proposed to use a probabilistic model of the robot's place belief

state, i.e. to record separately *for each place* how strongly the robot currently believes to be there. Upon receiving new sensor information, the belief state can be adapted by increasing the probability of matching nodes and decreasing the others. A key aspect of this approach is that the probability adaption is also conditioned on the previous belief state. This allows to transfer past (more exact) positional knowledge over to ambiguous situations. With this model, a navigation system can gradually increase its certainty about where it is over time by integrating several subsequent place detections. At the same time, information about the current quality of localization is retained and temporary confusion due to ambiguous situations (where multiple nodes have high probability) can be overcome again later.

[Vale 05] presents such a probabilistic topological navigation system that employs vision as primary sensor for place recognition. Formally, the task of localization is modeled as a partially observable Markov decision process (POMDP). If perceptual aliasing occurs, the POMDP model eliminates false place hypotheses based on its previous belief state and potentially unlikely place transitions. If this is not sufficient, the belief state becomes ambiguous, but can be recovered with future correct place recognitions. In addition to this probabilistic model, [Vale 05] allows the robot to choose between a whole range of different visual features for the place signature in order to select the most distinctive features at each location.

Another topological approach using POMDPs is presented in [Tapus 05]. Place signatures are constructed from distinctive color patches and vertical edges recorded by an omnidirectional vision system and corners detected by laser range finders. All found salient features are represented as a signature string, or 'fingerprint' of a topological node. Similarity of different fingerprints is estimated by applying string matching algorithms originally developed for DNA sequence analysis to the place fingerprints. Based on the currently sensed fingerprint and its similarity measures, the robot's place belief state is updated using the probabilistic POMDP model. To extend the topological map, new nodes are inserted whenever the place signature is sufficiently distinct to already stored nodes *and* the belief state is unambiguous. This prevents the robot from adding new topological nodes in situations where its own position is unclear.

A hidden Markov model for probabilistic place recognition is formulated in [Filliat 02]. This work explicitly distinguishes between topological relocalization *within* a known topological map and the task of topological mapping, that takes place whenever the robot explores unknown territory. If the probability of being inside an already mapped area is high (estimated by summing the belief state probability values of all nodes and thresholding), a multi-hypothesis localization model similar to the already mentioned POMDPs is used for global localization. Otherwise, it is assumed that the robot is exploring, and a single-hypothesis approach is adopted, that assigns the robots place to the best matching signature in the vicinity of the last position. If the best match is too bad, a new node is created. Filiat et. al. state that this double-algorithm strategy efficiently copes with the problem to decide whether localization is ambiguous because of perceptual aliasing or because the robot has indeed left the known territory. While the latter case requires the addition of new topological nodes, such an action is not warranted in the first case.

The focus of Yairi et al. [Yairi 02] is placed on map building rather than place recognition. Their approach tries to derive a consistent topological map from a series of action / observation data pairs obtained during an exploration phase. Places are initially created

through k-means clustering of sensor data, thus based purely on signature distinctiveness. As a result, perceptually aliased places are erroneously represented by a single, so-called *compound* state. In order to fix this, three entropy criterions are computed for each node quantifying the uncertainty of which state transitions will occur upon execution of a driving action. Large entropies are seen as an indication for a compound state and the corresponding place node is split into two places so that the entropy criterions are minimized. An indoor experiment validates that the proposed clustering and splitting approach can generate a topological map in which only distinct topological nodes remain.

3.1.3 Hybrid Maps

Recently, many researchers have proposed to attack the problems of autonomous mapping, localization and navigation using combinations of the metrical and topological methodologies presented in the last two sections. Generally, these *hybrid* approaches are designed to combine the benefits of both representational forms, ideally allowing localization and map building with the high precision of metrical maps while retaining the computational tractability and compactness of topological data structures.

When trying to classify the techniques found in literature, two main types of hybrid mapping strategies can be distinguished. One type is characterized by the use of *abstraction*. This class of approaches typically constructs a metrical map of the (local or global) environment as a basis, and then abstracts this map in order to create a compact topological representation. One of the benefits of this abstraction is that it allows more efficient planning of an approximate path to a given goal location than a detailed metrical map. However, the metrical map must often be kept for relocalization and obstacle avoidance purposes.

The other class of hybrid mapping methods does not use abstraction to derive one map type from another, but arranges the two map methodologies in a *hierarchical* fashion. This is accomplished by creating several local metrical maps with a limited scale each and tying them together using a global topological map (figure 3.6). This approach elegantly uses the classical divide-and-conquer paradigm to address the scalability problems that are inherent in large metrical maps. Also, it prevents errors incurred during metrical mapping from spreading over the entire mapped area. However, one also has to pay a price for the segmented map structure, as information contained in partially overlapping local maps cannot be used to enforce global consistency. This can lead to increased uncertainty for each local map, especially if they are closely spaced.

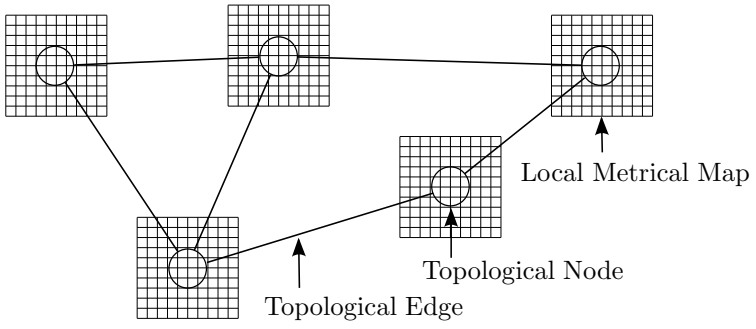


Figure 3.6: Schematics of a hierarchical hybrid map

In the following, some prominent abstraction based and hierarchical hybrid mapping strategies are presented.

Abstraction-Based Hybrid Maps

It is possible to construct a topological map purely based on the abstraction of immediate metrical range measurements that describe the local vicinity. [Choset 01] advocates the **Generalized Voroni Graph** (GVG) as a suitable abstraction, capturing the topology of the free space in the mapped environment, yet avoiding its extensive metrical modeling.

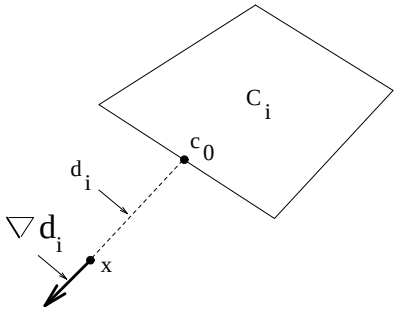


Figure 3.7: Distance and gradient used for GVG construction [Choset 96]

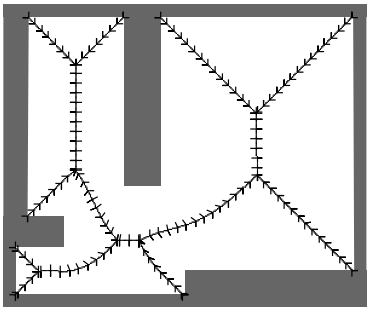


Figure 3.8: Example of a generalized voronoi graph [Choset 96]

The GVG is defined as the set of points in free metrical space equidistant to (at least) two obstacles. Formally, given a distance function d_i which measures the distance of the point x to the closest point on obstacle C_i: $d_i(x) = min_{c \in c_i}|x - c|$, a *GVG edge* is determined by two obstacles C_i and C_j as the set

$$F_{ij} = \{x : d_i(x) = d_j(x) \leq d_h(x) \text{ such that } \bigtriangledown d_i(x) \neq \bigtriangledown d_j(x)\}, \tag{3.2}$$

where $d_h(x)$ denotes the distance to any third obstacle C_h, and $\bigtriangledown d_i(x)$ the gradient of $d_i(x)$ (see figure 3.7). Furthermore, the end points of a GVG edge F_{ij} are termed **boundary points** for $d_i(x) = d_j(x) = 0$ and **meet points** for $d_i(x) = d_j(x) = d_h(x)$ with at least one h (see figure 3.8). [Choset 01] presents control laws that can be used to drive toward the generalized voroni graph from an arbitrary metrical position, traverse GVG edges and home in on GVG meet points. These control laws bear strong resemblance with those used by Kuipers for distinctive states - which is not surprising, as the underlying idea is essentially the same. [Choset 01] also introduces four stable features that can be used for topological relocalization by identifying the meet point the robot is currently at. These features are

1. the distance d to the closest obstacle,
2. the number of neighboring boundary points,
3. the number of edges connected to the meet point and as final criterion,
4. the relative departure angles of these edges.

If after arriving at an unknown meet point and matching these four features, more than one potential candidate remains, the robot proceeds to traverse another GVG edge and then reduces the current candidate set by intersecting it with the neighbors of the new candidates calculated at the target meet point. Of course, this strategy can fail, too, for *hyper-symmetrical* environments (where all meet points look the same), but this cannot be circumvented by topological approaches.

[Zwynsvoorde 00] presents another mapping technique based on the generalized voroni graph. The innovation of this work lies mainly in the used sensor model. Instead of using sonar sensors as done by [Choset 01], [Zwynsvoorde 00] employs a standard 180° planar laser scanner. With the increased angular resolution, it becomes possible to determine areas that could not be sensed completely and thus warrant further exploration. By adding virtual GVG edges and meet points, these explorative areas are incorporated in the topological map during mapping and can be queried for exploration strategies.

A good example of a hybrid mapping strategy that derives a voroni-graph based topological map through abstraction of a long-lived, complete metrical map is presented in [Thrun 98a]. Here, topological map building is initiated by thresholding an occupancy grid map built using Bayesian probability techniques. Then, the voroni diagram is built by selecting all free grid cells, whose two nearest occupied grid cells (the *base points*) are equidistant. Cells on the voroni diagram are termed critical points, if the base point distance is a local minimum. Lines between critical points and their base points divide metrical space into separate topological regions at locally narrow passages. From this regional decomposition, a topological graph can be generated easily (Figure 3.9 shows an example of these steps).

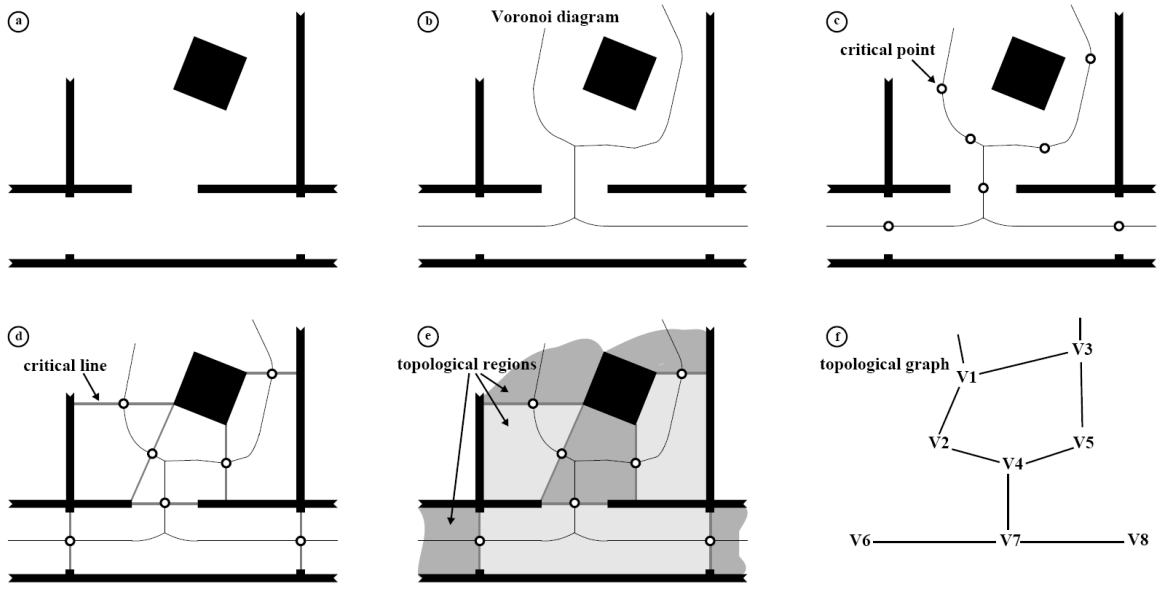


Figure 3.9: Steps in the topological abstraction scheme of [Thrun 98a]
a) Original thresholded occupancy map. b) Overlaid voroni diagram c) critical points d) critical lines e) topological regions f) resulting topological graph

After construction, this graph can be used for quick path planning without resorting to the detailed, underlying metrical map. During travel however, the arrival at a topological node can only be detected by using localization techniques based on the detailed map,

since the topological nodes are only characterized by their spatial position in map's frame of reference. Thus, while the topological representation improves path planning efficiency, this hybrid solution does not address the scalability issues faced by global metrical mapping techniques.

Hierarchical Hybrid Maps

[Tomatis 03] presents a technique that hierarchically combines a global topological map with local metrical feature maps for mapping and navigation in indoor environments. On the global level, a topological map is constructed which contains intermediate *nodes* spanning the topological graph, leafs that signify metrical *places* and *corner lists* on the links between nodes (see fig 3.10). The corners can be extracted easily from the data of a 360° laser scanner and serve as landmarks for a POMDP approach (see 3.1.2.1) performing localization on the topological scale.

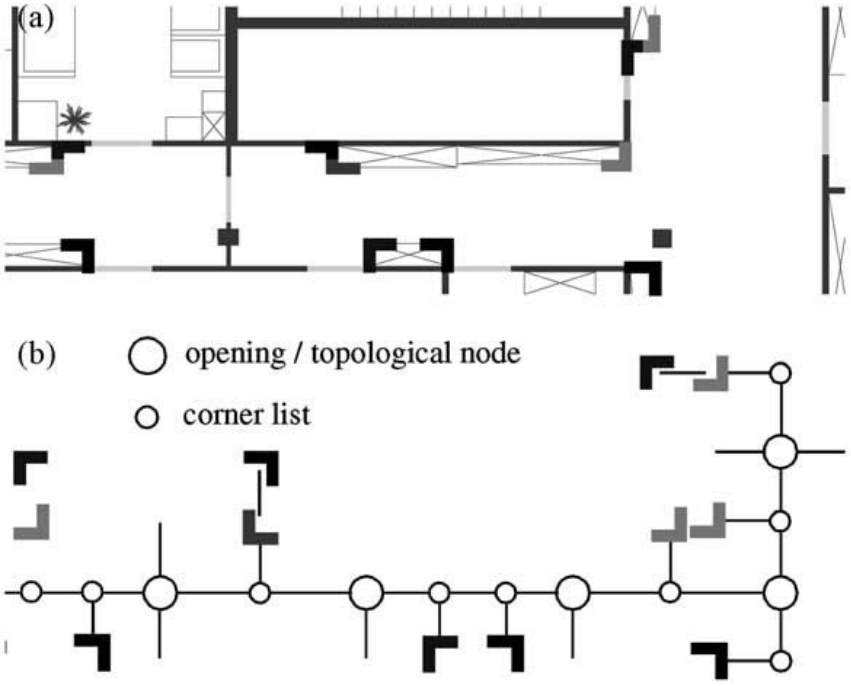


Figure 3.10: Hierarchical hybrid map of [Tomatis 03]
a) shows a portion of a hallway with extracted corner and opening features. b) represents the resulting topological map with nodes and corner landmarks. Open arches either lead to metrical places or other hallways.

While the POMDP topological localization strategy is used while the robots travels along map edges, the algorithm switches to a metrical method once the robot has arrived at a leaf 'place'. The local map stored in this leaf is a feature map as introduced in section 3.1.1.2 and is constructed using lines extracted from the 360° laser scan. For each line segment found in the scan, an 'infinite line feature' is generated that is described by the angle of the perpendicular to the line and its length. The local feature map can be used for localization by modeling all feature parameters and the current robot pose as the state vector of an extended kalman filter and using a standard EKF based SLAM algorithm (see sec. 3.1.1.2). With the use of this local metrical representation, the robot can travel

freely in the vicinity of the leaf node, performing navigational tasks with high metrical precision.

During exploration, a strategy must be defined that decides when to create a topological node and when to switch to the metrical paradigm. [Tomatis 03] arguments that for office scenarios, high precision is mainly required in rooms. So, it is sufficient to set up topological nodes for each opening detected in a *hallway* (a narrow and elongated space in the laser scan) and create leaf places once the robot has entered a room (a wider, more quadratic space).

Unfortunately, this strategy, along with the use of line features for the metrical feature map, is strongly tuned toward typical indoor environments. Extending this approach to more unstructured off-road scenarios therefore appears rather complicated.

A more general hybrid mapping framework not geared toward office environments is the ATLAS framework presented by [Bosse 03]. The framework's main concept is to connect local metrical maps via a global topological adjacency graph. However, the exact method used for metric mapping is not directly dictated by the framework. Although Bosse et. al. propose to use a feature based map containing point and line features, the only hard requirements for the metrical component are the capability to perform *localization* and *map matching*. While the need for localization is obvious, the map matching capability that identifies common features and estimates the similarity of two metric maps is required to detect loops during exploration.

Once a suitable local map representation is selected, the robot builds such a map using SLAM until the map complexity (f.e. number of features) exceeds a certain threshold. Then, a new local map is initialized at the estimated current robot position, and a topological edge is created that links the old and new metrical frames. [Bosse 03] proposes to store the available information about the current positional uncertainty in this edge, effectively annotating each edge with a covariance matrix expressing uncertainties in relative location and orientation between the two local maps. This information can be used for path planning in the topological map: By choosing the magnitude of the uncertainties (the determinants of the covariance matrices) as edge traversal costs, the dijkstra path planning algorithm can be used to plan the *most precise* paths from a start map to each local map node.

If map matching detects large similarities between the current map frame and an already existing local map within the current positional uncertainty bounds, a loop is assumed, and the best alignment of both map frames is computed using a least-squares fit of all matching features. Then, the existing map is updated with the information extracted from the other, which is subsequently discarded, and the topological loop is closed.

The ATLAS framework supports multi-hypothesis position tracking like the POMDP approaches, but uses a unique strategy to achieve this. *Each* local map frame can hold *one* hypothesis about the robots current pose. If this hypothesis is supported over several time steps with sensor data corresponding to the information stored in the local map, the hypothesis can become *mature*. The best mature hypothesis is then reported as the current robot pose. If the last mature hypothesis fails to be supported well by current sensor data, a new local map is created as the robot has probably moved into a new and unexplored area of the environment.

While ATLAS does not use any assumptions on the environmental characteristics such as the approach of [Tomatis 03], it relies more strongly on metrical information. ATLAS' performance depends on the accuracy of the positional uncertainty estimate both for path planning and loop closing. This imposes a strong constraint on the minimal acceptable quality of the metrical mapping components.

Two further hybrid mapping algorithms are presented in [Lisien 03] and [Estrada 05]. Both propose to use feature maps and the extended kalman filtering approach on the local level for precise metrical navigation. On the topological scale, [Lisien 03] suggests to employ the generalized voroni graph (GVG) already presented in conjunction with the abstraction based hybrid maps and anchor the metrical maps to the meet points of the GVG. The approach is somewhat incomplete as important issues such as loop closure are not addressed, and experimental validation is simplified through the use of artificial landmarks.

[Estrada 05] does not use a voroni graph to structure the arrangement of local maps, but restricts the number of allowed features per map similar to [Bosse 03], prompting the creation of new maps in regular spatial intervals for uniformly featured environments. The proposed algorithm's main strength is efficient loop closing. If a loop closure is triggered by a good match between an older and the current local map, the algorithm first constructs a global correlation matrix from all involved local map correlation matrices that have been computed previously by the extended kalman filter. Since the local maps are not correlated with each other, the global correlation matrix P is block-diagonal. P can then be used in a nonlinear constrained optimization that enforces the global metrical consistency in the closed loop, while the diagonality property allows the efficient solution of the optimization problem with linear complexity in the number of involved local maps.

3.2 Analysis of Map Suitability

While the different mapping algorithms presented in the last section vary substantially in their level of sophistication and the resulting map's quality, their exposition still allows to make some principal statements about which kind of map is better suited for different navigation-relevant applications (such as localization or path planning). The underlying map structure also has a large influence on the choice of algorithms that can be used. Table 3.1 summarizes these common characteristics.

With the survey of different mapping approaches as background, the question can be approached which of the presented map paradigms is the best choice for the envisioned deliberative off-road navigation layer. As this layer constitutes the bridge between the behavior-based piloting layer and the operation environment, the decision is influenced by the properties of both sides.

The target off-road environment is characterized mainly by its large spatial extent, its lack of structure and a complex three-dimensional layout. The large size of the operation area weights heavily against the construction of a single, dense metrical map such as an occupancy grid or elevation map, as this would require vast amounts of data storage and processing power. Also, the construction and maintenance of such a representation requires accurate knowledge of the robot's location and, to ensure map consistency, traversability estimates that are viewpoint independent. Both requirements appear to be

Map Type	Data Structure	Primary Application	Prominent Algorithms
Occupancy Grids	2D array	Indoor Trajectory Planning Localization	A*, Markov, MCL [Thrun 98b]
Elevation Maps	2D array	Outdoor Trajectory Planning	A*, D*, Extended Elev. Maps [Ferguson 04] [Pfaff 05]
Basic Feature Maps (Point Clouds)	kD-Tree	Localization	Scan-Matching, Pixel-SLAM [Besl 92] [Brennecke 03]
Landmark Maps	List	Localization	EKF-SLAM [Dissanayake 01]
Topological Map	Graph	Place Recognition Path Planning	POMDPs [Kuipers 00] [Vale 05]
Abstracted Hybrid Maps	Graph	Path Planning Localization	Voroni Graphs, GVG [Choset 01] [Thrun 96a]
Hierarchical Hybrid Maps	Graph + Feature Lists	Path Planning Localization	POMDPs + EKF-SLAM [Tomatis 03] [Bosse 03]

Table 3.1: Application domains for different map variants

difficult to ensure in highly vegetated areas that can exhibit large odometric drift and strong variations in sensor penetration.

Feature-based mapping approaches that rely upon the extraction of distinct landmarks can also be expected to face large difficulties once the robot leaves park-like environments with spatially well defined landmarks such as single trees. In more cluttered areas, the high variability of natural vegetation will pose problems for landmark recognition tasks, and one cannot simply rely on the presence of regular and easier detectable man-made structures. In contrast to this, mapping approaches using simple features such as point clouds sidestep the landmark recognition problem. However, the large amount of basic features needed for mapping again limits scalability and threatens to cause a substantial performance loss on the robotic system.

The performance issues that plague large metrical maps are no problem for topological representations. The ability to set up map nodes at arbitrary positions can be used to focus mapping on places that require navigational decisions, so that map complexity scales with the navigational complexity of the environment instead of its metrical size. The critical issue of topological place recognition is simplified substantially for the target outdoor scenario, thanks to the possibility of using absolute position sensors such as GPS. If the map nodes are spaced sufficiently far apart from each other, this information alone is enough to unambiguously identify the nearest topological node.

However, the topological navigation scheme also has drawbacks. For one, it is too abstract to directly support important local tasks like obstacle avoidance or calculating a fine-grained trajectory. This may very well explain the apparent lack of published topological navigation approaches for outdoor scenarios. Luckily, it is not an overly serious drawback for the two-layered architecture considered in the scope of this thesis. With this partioning of concerns, local navigation is within the domain of the behavior-based piloting layer, which deals with trajectory planning and obstacle avoidance in a mostly sensor driven, behavior-based fashion.

A more serious problem arises for the task of exploration. In order to select promising directions in which to extend the current topological map, the mapping component must have access to traversability estimates for terrain that is not yet covered by a topological

node or edge. In contrast to dense structures such as elevation maps, which can be easily queried about new trajectories, a purely topological map does not provide *any* information outside the realm of its constituent connections. Thus, traversability-driven exploration is a difficult problem when using a topological map.

One option to approach this dilemma is presented by the hierarchical hybrid maps covered at the end of the survey section. By instantiating a local metrical map containing traversability information at each topological node, it becomes possible to estimate the quality of new connections in the vicinity of existing nodes, without simultaneously becoming dependant on accurate global localization as systems handling one large global metrical map.

Table 3.2 summarizes the discussed criteria relevant for the selection of a suitable map representation in the scope of this thesis. As can be seen, the topological map types seem to be a better overall choice than the metrical and feature-based map type. Due to the improved support of explorative operations, the hierarchical hybrid map type thereby holds a slight advantage over the purely topological map.

Capability	Map Suitability					
	Grid	Basic Feature	Landmark Feature	Topological	Abstracted Hybrid	Hierarchical Hybrid
Supports path planning	+	-	-	+	+	+
Supports large scale mapping	-	-	+	+	-	+
Tolerant to large env. variations	+	+	-	+	[1]	[1]
Can incorporate 3D information	-	+	+	+	[1]	[1]
Tolerant to inaccurate localization	-	-	+	+	-	+
Ability to support exploration	+	-	-	-	-	+

[1] Depends on underlying metrical map structure

Table 3.2: Suitability of different map types for the off-road application scenario

3.3 A Hybrid Map for Off-Road Navigation

The last section has presented an overview of the different mapping paradigms and discussed their principal suitability for the task at hand. Based on this analysis, the hierarchical metric-topological map type appears to provide the best basis for successfully extending the navigational capabilities of a behavior-based off-road vehicle.

This concept is now developed further and expanded into a formal definition of the hierarchical hybrid map that will be used thorough this thesis. Initially, the definition will be restricted to the high-level topological component, as this is sufficient for the main focus of this chapter, i.e. navigation within a given map. The map model will be augmented with local metrical maps for exploration tasks in the next chapter.

The following design decisions have been made during the specification of the topological map component:

1. The absolute metrical node position shall be used for place recognition.
2. All stored map data shall retain full three dimensional information.

3. Map edges shall be directed.

4. Nodes and edges shall be typed.

Item 1 allows the abstraction from self-localization issues that are not focus of this thesis. Instead, it is assumed that navigation-relevant topological nodes are spaced sufficiently far apart so that they can be distinguished unambiguously by the global position attribute available through the onboard GPS system.

Item 2 is a necessary precondition for modeling complex environments containing traversable routes on multiple levels, such as territory with bridges or tunnels. Such elements are not uncommon in surveillance or predefined route following scenarios, and therefore the map should be designed to support three dimensional environments.

The choice of directed map edges in item 3 is very important, because it allows the representation of asymmetrical traversal costs between two nodes, depending on traversal direction. Such asymmetries are widespread for outdoor areas. For example, one can consider sloped terrain, where downhill travel is energetically much cheaper than driving in the opposite direction. Additionally, the traversability of many naturally occurring obstacles such as ramps, steps or underbrush depends on the direction of an attempted traversal.

Finally, the design decision in item 4 has been made so that it becomes possible to explicitly distinguish between normal, hypothetical and untraversable elements. This eliminates the necessity to encode these situation with unpleasant helper constructions, as for example modeling blocked cells in digital elevation maps by inserting an unrealistic, 'magic' height value.

3.3.1 Map Formalization

The design decisions presented in the last section provide a solid foundation for the formal definition of the topological map, its constituent map nodes and edges and related matter such as paths.

The global topological map M is modeled as a simple container according to definition 3.3.

Definition 3.3 (Topological Map) *A **topological map** M is defined as a pair $M = (N, E)$, where*

- *N is a set of* map nodes *with $N = \{n_1, n_2, \ldots\}$*
- *E is a set of* map edges *with $E = \{e_1, e_2, \ldots\}$*

The map nodes n_i specify world locations which are relevant for robot navigation. In general, these nodes signify locations where the robot's navigational behavior can change. This encompasses junctions, at which the robot can update the direction it is currently approaching, but nodes might also be placed where the obstacle configuration changes significantly and thus adaption of the obstacle avoidance strategies is required. Nodes consist of a position and a type attribute as follows.

Definition 3.4 (Map Node) *A **map node** $n \in N$ is defined as an attribute pair $n = (\vec{p}, t)$, where*

- *$\vec{p}$ specifies the three-dimensional node* position *using the earth centered earth fixed coordinate system ECS*
- *t specifies the node* type *with $t \in \{\texttt{speculative}, \texttt{accessible}\}$.*

The semantical meaning of the two different node type is as follows: Speculative nodes have not yet been approached and therefore it is not necessarily clear whether their position can actually be reached at all. Instead, the speculative node might lie inside an impassable obstacle which has been overlooked when the node was initially placed. In contrast to this, nodes with accessible type have been visited at least once, and thus, their position $\vec{p}$ can indeed be reached.

Coming to the definition of the map edges, each edge e_i expresses a belief or accumulated evidence about the connectivity between two map nodes. As motivated previously, map edges in this thesis are *directed* and self-loops (edges with the same start and end node) are explicitly forbidden as they carry no navigation-relevant information. Also, edges contain a type attribute analogous to nodes, and a set of annotations, which will be used to store cost information later.

Definition 3.5 (Map Edge) *A **map edge** $e \in E$ is a tuple $e = ((n, n'), t, \Lambda)$, where*

- *(n, n') is an ordered pair of map nodes with $n, n' \in N \wedge n \neq n'$.*
- *t specifies the edge* type *with $t \in \{\texttt{speculative}, \texttt{traversable}, \texttt{untraversable}\}$*
- *Λ is a set of* annotations *containing travel cost information. The annotation set will be defined further in section 3.5.5.*

Two notational conventions are introduced to simplify the following exposition. For one, n and n' are subsequently termed the edge's **start** and **end node** and are addressed by $e.n$ and $e.n'$. Furthermore, the expression $(n, n') \in E$ will be used with the meaning of $\exists_{t,\Lambda}.((n, n'), t, \Lambda) \in E$.

The semantical meanings of the three different edge types are similar to those of the node types. A speculative edge has not yet been traversed and it is thus not known with certainty whether the edge is indeed traversable, or how costly such a traversal is. Traversable edges, on the other hand, have been successfully traversed at least once and are thus known to allow travel from the start to the end node. Not surprisingly, the untraversable type signals that the edge cannot be successfully traversed and thus should not be considered as a valid alternative for further exploration or path planning steps.

The annotation set Λ is the final constituent of an edge and is related to the cost measure required for path planning. The description of the actual contents and layout of this set is not directly relevant for the formal definition of the edges and is thus postponed until section 3.5.5.

Agreeing with general convention, a *path* in the map M is defined as a list of consecutive map edges:

Definition 3.6 (Path) *A path P is defined as a list of map edges $(e_0, e_1, \ldots, e_j)$ for which the following conditions hold:*

$$e \in P \;\Rightarrow\; e \in E \tag{3.3}$$
$$0 \leq i < j \;\Rightarrow\; e_i.n' = e_{i+1}.n \tag{3.4}$$

3.3.2 Map Visualization Conventions

In the remainder of the thesis, example images generated by the visualization tool described in appendix A will be used frequently to supplement the textual explanation of map related topics. This section introduces some conventions which are valid for all of these images.

Figure 3.11 shows the representation of map nodes as spheres placed at their metrical position $\vec{p}$. Map edges are visualized by straight arrows running from their respective start to their end nodes. Edges running in different directions are offset from each other to increase viewability, but this does not hold any navigational significance. Optionally, both nodes and edges can be superscribed by a textual name used for reference in the image description text. The primary colors red, green and blue are used to convey *type information*. The color coding scheme is listed in table 3.3.

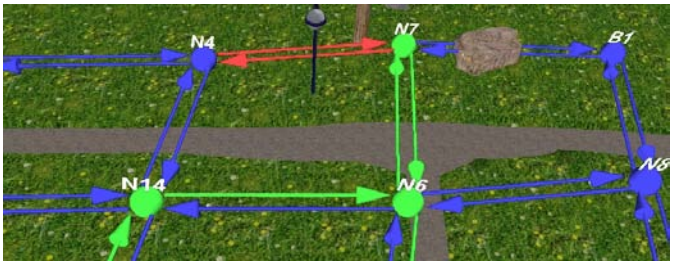


Figure 3.11:
Example of topological map visualization

Color	Node Type	Edge Type
Red	(unused)	untraversable
Green	accessible	traversable
Blue	speculative	speculative

Table 3.3:
Color scheme for map visualization

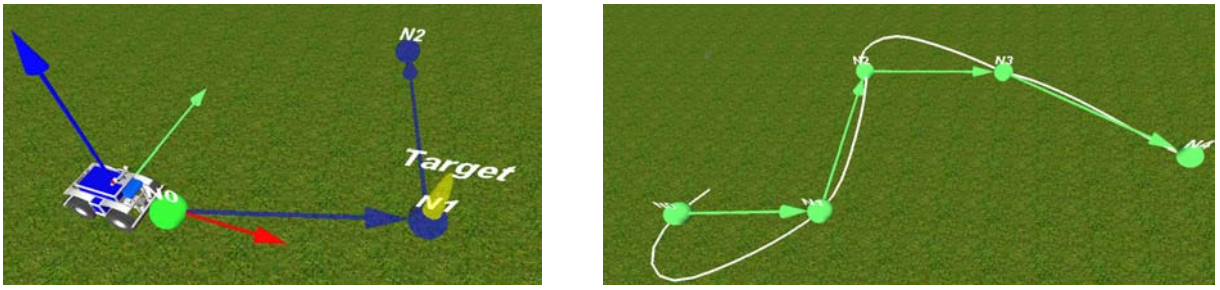
3.4 Basic Map Functionality

After introducing the topological map, the exposition continues with the basic functionality required to employ the map for navigation. This section describes how a path $P = (e_0, e_1, \ldots, e_j)$ (as generated by path planning described later in section 3.6) can be translated into appropriate motion commands for the pilot layer. Also, the detection of successful and unsuccessful traversal attempts is explained in detail, along with a description of the corresponding map updates prompted by these events.

It should be noted that the design of these functionalities is not as trivial as it might seem at first glance. In order to function reliably, special care has to be taken to make these operations both robust against inaccurate position information and tolerant against the pilot's black-box behavior when it responds to a motion command by the navigator.

3.4.1 Generation of Motion Commands

Each path traversal task that is to be carried out by the robot is a chain of subsequent edge traversal tasks. As the metrical positions of all map nodes are known, the simplest way to communicate these edge traversal tasks to the robot's pilot over the available interfaces (see sec. 2.2.2) is to take the position $\vec{p}$ of the current edge's end node $e_i.n'$ and to activate the Approach Target Position behavior of the pilot (figure 3.12a). Once the robot has arrived at that position, the position of the next edge's end node $e_{i+1}.n'$ can be set as target. Of course, the ECS node position must be converted into the pilot's WCS system prior to sending the coordinates by applying the pose transformation matrix M_{ECS}^{WCS}.



(a) Single motion command (b) Resulting trajectory

Figure 3.12: Path traversal using the Approach Target Position behavior
(a) Target position is set to current edge's end node position.
(b) Steep turns lead to large steering corrections.

Unfortunately, implementing path traversal by simply approaching a sequence of end node positions causes the robot to perform extensive turning maneuvers every time it reaches an intermediate node and receives a new goal lying in another direction. This behavior can be observed well in figure 3.12b, which has been produced by tracing the robots trajectory while it traveled along the nodes N_0, N_1, N_2, N_3, N_4.

The reason for these large deviations is obviously the fact that the pilot approaches the target position using the most direct route and does not take the heading toward the next goal into account until it has finished approaching the current one. This is especially problematic as the robot is distinctly non-holonomic.

In order to reduce the swaying, it is necessary to let the robot orient itself earlier toward the goal that will become active after the current one. A suitable interface to communicate such requests is the Approach Target Pose behavior, which causes the pilot to approach a certain WCS position while simultaneously trying to attain a given WCS yaw angle. As the navigator works in the ECS coordinate system, a suitable three-dimensional ECS target pose needs to be constructed, transformed using the conversion matrix M_{ECS}^{WCS} and then supplied to the pilot. The ECS target pose can be constructed using the position of the current target node $e_i.n'.\vec{p}$ and the heading $\vec{h}$ from the current to the next target node given by

$$\vec{h} = e_{i+1}.n'.\vec{p} - e_i.n'.\vec{p} \tag{3.5}$$

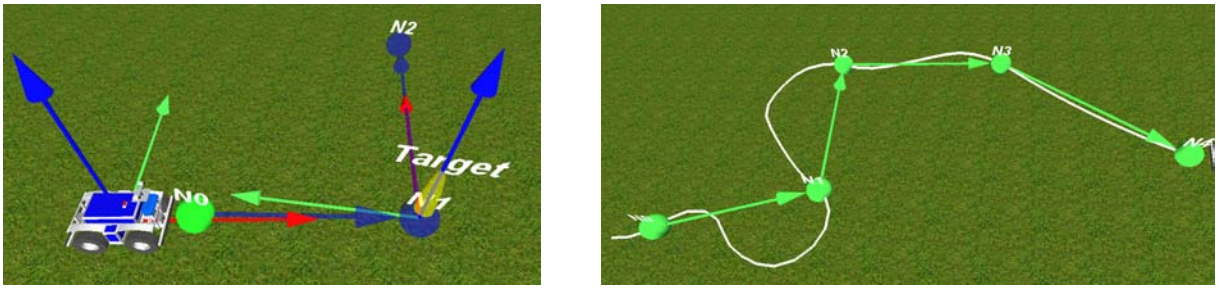
To complete the specification of the target pose, an additional constraint on the z-axis of the target pose is needed. After recalling that the WCS yaw-angle is interpreted as

a rotation around the WCS z-axis, one can argue that the selection of the WCS z-axis image in ECS coordinates

$$\vec{\Psi}_z = (M_{ECS}^{WCS}(0,2), M_{ECS}^{WCS}(1,2), M_{ECS}^{WCS}(2,2)) \tag{3.6}$$

as the ECS target pose z-axis results in WCS roll and pitch angles of 0 after coordinate-system transformation and thus minimizes distortion of the yaw-angle.

With these specification, the full three-dimensional ECS target pose can be computed using the SetupCSWithZAxis function introduced in 2.3.3 by invoking SetupCSWithZAxis($e_i.n'.\vec{p}$, $\vec{h}$, $\vec{\Psi}_z$). After converting the resulting matrix into a WCS pose, it can be used as input for the Approach Target Pose behavior for all edge traversals except the last one. As the last edge traversal command does not have a successor node required to compute $\vec{h}$, it must be conveyed to the Approach Target Position behavior instead.



(a) Single motion command (b) Resulting trajectory

Figure 3.13: Path traversal using the Approach Target Pose behavior
(a) Target pose orientation points directly toward next goal.
(b) Large steering corrections now occur mainly *before* steep turns.

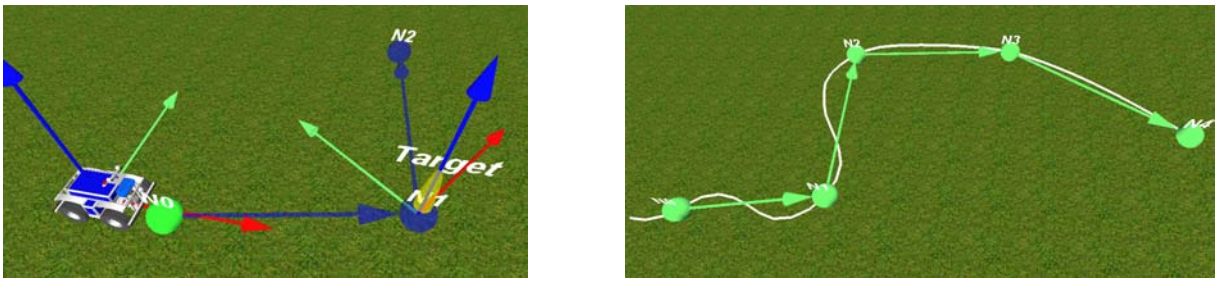
Figure 3.13 shows the effects of the updated path traversal technique. Unfortunately, it can still be observed that the pilot deviates widely from what a human would consider an 'optimal' trajectory. It appears that by specifying the target pose with an orientation pointing directly toward the next goal, the pilot is forced to perform the steering corrections during the approach to the current goal too early and too aggressively.

Thus, a third variant to translate path traversal tasks into motion commands has been developed. This time, the target pose orientation is calculated as the *angle bisector* between the direction to the current goal node and the direction to the next node. Mathematically, this bisection vector can be computed easily using the formula

$$\vec{h'} = \frac{(e_{i+1}.n'.\vec{p} - e_i.n'.\vec{p}) + (e_i.n'.\vec{p} - e_{i-1}.n'.\vec{p})}{2} \tag{3.7}$$

$$= \frac{(e_{i+1}.n'.\vec{p} - e_{i-1}.n'.\vec{p})}{2} \tag{3.8}$$

An example is given in figure 3.14 to illustrate the effects of the proposed computation method for the target pose. It can be seen that the use of the angle bisector as intermediate target orientation spreads the necessary steering corrections relatively uniformly over the whole path that is to be traversed. Thus, this technique has been adopted as the most suitable approach for translating path traversal tasks into motion commands for the piloting sublayer.



(a) Single motion command (b) Resulting trajectory

Figure 3.14: Path traversal using the improved Approach Target Pose behavior
(a) Target pose orientation is set half way between direction to current and direction to next goal.
(b) Large steering corrections are eliminated.

Formulation of this strategy in form of pseudo-code leads to algorithm 3. With the use of the ApproachTargetECSPose method, the traversal of path P can be effected by calling a sequence of ApproachTargetECSPose($e_i.n.\vec{p}$, $e_i.n'.\vec{p}$, $e_{i+1}.n'.\vec{p}$) functions for all valid i of P (the exception necessary for the last edge has already been discussed).

Algorithm 3: ApproachTargetECSPose(Point3D start_pos, Point3D target_pos, Point3D next_pos)

Data: Point3D start_pos, Point3D target_pos, Point3D next_pos
Result: Let the robot approach the ECS position target_pos starting from ECS position start_pos and obtaining a final orientation that is suitable for continuing toward the position next_pos
Vector3D h = $\frac{\text{next\_pos}-\text{start\_pos}}{2}$
Θ = SetupCSWithZAxis(target_pos, h, $\vec{\Psi}_z$)
ApproachWCSPose ($M_{ECS}^{WCS} * \Theta$)

3.4.2 Arrival Detection

The path traversal scheme presented in the last section must be able to detect the arrival at the currently active target node in order to proceed to the next edge traversal subtask. As the stored information usable for node recognition is limited to the global position attribute of each node, the issue of *arrival detection* must rely exclusively on this data. The task is complicated by two factors:

1. the robot's positional information may drift substantially during target approach.
2. the behavior-based pilot may choose to approach the target inexactly, f.e. due to nearby obstacles or steering restrictions.

These facts prevent the implementation of a node arrival detection algorithm which simply waits until the difference between robot position and target position becomes zero. Instead, some finite sized *catchment area* must be defined around the target with a size that compensates both localization drift and steering inaccuracies. As the exact values of

these two quantities are often unknown, robustness demands to estimate their magnitude pessimistically, easily leading to catchment areas of 5 meters or more.

One method to reduce the area's size presents itself after analyzing the drift characteristics of the GPS localization subsystem. As can be seen in table 3.4, the large variance of the GPS elevation value (parallel to the WCS Z-axis by construction) contributes substantially to the total localization drift.

Axis	Std. Deviation [m]
WCS X	2.75
WCS Y	1.89
WCS Z	5.63

Table 3.4:
Standard deviations of GPS positions in direction of the three WCS coordinate axes
Mean and standard deviation has been estimated from ≈ 100000 measurements taken over a period of 28 hours using a stationary Holux GPSlim 236 GPS device [Schmitz 05].

Thus, it is possible to blank out the adverse effect of drifting elevation measurements by using a distance metric that excludes the robot's WCS Z position. An obvious approach is to project both robot and target position into the WCS X-Y plane and compute the two-dimensional, euclidean distance. In the navigator's ECS system, this is equivalent to calculating the shortest distance between the ECS robot position and a line running through the target position with direction of the projected WCS Z axis $\vec{\Psi}_z$. The mathematical solution to this problem is easily obtained using the cross-product (figure 3.15).

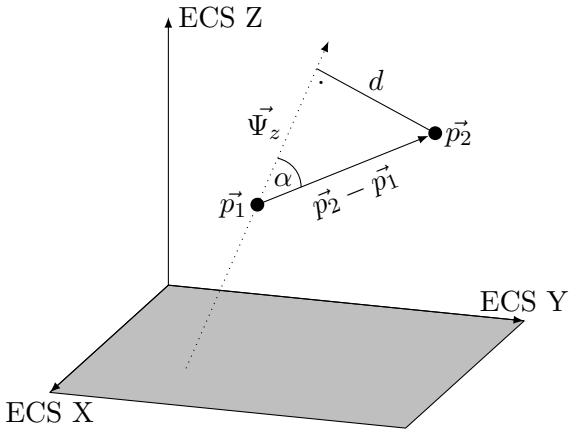


Figure 3.15: A WCS Z-coordinate invariant distance measure in ECS
The distance d between two ECS-points $\vec{p_1}$, $\vec{p_2}$ after projecting them along the WCS Z-Axis $\vec{\Psi}_z$ can be calculated by observing that $|\vec{\Psi}_z \times (\vec{p_2} - \vec{p_1})| = |\vec{\Psi}_z||\vec{p_2} - \vec{p_1}| \sin\alpha$

The two equations

$$|\vec{\Psi}_z \times (\vec{p_2} - \vec{p_1})| = |\vec{\Psi}_z||\vec{p_2} - \vec{p_1}| \sin\alpha \tag{3.9}$$

$$\sin\alpha = \frac{d}{|\vec{p_2} - \vec{p_1}|} \tag{3.10}$$

lead to definition 3.7 of the *projected distance* between two points.

Definition 3.7 *The* projected distance $d(\vec{p_1}, \vec{p_2})$ *between two points* $\vec{p_1}$, $\vec{p_2}$ *specified in ECS coordinates is given by*

$$d(\vec{p_1}, \vec{p_2}) = \vec{\Psi}_z \times (\vec{p_2} - \vec{p_1}) \tag{3.11}$$

In the following, the term **target distance** is used to denote the *projected* distance between robot and a target ECS position.

It is easy to implement catchment area based arrival detection by monitoring the target distance and signaling the arrival event once the distance falls below the catchment area's size $d_{arrival}$. However, this intuitive approach is suboptimal in the sense that it does not profit from situations where accurate localization information is available and the pilot could approach the given target position much more precisely than up to the distance s. A solution to this problem is to postpone sending the 'arrival detected' signal while the target distance is less or equal to s and continues to shrink. Experimental validation has shown that it is sensible to terminate the event postponement once the target distance has not decreased further below the previous minimal distance for a short slack time period. This slack time window with typical values of $300 - 600$ ms is useful to make the target arrival detection more tolerant against small correction maneuvers or localization glitches, which can increase the target distance minutely during the approach movement. Listing 4 shows a pseudo-code implementation of this algorithm for target arrival detection with postponement.

Algorithm 4: DetectTargetArrivalWithPostponement

```
Data: Point3D target, Point3D robot, float d_arrival, int slack_time
Result: Terminates when robot has detected maximal proximity to target
bool arrived = false;
while forever do
    if d(target, robot) ≤ d_arrival and not arrived then
        arrived = true;
        min_dist = d(target, robot);
        time = current_system_time;
    end
    if arrived then
        if d(target, robot) < min_dist then
            min_dist = d(target, robot);
            time = current_system_time;
        else if current_system_time - time > slack_time then
            return true;
        end
    end
end
```

It is important to note that this algorithm guarantees that the time between entering the catchment area (`arrived = true`) and triggering the target arrival event (`return true`) is finite, given the reasonable assumption that the robots velocity is either 0 or always

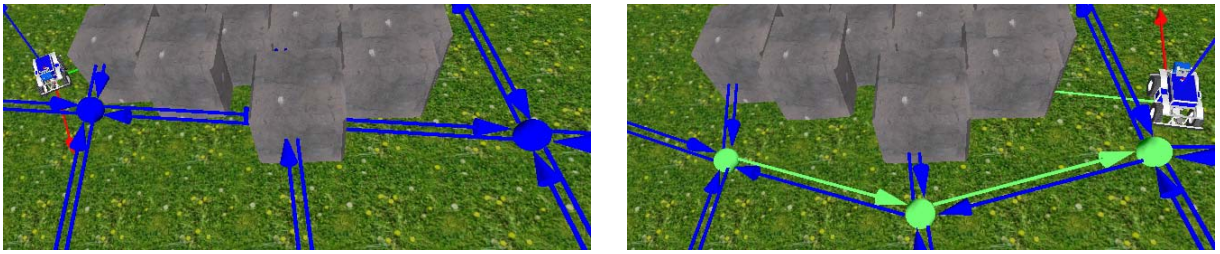
remains larger than some minimal value $\epsilon > 0$. Under these conditions, `min_distance` can only decrease finitely often (lines 9-11). After the last decrease, the variable `time` is not updated any more and thus the condition at line 13 is fulfilled with `slack_time` delay.

In conclusion, robustness to drifting localization and inaccurate steering is achieved using an elevation independent distance metric and a catchment area around the topological node's position. However, if localization accuracy permits, the robot is still able to converge to the precise location of the topological node despite the catchment area because the arrival event is postponed while the robot continues to approach the topological node.

3.4.3 Relocalization of Speculative Nodes

Whenever the robot detects the arrival at a speculative topological node, its type attribute is altered to accessible in order to record this information and remain consistent with the semantics given in definition 3.4. However, experiments have shown that this is also a good time to correct the speculative node's position, if it has initially been placed too close to (previously unknown) obstacles. In such a case, the best effort characteristics of the pilot layer and the target arrival detection algorithm will have offset the robot from the original position of the badly placed node toward a better reachable position within the target catchment area. Thus, the position of the robot upon signaling the target arrival event can be used to *relocalize* the speculative node.

An example of the effects obtained by this technique is depicted in figure 3.16. As can be seen, the middle node was initially placed inside an obstacle, but has been successfully 'pulled out' by node relocalization. Overall, validation in both simulation and real world experiments has shown that the proposed relocalization step substantially increases the robustness of the whole navigation scheme against badly placed topological nodes.



(a) Before path traversal (b) After path traversal

Figure 3.16: Relocalization of speculative nodes

3.4.4 Fusion of Relocalized Nodes

Even under the assumption that all nodes in the initial map are well separated from each other, relocalization of a speculative node may result in a situation where two or more nodes are placed closely together. This is problematic due to several reasons. For one, it becomes impossible to decide solely based on the current robot pose at which node the robot is located once their spatial locations are within the inaccuracy range of the robot's position estimate. For another, the amount of turning required for a large non-holonomic robot makes the traversal of very short edges difficult. Thirdly, as each

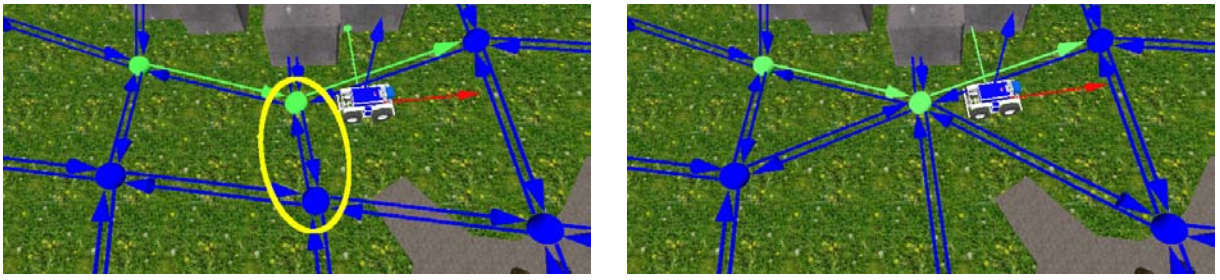
topological node is an abstracted representation of the environment surrounding it, closely spaced nodes contain rather redundant information that does not contribute much for the task of efficient large scale navigation. Because of these detrimental effects, the quality of the navigator's topological map can be improved by joining nodes that come to lie too close together. This can be accomplished using a **node fusion** algorithm.

To counteract the creation of closely packed nodes after node relocalization, it is necessary to check for such an occurrence every time a speculative node t has been moved as proposed in section 3.4.3. In the general case, this proximity check can produce a *set* of nodes M whose projected distances to t fall below a given minimal distance threshold. If M contains at least one accessible node, the former speculative node has evidently moved into 'known terrain' already represented by other, established nodes. Thus, a viable strategy to eliminate the agglomeration of nodes in this case is to select the accessible node c which is *closest* to t and fuse t with it. The fusion algorithm's task is then to reconnect all incoming and outgoing edges of t to c, ensuring that no duplicated edges or self-loops are created, and finally to remove t from the topological map.

A different situation occurs when the set M contains *no* accessible nodes. In this case, the freshly relocalized node has been placed at an accessible location in previously unknown terrain. Therefore, it should be preserved, as it represents more definite information as other, still speculative nodes. Consequently, the fusion algorithm should now be invoked with t as the node to fuse with and the set M as the set of nodes that are to be fused.

Algorithm 5 shows a pseudo-code implementation of the fusion algorithm that has been developed in order to eliminate clustered nodes after node relocalization. As described, the actual parameters given to the algorithm depend on the type of nodes in the set M. In case an accessible node is found near the freshly relocalized node t, the procedure is invoked by calling FuseNodes(c, $\{t\}$). If only speculative nodes are close, the call FuseNodes(t, M) is used instead.

An example application of the fusion algorithm is shown in figure 3.17. Here, the scene from the node relocalization example is embedded into a larger context. As can be seen, a second row of speculative nodes is located close to the three nodes from the previous example. Because the relocalized node comes to lie closer to the speculative node in the middle than the minimal threshold, the FuseNodes algorithm is invoked and fuses the speculative node with the freshly relocalized one.



(a) Before node fusion (b) After node fusion

Figure 3.17: Node fusion
(a) Node relocalization has moved a node too close to another (marked yellow). (b) The accessible node assimilates the nearby speculative node and its edge connections.

Algorithm 5: FuseNodes(Node t, Nodeset M)

Data: Graph $G = (N, E)$ // *the global topological map*
Data: EdgeSet C // *edges to be reconnected with* t
Data: EdgeSet R // *edges to be removed from* E

// mark incoming edges of M *either for reconnection or removal*;
foreach $(n, n') \in E$ *with* $n' \in M$ **do**
 | **if** $(n, t) \in E$ **or** $n = t$ **then** $R \leftarrow R \cup (n, n')$ **else** $C \leftarrow C \cup (n, n')$;
end
// reconnect unique edges and remove duplicates;
foreach $(n, n') \in C$ **do**
 | **if** $(n, t) \in E$ **then** $R \leftarrow R \cup (n, n')$ **else** $E \leftarrow E \setminus (n, n')$; $E \leftarrow E \cup (n, t)$;
end
$E \leftarrow E \setminus R$; $R \leftarrow \emptyset$; $C \leftarrow \emptyset$;

// mark outgoing edges of M *either for reconnection or removal*;
foreach $(n, n') \in E$ *with* $n \in M$ **do**
 | **if** $n' = t$ **then** $R \leftarrow R \cup (n, n')$ **else** $C \leftarrow C \cup (n, n')$;
end
// reconnect unique edges and remove duplicates;
foreach $(n, n') \in C$ **do**
 | **if** $(t, n') \in E$ **then** $R \leftarrow R \cup (n, n')$ **else** $E \leftarrow E \setminus (n, n')$; $E \leftarrow E \cup (t, n')$;
end
$E \leftarrow E \setminus R$; $N \leftarrow N \setminus M$;

3.4.5 Detection of Navigation Failures

The pilot layer cannot guarantee to actually find a traversable path to the target position or pose requested by the navigator, even if one exists. The possible causes for such *traversal failures* are manifold, but most can be traced back either to the limited local world model used by the pilot or to the incomplete or misinterpreted sensor data available. In one of the most common scenarios, the robot becomes trapped in a dead end extending beyond the scope of the pilot's world model. Thus, even if the pilot detects the dead end after moving inside, the local memory is insufficient to contain this information long enough to allow the robot to turn around and leave the dead end completely. Instead, it is quite possible that the vehicle turns back again and consequently starts to oscillate, or becomes stuck outright. Other cases of navigation failures are caused by changes in terrain perception. Due to the uncertainty inherent in sensor data processing, a terrain patch that appeared to be traversable from afar can easily be reevaluated as impassable upon approach, causing the pilot to change course and possibly stray away from the goal completely.

Possible traversal failures can be categorized into two main classes:

1. Deadlocks: The pilot cannot move toward the target position at all or only with an unacceptably slow speed.

2. Livelocks: The pilot can still move swiftly toward the target, but is also forced to drive away from it again at some point, so that the robot starts to oscillate and never arrives.

In order to build a robust navigation system capable of recovering from these unavoidable failures, a **failure detection** strategy must be developed that can detect both dead- and livelock failure types. After detection, the information must be entered into the topological map to allow subsequent navigation commands to avoid the action that led to the failure.

The reliable detection of traversal failures is complicated by the design constraint to treat the pilot layer as a 'black box' whose internal workings cannot be influenced in detail. With this restriction, one must resort to examining the effects produced by the pilot, i.e. the positional change of the robot relative to the target. A straightforward method to implement failure detection within these limits is to define a *timeout* for each issued edge traversal command and assume that the target can not be reached if the allotted time expires before a successful target arrival is detected.

However, the selection of a generally applicable, absolute timeout value is problematic, as the time needed to travel to a given position depends on many factors such as the target distance, the maximal possible vehicle speed and the terrain difficulty on the way. Ignoring the influence of terrain complexity for now, both distance and maximum speed can be used to adapt the timeout value and create a more flexible failure detection rule. Unfortunately, this rule still exhibits the unpleasant characteristics that a failure event can be produced even while the robot is driving directly toward the target, in case it has lost too much time negotiating an obstacle situation earlier on.

It is possible to eliminate this behavior by dropping the idea of specifying a general timeout and instead limiting only the time during which the robot is *not approaching the goal* swiftly enough. This line of thought leads to the formulation of the actual failure detection constraint used in this thesis. Its central idea is to demand a target approach with a minimal speed indirectly by setting a lower limit on the decrease in target distance between two subsequent timesteps t_0 and t_1. For this, the maximal possible decrease in target distance Δd_{max} is calculated first using the maximum possible vehicle speed v_{max} and the available time interval $\Delta t = t_1 - t_0$:

$$\Delta d_{max} = v_{max} \Delta t \tag{3.12}$$

A *desired target distance* for time t_1 is then computed from the target distance between the robot position $\vec{r}$ and the goal position $\vec{p}$ at time t_0 by subtracting a fraction κ of Δd_{max}:

$$d_{desired} = \max\left(d(\vec{r}(t_0), \vec{p}) - \kappa \Delta d_{max}, 0\right) \tag{3.13}$$

Now, for each discrete time step t_1 that the navigator awaits the pilot's arrival at the target position, the current target distance $d(\vec{r}(t_1), \vec{p})$ is compared with the desired target distance $d_{desired}$ extrapolated from the previous time step t_0. Each time $d_{desired}$ lies below $d(\vec{r}(t_1), \vec{p})$, a failure counter t_{fail} is incremented:

$$t_{fail} \leftarrow t_{fail} + \Delta t \text{ iff } d_{desired} < d(\vec{r}(t_1), \vec{p}) \tag{3.14}$$

Based on t_{fail}, a failed target approach is detected once its value rises above a given delay time threshold T_{fail}, which determines the time one wants to allow for making detours

during target approach. Suitable values for T_{fail} must be determined empirically, but can be guided by considering the expected amount of clutter in the operation environment and the maximal size of detours that the pilot's world model can generate. The choice of κ is less critical. In practice, a value of 0.25 has proven to be useful. Thus, the pilot is considered to be off track if it does not approach the target with at least a quarter of the current maximum velocity.

Once a traversal failure is detected while the pilot is executing an edge traversal command, the navigator layer must become active and perform two tasks. First, the failure must be represented in the topological map in a way that prevents the repetition of the unsuccessful attempt in the future. Second, a new command must be issued in order to let the pilot resume movement. The second task falls into the responsibility of the navigator's path planning and exploration components and will be discussed in their respective expositions in sections 3.6 and 4.2. Here, the presentation will continue with the issue of **failure representation**.

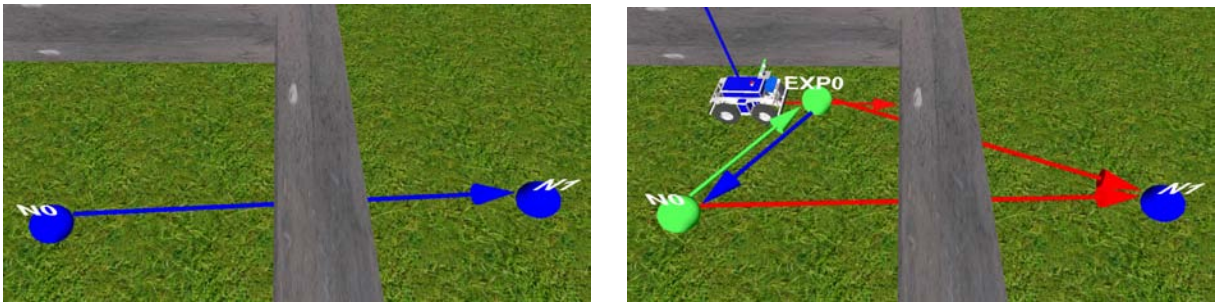
An obvious step that needs to be taken when the pilot fails to traverse an edge e is to change e's type attribute to untraversable in order to exclude it from further use in path planning. However, this does not capture all information that has been gained during the failed attempt. Some of this information is also contained in the position occupied by the robot at the instance of the failure attempt. This location is obviously accessible, but it is also navigation-relevant in the sense that it is probably as close to the original goal as the piloting layer could manage. Therefore, it appears sensible to insert an accessible node exp at the robot's position, along with a traversable edge originating from $e.n$ and an untraversable edge to $e.n'$. Finally, it is reasonable to assume that the pilot can steer back from exp to e, so at least the insertion of a speculative edge $(\texttt{EXP}, e.n)$ is warranted. The complete update rule is formulated in algorithm 6 and an example is displayed in figure 3.18.

Algorithm 6: RepresentFailure(Edge e, Point3D *robot*)

Data: Graph G = (N, E) *// the global topological map*
$((n, n'), t, A) := e$;
$exp := (robot, \text{accessible})$;
$N \leftarrow N \cup exp$;
$E \leftarrow E \setminus e$;
$E \leftarrow E \cup ((n, n'), \text{untraversable}, A)$;
$E \leftarrow E \cup ((e.n, exp), \text{traversable}, \emptyset)$;
$E \leftarrow E \cup ((exp, e.n), \text{speculative}, \emptyset)$;
$E \leftarrow E \cup ((exp, e.n'), \text{untraversable}, \emptyset)$;

3.5 Cost Measure Definition

The exposition of basic functionalities in the last section has shown how a given path traversal command can be translated into motion requests for the robot pilot and how both successful and unsuccessful command execution can be detected robustly. Now, it is possible to approach the question how the navigator can actually *find* the most suitable path through the topological map to a given goal in the first place.



(a) Map before traversal failure (b) Map after traversal failure

Figure 3.18: Representing traversal failures

The most important step required in order to solve this task of topological path planning is to cast the informal notion of the 'most suitable path' into a mathematical formalism that can be treated with the help of a path planning algorithm. This can be accomplished by adding a **cost measure** to each graph edge and defining the total cost of a candidate path as the sum of its constituent edge costs. Finding the 'most suitable' path then amounts to a search for the path from start to end node which minimizes the path cost.

Since the cost measure is key to the whole process of path planning, its definition needs to be approached carefully. A good choice should fulfill at least the following three requirements: For one, the cost measure should be *complete*, e.g. capture all of the environmental aspects that have a substantial impact on the robot's performance. For another, as the importance of the different aspects might change due to new mission requirements, the cost measure should be *adaptable* based on externally selectable priorities. And thirdly, it should be ensured that edge costs always remain *consistent* with the actual difficulties experienced upon map edge traversal, even if these were mis-estimated initially or change over time.

The next sections give a short overview of existing cost metrics and discuss their suitability with respect to the given set of desired characteristics. Afterward, the multi-criterion cost model that has been developed in the context of this thesis is introduced. Subsequently, the mechanisms which allow the adaption of the cost measure to changing path planning requirements and a learning scheme that ensures cost consistency is presented. The chapter concludes with a description of the path planning process that is based on the previously introduced cost function. A series of both simulated and real experiments are conducted in order to evaluate the achieved results.

3.5.1 Existing Measures

Literature provides ample examples of cost measures for robot navigation. However, most proposals deal with dense metrical maps and are tuned toward sensor-based traversability analysis. For topological approaches, issues such as place recognition and perceptual aliasing have dominated research and cost measures propositions are considerably rarer. Nevertheless, this section gives a survey over different classes of existing cost measures for topological maps.

Scalar Cost Measures

In the simplest case, the cost measure takes the form of a single scalar value and its natural order. In this case, classical dynamic programming approaches such as the dijkstra [Dijkstra 59], A* or D* algorithm can be used for path planning. These allow substantial combinatorial reductions in the search space of possible paths and have a typical complexity of $O(\|N^2\|)$ for graphs containing N nodes.

Topological path planning systems using scalar cost measures usually work with edge traversal times or metrical distance betweens start and end nodes. Examples using node distance as cost metric include the TOUR model from Kuipers [Kuipers 77] or the hybrid mapping approach from Thrun [Thrun 98a]. [Ryu 99] also employs metrical node distance to define a cost measure for topological path planning.

Stochastic Cost Measures

Dynamic environments that contain elements such as doors or mobile obstacles can exhibit large traversability variations over time. If the changes follow some consistent pattern (such as a door which is closed 30% of the time), they can be taken into account for improved robot path planning. One way to achieve this is to model the observed variations into the edges' cost measures, transforming them into **stochastical cost measures**. [Loui 83] summarizes some ways of dealing with such weights.

One approach is to calculate the expectation of each cost measure and solve the resulting deterministic problem using a technique for scalar measures. [Hu 97] presents a path planning algorithm for dynamic environments that employs such a strategy. Here, two probabilistic models are maintained for each edge in a predefined topological map: one model for the probability of encountering obstacles that partially obstruct a robots path and thus increase travel time linearly and another model for the likelihood of fully blocked edges which require expensive backtracking. By continuously adapting the parameters of both probability models whenever obstacles are detected using Bayes estimates, it is possible to compute an average 'uncertainty cost' from the models expectation values. This cost can then be added to the static scalar edge cost based on travel time. Further publications which employ stochastical edge cost models are [Simmons 97] or [Kruusmaa 03]. [Kruusmaa 03] actually represents the variance of edge costs in a 'case-base' for case based reasoning, which is a rather unconventional approach. However, the resulting behavior of the path planner is comparable to the other stochastic methods. As these techniques map the gathered probability information onto a single scalar cost value for each edge separately, path planning itself can again be accomplished using the dijkstra algorithm.

In contrast, the use of full probability distributions during path construction allows to optimize probabilistic conditions such as finding the path with 'the greatest probability of realizing the least weight'. Unfortunately, such approaches cannot be solved with dynamic programming and are thus computationally intractable for online robotic applications.

Multi-Criterion Cost Measures

In real applications, it is easy to come up with more than one aspect of 'cost' that should be optimized by a path planning strategy. For example, three important cost aspects of paths subject to minimization are a) path length, b) energy consumption and c) travel time.

While the required information can be easily stored by extending a scalar cost measure to a **multi-dimensional cost vector** that records separate values for each cost component, the definition of an ordering relation becomes *much* harder. For instance, is a path superior to another if it requires one unit of energy less but needs two seconds more? And how do different path lengths play into this?

Multi-criterion decisions similar to the one outlined above have been studied in the context of operations research and decision-making theory for a long time. Available surveys and theoretical works on this topic can be found in [Loui 83], [Horn 97] and [Zitzler 99]. For applications outside the area of robotics, a popular approach is to rule out all solutions that are definitely worse than another one *before* any multi criterion decisions are made [Horn 97]. This can be accomplished as soon as the domain of each single cost component is partially ordered, which can be expected for all realistic scenarios. Given such orderings, a solution is said to *dominate* another one if a) all of its components are equal or better that those of the dominated one and b) at least one component is definitely better. With this definition, it is clear that dominated solutions are not relevant for the selection of the final decision and can thus be discarded. Once the set of non-dominated solutions (also called the 'pareto set') is found, its cardinality might allow a human operator to simply select the solution that he or she deems best. In this case, automated decisions about the difficult multi-criterion problem are side-stepped completely.

Of course, human interaction is no option for an autonomously operating robot. Thus, some order on the cost measure vector must be imposed. The **scalar aggregation** method defines an *utility function* $u(\vec{c}) : \Re^k \rightarrow \Re$ that maps the k components of the cost measure vector $\vec{c}$ into a single utility value. Based on this value, simple scalar path planning can be performed. A straightforward choice for u is a linear combination of all cost aspects with user-defined weights w_i: $u(\vec{c}) = w_0c_0 + w_1c_1 + \ldots + w_kc_k$. The weights can be used to specify the relative importance of each cost factor, but the method cannot express more complicated orderings such as non-linear relations or dependencies between components (like in the statement 'less battery consumption is only better if travel time does not increase'). Nevertheless, scalar aggregation is often used in practice. [Soltani 02] presents an application for path planning in construction sites, where possible travel speed, obstacle clearance and visibility of the surrounding are modeled and aggregated into a single cost scalar.

Another possibility to define an order on the cost vector is to introduce a hierarchy similar to a lexicographic ordering. [Fernández-Madrigal 99] presents such a **hierarchical** multi-criterion approach for path planning. Here, a hierarchy of 'less than', 'equal to', or 'greater than' constraints is formulated on the cost factors. Each constraint imposes a partial ordering for the involved cost aspects *on that level*. Between levels, the satisfaction of a higher level constraint is infinitely more important that that of a lower level. This implies that the optimization of a path according to level 2 criterions can only work with the solutions that fully satisfy all level 1 criterions. In practice, this allows the formulation of a multicriterion order without the need to mix or compare different cost factors by defining constraints for different cost aspects on distinct levels. For example, a sensible ordering of a cost vector with size 3 could be introduced by the following three constraints:

- Level 1: safety $(c_0) > 0.9$
- Level 2: distance $(c_1) < 100$m

- Level 3: time (c_2) < 120s

In this example, the proposed METAL-A* search algorithm would first find paths with a safety value above 0.9, then try to minimize distance among the remaining candidates and finally consider the time cost factor among the selected solutions. If one constraint cannot be successfully satisfied, the implied order is used to provide a best effort approximative solution. In the example, if there are no safe paths with distances < 100m, the shortest path that still holds $c_0 > 0.9$ would be selected.

3.5.2 Cost Measure Selection

The presented approaches cover a wide spectrum of possible ideas for cost measures. However, not all of them are equally well suited for the use in the outdoor scenario which is the target here. For instance, single valued cost metrics are too restricted to record all information about important cost factors in outdoor environments. As an example, taking the euclidean distance between connected nodes as sole cost source completely neglects different slopes or the inherent difficulty of negotiating various terrain types. As both have significant impact on the terrain traversal performance in outdoor environments, any cost metric which does not also take these factors into account is unavoidably incomplete.

Multi-dimensional cost metrics are better suited to store all relevant data about terrain slopes etc. However, they require deliberation of the relative importance of the different cost factors and should provide a way to adjust these weights whenever mission requirements change. With respect to this, approaches that perform scalar aggregation appear to be more flexible than hierarchical methods. Aggregation methods allow continuous adjustment of the cost factor importance through parameter tuning of the utility function, while the alteration of hierarchical constraints introduces unsteadiness of the cost function whenever constraint levels are changed. The increased flexibility is bought at the price of requiring a sensible way to compare the different cost factors, which can be avoided when using hierarchical techniques.

Finally, the combination of high environmental complexity and the navigator's indirect control over the vehicle trajectory (as a result from employing a behavior-based pilot layer) introduce an amount of cost uncertainty that could be modeled more accurately by probabilistic measures than by standard multi-dimensional techniques. However, the use of complex models or accurate estimation requires many samples for each topological edge. As this is an unrealistic proposition for the large scale environment that is targeted in this thesis, any probabilistic elements of the selected cost measure should be introduced rather as an optional component than a central issue. Also, assumptions about the form of the cost distribution should be avoided because they could introduce unsound statistical assumptions.

In any case, the issue of cost consistency is critical due to the use of a behavior-based pilot instead of accurate metrical path planning. As the path planner has no a priori knowledge of the real trajectory that will emerge during robot motion, the calculation of consistent traversal costs requires a substantially different approach than established techniques which compute trajectories and the associated costs *a priori*. As will be presented in section 3.5.4, an *a posteriori* learning scheme has been developed instead which estimates the cost factors based on the robot's experience once the actual trajectory has been

executed. Assuming that a similar trajectory will be produced the next time this edge is traversed, the estimated costs can be used to gradually build a consistent edge traversal cost estimate. In order to make this approach work, it is essential to use components for the cost function which can be correlated with observable robot parameters. This need for *observability* must be taken into account when selecting suitable cost factors for the edge cost metric.

3.5.3 Cost Metric Selection

Based on the deliberations presented in the last section, a **multi-dimensional cost metric** is chosen for the purpose of topological path planning in order to capture *all* relevant cost aspects. Three independent scalar **cost factors** are singled out as the main components of the metric for two main reasons. First, they capture the essential environmental aspects that influence the robot's performance. Second, they can be adjusted after each edge traversal has concluded based on the experiences collected by the robot.

Each of the three scalars summarizes a different cost-relevant aspect of the edge under question. The first value captures how difficult the execution of the edge traversal command is for the piloting layer due to problematic terrain conditions or the occurrence of obstacles etc. and is subsequently called the **risk** R. The second value, **effort** W, is a measure of the amount of energy and time the robot must invest in order to traverse this edge, and is therefore linked to the traveled slope and distance. The third component is defined as the robot's **familiarity** F with the edge. The familiarity value indicates how often the robot has traveled along the edge before and correlates with the accuracy of the effort and risk estimations. Its main purpose is to provide an elegant way for effecting explorative behavior with adjustable priority by tuning edge costs based on how frequent they have been used previously.

3.5.4 Learning Cost Factors through Self-Observation

The abstracted, topological map chosen as world model for the navigator does not explicitly store local terrain traversability but rather assumes that the subordinated obstacle avoidance layer copes with these spatially limited issues. The physical trajectory emerges *in situ*, after the traversal of an edge has been initiated, through complex interaction of the obstacle avoidance layer, the terrain and the traversability properties perceived at that moment.

This explicit abstraction from the real trajectory is a mixed blessing. On the positive side, the topological world model is less dependent upon accurate position information to merge sensor data correctly with the map and thus more robust against errors, which is of great value in cluttered environments where localization is difficult and error prone. On the negative side, the lack of knowledge effectively prohibits the a priori computation of risk and effort measures that reflect the true characteristics of the emerging trajectory. But without such a measure, cost-optimal path planning cannot be achieved. This dilemma is a central issue that results directly from the design decision to split the navigational competences into two different levels of abstraction and reduce metrical knowledge on the higher level to a minimum. A solution is therefore not only vital to the successful application of the navigation system proposed in this thesis, but also contributes significantly to *all* similar architectures.

Proposed Solution

Even though there seems to be no way to estimate a consistent edge cost *prior* to edge traversal without making at least some assumptions about the metrical trajectory that will emerge, it is well possible to do so *after* edge execution. More precisely, the proposed approach's central idea is to learn a consistent cost measure after an edge has been traversed by *observing* the pilot's activities during the robot movement. For instance, by monitoring the behaviors responsible for avoiding dangerous elements such as obstacles, an estimate of the risk involved in traveling an edge can be made. Likewise, behaviors sensitive to energy consumption can be examined to gain a notion of how much electrical energy is spent for the transition.

As the pilot is a behavior-based system, this technique requires that at least the cost-relevant behaviors generate some sort of situation assessment or activity signal which can be observed by other software components. This is certainly true for all behavior within the iB2C framework used by the RAVON robot, because every behavior module exports a **target rating** encoding the behavior's actual 'happiness' with the current situation. Nevertheless, many other established behavior-based systems do also provide such information or can be extended easily, because such assessment signals provide key information for the internal fusion of multiple behavior outputs.

The continuous collection of situation assessments quickly yields a large amount of data samples. To condense these into a single value for each cost factor summarizing the complete edge, data collection is followed by a weighting and integration stage. Two different integration schemes have been developed. For proprioceptive behaviors reacting to internal information only (such as energy consumption), a *temporal integration* of the situation assessments suffices. Contrastingly, exteroceptive behaviors that process stimuli from locations outside the robot such as obstacle avoidance behaviors require an additional *spatial integration* in order to prevent excessive influence of repeatedly detected obstacles.

After finishing data integration, the obtained information about the experienced risk and effort cost factors is encapsulated in an **annotation** and appended to the edge's data record. Multiple traversals produce multiple annotations per edge, which allows a certain degree of probabilistic reasoning about the real edge costs. For path planning, the collected annotations are compressed into a single cost value in two steps. First, all acquired annotations of an edge are combined into a single estimate for the cost vector. Then, the cost vector is aggregated into a single scalar value via an utility function. Each of these steps will be elaborated in detail in the subsequent sections.

A major benefit of the proposed approach is that by virtue of its design, it *guarantees* that the learned cost measure is consistent with the robot's actual behavior guided by the behavior-based subsystem. However, it is also computationally efficient, since the cost learning does not require any additional sensor evaluation, but can effectively re-use the data interpretation already performed by the low-level pilot layer.

3.5.4.1 Learning System Design

A schematic overview of the developed observational system for edge cost learning is shown in figure 3.19. The robot's sensors and actuators constitute the lowest level of the navigation system depicted on the left side of the figure. The actors are controlled by the

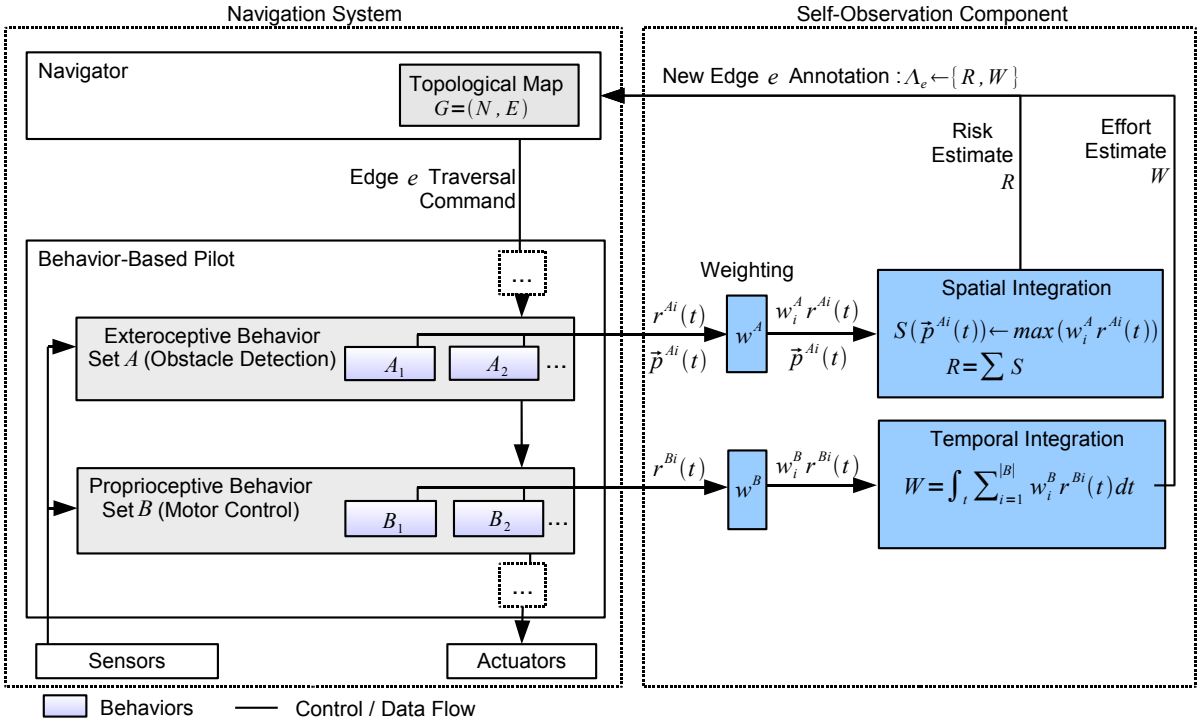


Figure 3.19: Schematic of the observation framework for edge cost learning

behavior-based pilot, of which two sets of behaviors $A = \{A_1, A_2, \ldots\}$, $B = \{B_1, B_2, \ldots\}$ are singled out as their observation provides useful cost information about risk and effort cost factors. The 'Motor Control Behaviors' in set B are proprioceptive and their situation assessment signal $r^{B_i}(t)$ corresponds to the current at time t fed into the actuator assigned to the behavior. r is assumed to be normalized and ranges from 0 if no current is applied up to a value of 1 at full current. The motor control behaviors are activated and controlled by the set A of obstacle avoidance behaviors, which have the purpose of steering the robot safely around obstacles using sensor data such as range measurements. In contrast to B, set A contains exteroceptive behaviors reacting to stimuli (obstacles) originating from a spatial location outside the robot. For these behaviors, the situation assessment $r^{A_i}(t)$ is expected to be 0 if behavior A_i does not see any necessity to influence the robot's trajectory. It should be 1 if the behavior wants to produce the maximal effect achievable in its scope. For example, a value of 1 would be attributed to a velocity reduction behavior trying to bring the robot to a complete halt, or a turning behavior which wants to turn the robot with the maximal possible steering angle. Although not strictly required, it is beneficial for later cost learning if situation assessments vary smoothly between the extremas of 0 and 1. For example, the obstacle behaviors could express dissatisfaction (and a corresponding assessment signal) in proportion to the lowest distance to obstacles closer than a safety margin.

In addition to a suitable situation assessment signal, exteroceptive behaviors are required to export the spatial location $\vec{p}^{A_i}(t)$ of the stimulus responsible for the situation assessment. This data will be used in the spatial integration scheme presented in section 3.5.5. If no single point is responsible, an abstraction, such as exporting centroids or the nearest location, is also acceptable. The topological path planner is placed at the uppermost part

of this schematic and manages the topological map. The right side of the figure displays the main components of the edge cost learner and will be explained in the next sections.

3.5.5 Integrating Behavior Observations into Annotations

Once the path planner has selected an edge e that should be driven, it signals this command to the behavior-based pilot. In the following, t is limited to the execution time of a single edge traversal command e starting at time t_0 and ending at t_1 ($t_0 \leq t \leq t_1$). The edge cost learning component on the right side of figure 3.19 observes the situation assessments $r(t)$ of the two cost-relevant behavior sets A, B. In the depicted scenario, the assessments of set B are directly related to the motor currents, so it is reasonable to assume that their observation can yield a measure of how much energy the edge traversal requires, i.e. an estimate of the 'effort' cost factor W. Similarly, the assessments of the obstacle avoidance set A should contain a notion of how much trajectory deviation due to obstacles was necessary. Under the assumption that traveling close to obstacles constitutes a risk and should be costly, these assessments are suitable to produce an estimate for the 'risk' cost factor R.

During robot motion, the scalar situation assessments $r^{A_i}(t)$ and $r^{B_i}(t)$ ranging from 0 to 1 are first weighted by user-provided factors w^{A_i} and w^{B_i}. This allows specification of the relative cost impact of different behaviors. For example, a behavior responsible for (dangerous) backtracking out of dead-end situations could be awarded a higher w^{A_i} that a simple 'slow down near an obstacle' behavior.

Temporal Integration

After weighting, the continuous stream of assessments is integrated in order to generate one single cost estimate for the complete edge. For proprioceptive behaviors such as those in set B, this integration step amounts to a simple temporal integration of the weighted sum of the set's assessments:

$$W = \int_{t_0}^{t_1} \left(\sum_{i=1}^{|B|} w^{B_i} r^{B_i}(t) \right) dt \tag{3.15}$$

In the depicted control system, the result W is proportional to the total amount of energy spent by all motors during traversal of edge e and can thus be used as *effort estimate* W.

Spatial Integration

For exteroceptive behaviors, plain temporal integration is too simplistic. It leads to a dependency of the final cost estimate on the robot motion speed, as a slower movement causes a longer exposition to the external stimulus and thus a larger integrated value. Although there may be cases where this is desirable (and thus equation 3.15 can be applied), it appears counter-intuitive that a slow robot should judge a given obstacle situation more negatively than a faster model. As a solution, a *spatial* integration step has been developed for combining exteroceptive behaviors situation assessments. For this, the robot must provide a locally stable frame of reference *during edge traversal* onto which the source location $\vec{p}^{A_i}(t)$ of each behavior assessment $r^{A_i}(t)$ is referenced. It is

not required to provide an exact or globally stable coordinate frame, so accurate robot localization remains unnecessary. As data structure for storing the assessments, a *spatial map* S has to be used. S can be modeled as a *grid map* with grid cell sizes equal to the accuracy of the sensor data used by set A or the accuracy of the reference frame, whatever is lower. This helps to make S both: robust against noisy location data and memory efficient. As notational conventions, $S(\vec{p})$ should be interpreted as an access to the grid cell of S which represents location $\vec{p}$. $\sum S$ denotes the sum of all grid cell values and is assumed to be initially zero.

Coming back to the spatial integration scheme, at each sampling time t, the $r^{A_i}(t)$ values of all behaviors are stored into S at their corresponding positions $\vec{p}^{A_i}(t)$ using a maximum update rule:

$$\text{for each } i\text{: } S(\vec{p}^{A_i}(t)) := \max\left(S(\vec{p}^{A_i}(t)), w^{A_i} r^{A_i}(t)\right) \tag{3.16}$$

Once all assessments have been added to S, the integrated scalar risk cost estimate R for edge e can be derived directly through summation ('spatial integration' over S):

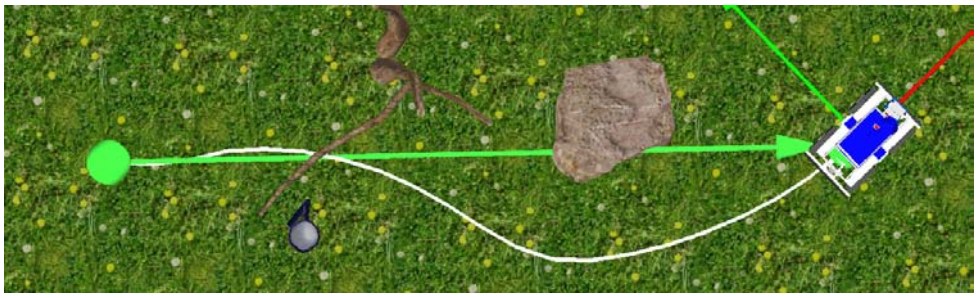
$$R = \sum S \tag{3.17}$$

Figure 3.20 shows an example. A trajectory that emerged as the robot steered around three obstacles is depicted in figure 3.20a. Figure 3.20b shows the corresponding spatial map S that has been constructed using the maximum update rule from the obstacle avoidance behavior situation assessments during traversal. Green colors thereby represent the robot trajectory, whereas the red color intensity indicates the value of S at that location. Figures 3.20c and (d) show the temporal development of the used assessments separately for behaviors reacting to obstacles on the right resp. the left side of the robot.

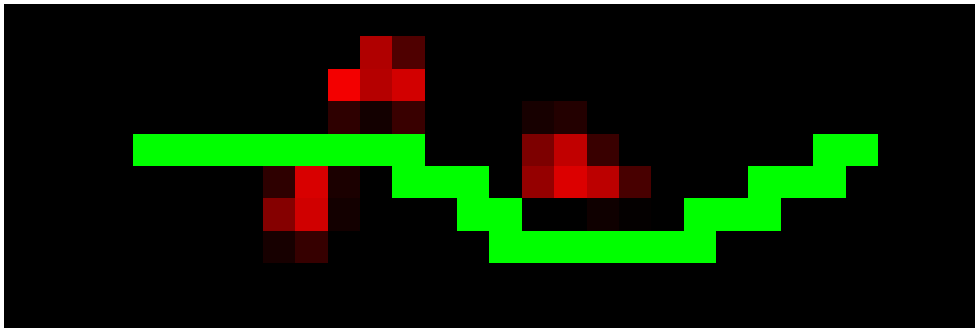
After risk and effort estimates R, W have been generated for the topological edge e, they are attached to it as an *annotation* $\lambda = (R, W)$ and stored in the edge *Annotation Set* Λ (which has already been introduced as part of the formal definition of the edge made in section 3.3.1, but not used until now) for use in cost computation. This process is detailed in the next section.

3.5.6 Computing Cost Factors from Annotations

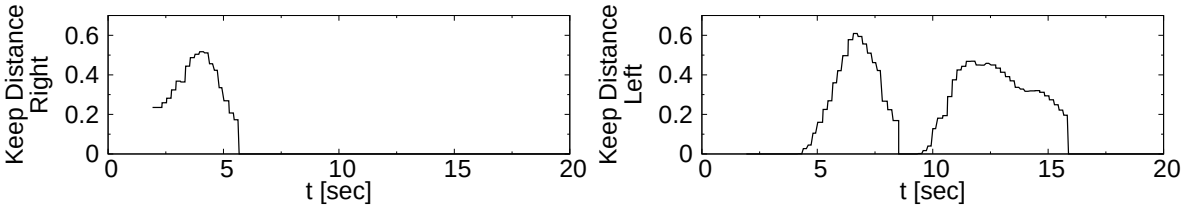
For the first step, let's assume that an edge e is annotated with the set $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ where $\lambda_i = (R_i, W_i)$, $1 \leq i \leq n$. It is expected that each annotation is a sample taken from an unspecified distribution centered at the true values of effort and risk. Without assuming any specific form of this distribution, it is still possible to estimate the *sample mean* and *unbiased sample variance* of the annotation set. Both should be accounted for in the cost metric in order to maximize its information content. To do so, a combined annotation $\widehat{\lambda} = (\widehat{R}, \widehat{W})$ is computed containing the estimated mean plus an adjustable portion δ of the estimated standard deviation:



(a) Robot trajectory



(b) Spatial map S



(c) Right obstacle beh. assessments

(d) Left obstacle beh. assessments

Figure 3.20: Spatial integration example

$$\mu = (\mu^R, \mu^W) = \frac{1}{n} \sum_{1 \leq i \leq n} (R_i, W_i) \tag{3.18}$$

$$\sigma = (\sigma^R, \sigma^W) = \sqrt{\frac{1}{n-1} \sum_{1 \leq i \leq n} \left(\left(R_i - \mu^R\right)^2, \left(W_i - \mu^W\right)^2 \right)} \tag{3.19}$$

$$\widehat{\lambda} = \left(\widehat{R}, \widehat{W}\right) = \mu + \delta\sigma \tag{3.20}$$

The influence of the deviation weighting parameter δ can be understood more intuitively by interpreting it as the path planner's overall '**pessimism**'. Positive values of δ increase an edge's cost proportional to the estimated standard deviation. Thus, this parameter setting favors edges with low variance, typically encountered as part of well established routes with little critical situations that can 'go wrong' and lead to increased costs. Intuitively, the path planner is pessimistic and expects the worst. Consequently, it prefers to select well known paths that do not contain cost surprises.

On the other hand, using negative values for δ produces 'optimism under uncertainty' and actually favors such unstable connections. Aside from critical situations, another possible

cause of high variance can be that the edge has simply been used rarely. This effect can be exploited to produce exploratory behavior, with δ determining the exploration / exploitation tradeoff. However, care must be taken for $\delta < 0$ as many path planning algorithms (such as the dijkstra algorithm) require strictly positive edge traversal costs in order to terminate.

The estimation of the third cost factor familiarity F cannot be accomplished through observation of the pilot, since it is a measure of the map itself and the pilot is unaware of this global representation. Therefore, the familiarity cost $\widehat{F}$ is derived by taking the cardinality of e's annotation set Λ and setting $\widehat{F} = |\Lambda|$. Since an annotation is added each time an edge is traveled, frequently traversed edges quickly increase their familiarity score.

Concatenating all three combined cost factors together produces the raw **Cost Vector** $\vec{C}$ of edge e:

$$\vec{C} = \begin{pmatrix} \widehat{R} & \widehat{W} & \widehat{F} \end{pmatrix}^{\mathrm{T}} \tag{3.21}$$

3.5.7 A Utility Function for Scalar Aggregation

Once the cost vectors $\vec{C}^e$ are known for all edges e in E, an ordering relation can be defined to compare them. As discussed in section 3.5.2, the relative cost factor importances of a multicriterion cost measure can be adjusted more flexibly if scalar aggregation is used instead of a constraint hierarchy. As *utility function* $u(\vec{C}^e) : \Re^3 \rightarrow \Re$ for the aggregation of edge e's three dimensional cost vector $\vec{C}^e = \begin{pmatrix} \widehat{R^e} & \widehat{W^e} & \widehat{F^e} \end{pmatrix}^{\mathrm{T}}$, plain weighted averaging is chosen due to its simplicity according to the formula:

$$u\left(\vec{C}^e\right) = \alpha\widehat{R^e} + \beta\widehat{W^e} + \gamma\widehat{F^e} \tag{3.22}$$

where α, β, γ specifies the relative weight of each cost aspect.

It can not be expected that the risk, effort and familiarity cost estimates are always comparable in scale or do fall within predetermined limits. Instead, the numerical range of each cost component depends largely on the initially unknown structure of the topological map in conjunction with the underlying terrain and is in principle unbounded. Thus, if the weights α, β, γ are not carefully chosen for each operation environment, it can easily happen that one cost factor dominates the others just because its value range is unexpectedly large in the current scenario.

In order to avoid this effect which complicates the definition of general path planning priorities, a *normalization* step can be used to make the scales of the different cost aspects comparable regardless of the current scenario. It appears to be sensible to scale all cost values into the range $[0, 1]$ using the current map's respective risk, effort and familiarity maxima over all map edges. Using the maximal values $\widehat{R}^{\max}, \widehat{W}^{\max}, \widehat{F}^{\max}$ found up to now results in scaled values between 0 and 1 and leaves the relative proportions within a category intact (an edge that is twice as risky as another will remain so regardless of scaling). However, the proportions *between* categories may change if new maximal values are found. This is not assumed to be a significant problem, since the scaling factors tend to stabilize quickly as measurements accumulate.

On the basis of $\widehat{R}^{\max}$, $\widehat{W}^{\max}$ and $\widehat{F}^{\max}$, the **normalized utility function** $\tilde{u}\left(\vec{C}\right)$ can be defined:

$$\tilde{u}\left(\vec{C}^e\right) = \alpha\frac{\widehat{R^e}}{\widehat{R^{\max}}} + \beta\frac{\widehat{W^e}}{\widehat{W^{\max}}} + \gamma\frac{\widehat{F^e}}{\widehat{F^{\max}}} \tag{3.23}$$

As only the relative proportions of the cost factor weights are important, it is safe to impose the constraint $\alpha + \beta + \gamma = 1$ without losing expressiveness. This constraint ensures that $\tilde{u}\left(\vec{C}^e\right)$ is bound by the interval $[0, 1]$.

Analogous to the variance weight parameter δ, the weights α, β, γ can be given a more intuitive interpretation as the **motivational state** of the global navigation system. Seen from this motivational viewpoint, α expresses the path planner's 'fear' level (high α values increase the cost of risky drive commands, making them less likely to be included in paths). Likewise, β denotes the robots 'impatience' and γ the robots 'curiosity'.

The introduction of a motivational state allows a human operator to influence the path planning strategy in an intuitive way, so that it can be set according to the current requirements of a mission. Furthermore, it provides an interface that facilitates the addition of another (behavior-based) control layer on top of the path planner in the future, influencing the three basic motivations. Although this issue has not been approached in the scope of this thesis, the addition of such a high-level system is an interesting area for future work and promises some benefit for navigational tasks. For example, repeated failures to reach certain way points could increase the fear level, automatically leading to the selection of safer paths. Likewise, the lack of new experiences could increase the robots curiosity, making well-traveled paths more and more unattractive. A distributed, behavior-based architecture that implements these deliberations can be seen as a model for distributed decision-making, with the weight aspects computed separately and the concrete action selected by the path planning subsystem. Some further interpretation of these effects in the context of the simulation of emotions have been published in [Hirth 07].

3.6 Putting it Together: Topological Navigation

With the building blocks presented in the last sections, the complete topological navigation scheme can be put together. It has to be flexible enough to allow maneuvering the robot to an arbitrary position in the environment while taking advantage of the topological map as much as the operator deems necessary. In the general case, both start and goal locations must *not* coincide with topological nodes. Such a flexible movement sequence consists of three distinct phases:

1. Approach: The robot moves from its current position toward a topological node n^s.
2. Path Traversal: The robot travels along a path $P = e_0, e_1, \ldots, e_n$ of map edges between $e_0.n = n^s$ and $e_n.n' = n^e$.
3. Departure: The robot moves from the last topological node n^e to the goal position $\vec{p}^{end}$.

Given the **map entry** and **map exit** nodes n^s and n^e plus the optional **goal position** $\vec{p}^{\,end}$ by the user of the navigation system, this movement sequence can be implemented using the various functions and algorithms introduced in this chapter.

First, the approach stage is initiated by invoking the ApproachTargetECSPose function (alg. 3, p. 46) with parameters ($e_0.n.\vec{p}$, $e_0.n.\vec{p}$, $e_0.n'.\vec{p}$). Likewise, the departure stage is mapped to the call ApproachTargetECSPose($e_n.n'.\vec{p}$, $\vec{p}^{\,end}$, $\vec{p}^{\,end}$). Both movements do not correspond to an edge contained in the topological map and thus no place exists to attach annotations. Therefore, the behavior observation and cost learning scheme is not used during these stages.

The path traversal phase is the central part of the navigation system, as the computed path should be cost-optimal according to the set motivational state and also take advantage of previous cost experiences. Thus, the implementation utilizes most of the various algorithms introduced in this chapter. In order to compute the path P between map entry n^s and exit n^e, the stored edge annotations are condensed into one scalar cost value $\tilde{u}(C^e)$ per edge e using the current motivational state of the path planner and the techniques detailed in the previous sections 3.5.6 and 3.5.7. Then, the standard dijkstra algorithm [Dijkstra 59] is applied to find the cheapest path between entry and exit node. This path is subsequently traversed by alternatingly calling ApproachTargetECSPose($e_i.n.\vec{p}$, $e_i.n'.\vec{p}$, $e_{i+1}.n'.\vec{p}$) with increasing i and awaiting edge traversal completion using the DetectTargetArrivalWithPostponement algorithm (alg. 4, p. 48). If the edge has been successfully traversed, the reached node is then relocalized if it was speculative (sec. 3.4.3) and potentially fused with nearby nodes using the FuseNodes algorithm (alg. 5, p. 51). Additionally, a new annotation is attached to the traversed edge using the observation-based cost learning scheme (sec. 3.5.5). In case the failure detection detailed in section 3.4.5 signals an unsuccessful edge traversal, the RepresentFailure algorithm is executed and a new path is planned from the node inserted at the current robot position to the end node n_e.

3.7 Experiments and Results

In order to validate the presented methods, a series of experiments has been conducted. These were carried out using both the simulation environment introduced in appendix A and the real robot in a test area featuring various terrain peculiarities such as hills, trees, bushes, tall grass as well as diverse impassable boulders, poles or concrete structures. Details concerning the implementation of the navigation system as well as the experimental results are presented below.

3.7.1 Implementation

The methods presented in this chapter have been implemented as part of a global navigation layer put on top of the behavior-based pilot described in section 2.2. As can be seen in figure 3.21, the navigator contains and manages the topological map data structure introduced in this chapter. Control of the robot is achieved using only a very narrow interface between the high level navigator and the pilot. To control the robot actions, the navigator activates the Approach Target Position and Approach Target Pose interface behaviors of the pilot. This is implemented by first converting the active ECS target position or pose into

the working coordinate system WCS using the conversion matrix M_{ECS}^{WCS} and supplying the resulting WCS position $\vec{p}_{target}^{\,WCS}$ or pose $(\vec{p}_{target}^{\,WCS}, \gamma^{WCS})$ to the pilot. If a WCS *position* is to be attained, an activation value of 1 is sent to the Approach Target Position iB2C behavior within the piloting layer. For WCS *poses*, the Approach Target Pose behavior is activated instead.

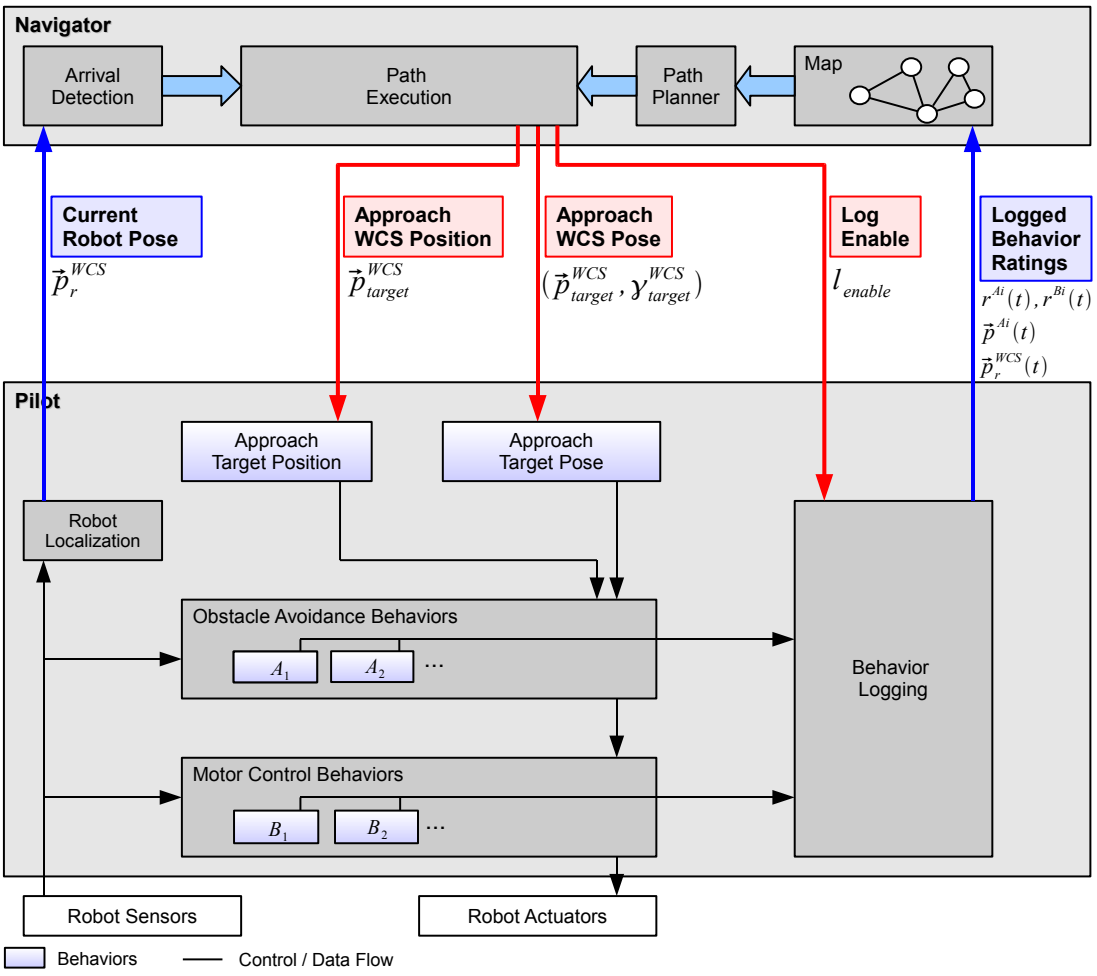


Figure 3.21: Navigator - pilot interface

As feedback from the pilot, the navigator primarily requires the current position estimate of the robot to determine whether a travel command has been successfully completed. Given the WCS robot position estimate $\vec{p}_r^{\,WCS}$, the navigator computes the required ECS position $\vec{p}_r^{\,ECS}$ using $\vec{p}_r^{\,ECS} = M_{WCS}^{ECS} * \vec{p}_r^{\,WCS}$. Furthermore, the cost learning scheme requires that the navigator can observe the target ratings of the pilot behaviors dealing with the motor currents and obstacle avoidance. Table 3.5 summarizes the behavior sets A and B that have been selected for this purpose.

For performance reasons, the observation part is implemented through a logging mechanism that is integrated into the pilot. Whenever the navigator issues a command that corresponds to the traversal of an existing edge in its topological graph, it also activates the logging mechanism in the pilot using a flag l_{enable}. The created log contains the target ratings $r(t)$ of the cost relevant behaviors as well as their stimulus locations $\vec{p}(t)$ and the current robot pose estimate $\vec{p}_r^{\,WCS}(t)$. Upon completion of an edge traversal, the navigator

Set	Main Group	Full Name	w	Purpose
A	Avoid Left	FW Keep Dist Left (Crit) Rot	1	Rotates away from (critical) left obstacles during forward motion
		BW Keep Dist Left (Crit) Rot	1	Rotates away from (critical) left obstacles during backward motion
		Keep Dist Sideward Left	1	Translates away from obstacles on left side
	Avoid Right	FW Keep Dist Right (Crit) Rot	1	Same as above, for right side
		BW Keep Dist Right (Crit) Rot	1	
		Keep Dist Sideward Right	1	
	Evasion	Forward Evasion	0.75	Selects right or left avoidance direction for obstacles directly ahead
		Backward Evasion	0.75	Selects right or left avoidance direction for obstacles directly behind
	Slowdown	FW Obstacle Slow Down	0.5	Reduces forward speed near front obstacles
		BW Obstacle Slow Down	0.5	Reduces backward speed near rear obstacles
		Side Obstacle Slow Down	0.5	Reduces overall speed near side obstacles
	Stop	FW Crit. Obst. Stop	2	Stops forward motion before hitting obst.
		BW Crit. Obst. Stop	2	Stops backward motion before hitting obst.
B	Curr. Limit	Limit FR	1	Monitors front right motor current
		Limit FL	1	Monitors front left motor current
		Limit RR	1	Monitors rear right motor current
		Limit RL	1	Monitors rear left motor current

Table 3.5: Behavior sets observed for edge cost learning

commands the logging system to stop and transmit the data that has been collected so far. Based on the received log data, the cost learning algorithm can reconstruct all relevant information about the completed edge traversal and compute the resulting cost estimate. With this logging mechanism, cost learning does not cause any ongoing data transfer during the edge traversal itself. Instead, the whole log is transferred in one piece just after a move has concluded. Since navigator and pilot software is executed on different physical machines, this saves a significant amount of network communication overhead.

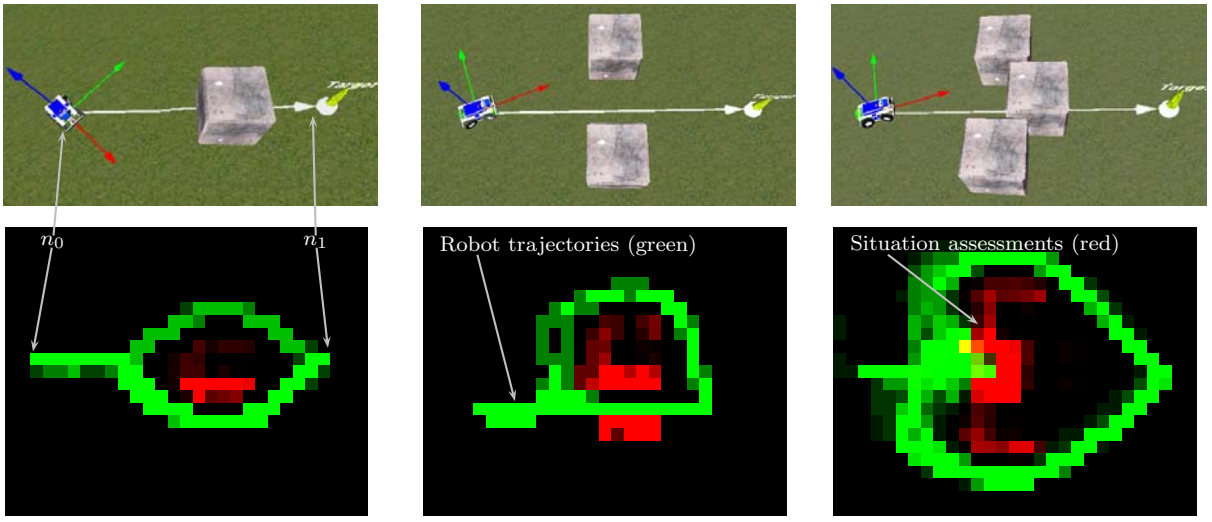
3.7.2 Quantitative Analysis

The first set of experiments has been designed to study the quantitative behavior of the edge cost observation framework in both simulation and real world settings. For this, the cost estimation for a single edge has been observed in situations that become increasingly risky and energetically taxing.

3.7.2.1 Simulation

Figure 3.22 shows three edge cost estimates obtained using the SimVis3D framework. The top row shows a topological edge $e = (n_0, n_1)$ obstructed by three increasingly complex obstacle configurations. The middle row shows the corresponding spatial map S with a grid cell size of 0.75 m averaged over 50 edge traversals. Green pixels indicate locations passed by the robot, red pixels denote stored situation assessments $w_i^A r^{A_i}(t)$ of the obstacle avoidance behavior set. In the bottom row, the resulting cost estimates are listed along with the behavior classes that contribute the most to the obtained risk estimate.

As can be seen, the increase in obstacle number and placement complexity corresponds well with the increase of the edge's derived risk estimate. This is a very important result.



Cost Estimates (μ, σ)	
Risk	(**1.8**, **0.3**)
Effort	(**690.18**, **37.28**)

Main Risk Contributors	
Avoid Left	75 %
Avoid Right	24 %

(a) Single Obstacle

Cost Estimates (μ, σ)	
Risk	(**5.6**, **1.2**)
Effort	(**405.8**, **75.22**)

Main Risk Contributors	
Avoid Left	53 %
Avoid Right	47 %

(b) Passage

Cost Estimates (μ, σ)	
Risk	(**11.4**, **3.9**)
Effort	(**1058.84**, **175.9**)

Main Risk Contributors	
Avoid Left	35 %
Avoid Right	34 %
Evasion	26 %

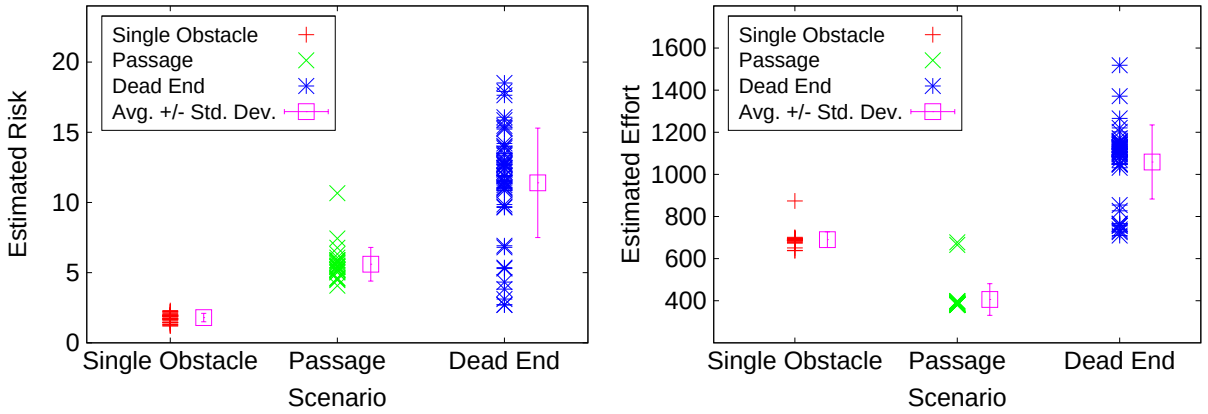
(c) Dead End

Figure 3.22: Cost estimates for three different simulated obstacle scenarios

It documents the principal effectiveness of the proposed risk learning scheme in deriving abstracted cost information from the pilot behaviors. The figure also illustrates the uncertainty of the actual trajectory driven by the robot, as minute differences in starting orientation can decide whether the obstacle is passed on the right or left side. This uncertainty is the main contributing factor for the estimated standard deviation. It also becomes apparent in the fact that *both* the 'Avoid Right' and 'Avoid Left' behavior classes are significant contributors for the risk estimates.

A more detailed view of the individual annotations leading to the estimated cost values is depicted in figure 3.23. It can be observed that although both risk and effort observations vary and contain several outliers, the estimations tend to cluster around a certain central value. Mean and standard deviation are adequate to describe this central value in all three examined cases.

Whereas risk estimates increase monotonically with increasing obstacle complexity, the effort estimate is actually lowest for the 'Passage' scenario, followed by 'Single Obstacle' and 'Dead End'. This apparently unintuitive result can be explained when considering figure 3.24. Here, the length of the real trajectory that the robot traveled during edge traversal is plotted against the obtained annotation effort. As can be seen, the traveled distance is actually lowest for most annotations of the 'Passage' scenario (corresponding to trials where the robot passed straight through the passage between the two obstacles), followed by the 'Single' and 'Dead End' scenarios. Overall, a strong correlation between travel distance and effort can be observed. As the simulated scenarios are flat, distance



(a) Risk (b) Effort

Figure 3.23: Individual cost annotations for the simulated scenarios

indeed remains as the single most important factor that influences the robot's energy consumption.

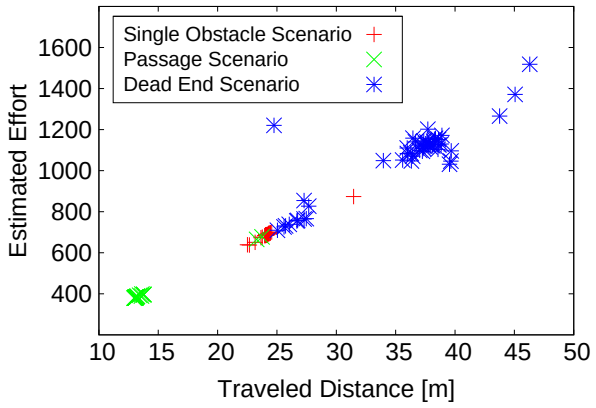
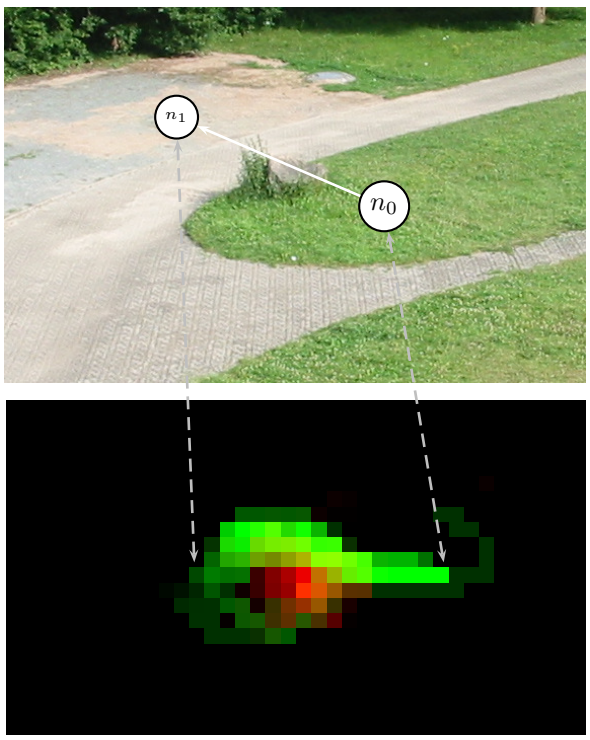


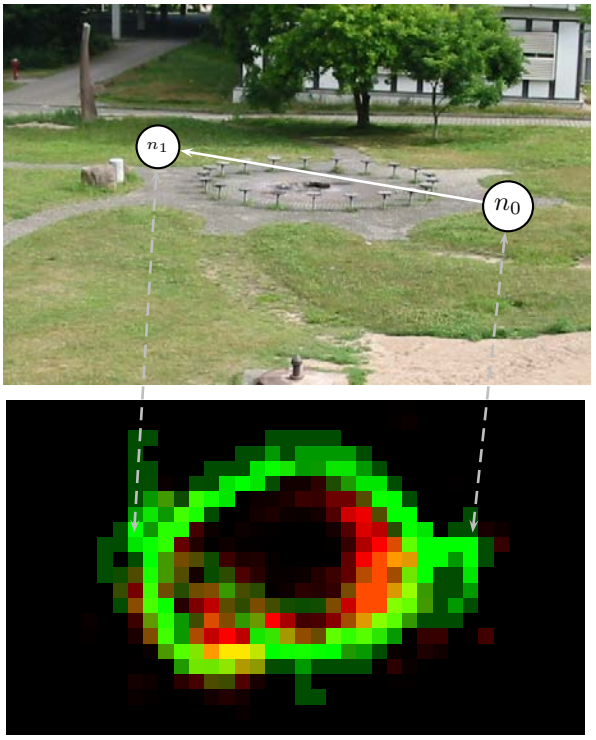
Figure 3.24: Effort per distance in the simulated scenarios

3.7.2.2 Real World

The quantitative experiments conducted in the real world resemble those conducted in simulation. Two scenarios with increasing difficulty have been selected. The first scenario 'Stone Block' contains an isolated, single stone obstacle in the middle of the edge to be traversed. The setup along with the resulting spatial map (averaged over 30 trials) and the derived cost factors is shown in figure 3.25a. The second scenario 'Ring of Seats' contains a circle of permanently mounted, impassable stools as well as an additional stone block on the far side of the seat ring. The scene is more difficult to traverse than the 'Stone Block', since the obstacles are spread across a larger area and are less compactly placed. The seats themselves are also more challenging to detect for the pilot (the support of the seats is rather delicate in the horizontal scanner's measurement plane). Figure 3.25b shows the scene and the derived estimates.

As before in the simulation experiment and required for a consistent cost measure, the estimated risk and effort costs rise for the more difficult scenario. However, the single





Cost Estimates (μ, σ)	
Risk	(**4.9**, **3.8**)
Effort	(**669.0**, **372.6**)

Main Risk Contributors	
Avoid Left	54 %
Avoid Right	24 %
Evasion	22 %

(a) Stone Block

Cost Estimates (μ, σ)	
Risk	(**19**, **13.5**)
Effort	(**1275.6**, **835.41**)

Main Risk Contributors	
Avoid Right	44 %
Avoid Left	31 %
Evasion	20 %

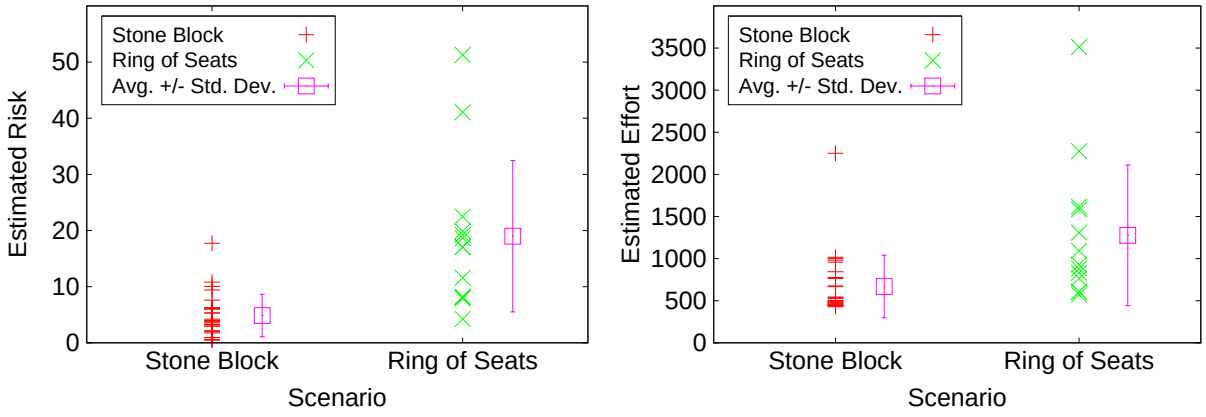
(b) Ring of Seats

Figure 3.25: Cost estimates for two different real obstacle configurations

annotations are more spread out than in simulation (figure 3.26), which results in a larger variance of the cost measure. This can be attributed to the influence of sensor noise and mechanical inaccuracies that cause additional variations. Furthermore, the 'Ring of Seats' is asymmetrical, as the stone block only poses a problem when the seat ring is traversed around the left side. As a consequence, further spread of the cost estimates must be expected.

3.7.3 Qualitative Analysis

In the second set of experiments, the performance of the proposed topological map scheme and edge cost learning was tested in a larger environment. These experiments were conducted to allow a qualitative analysis of the benefit that can be drawn from the proposed method for the optimization of path planning in a given topological map.

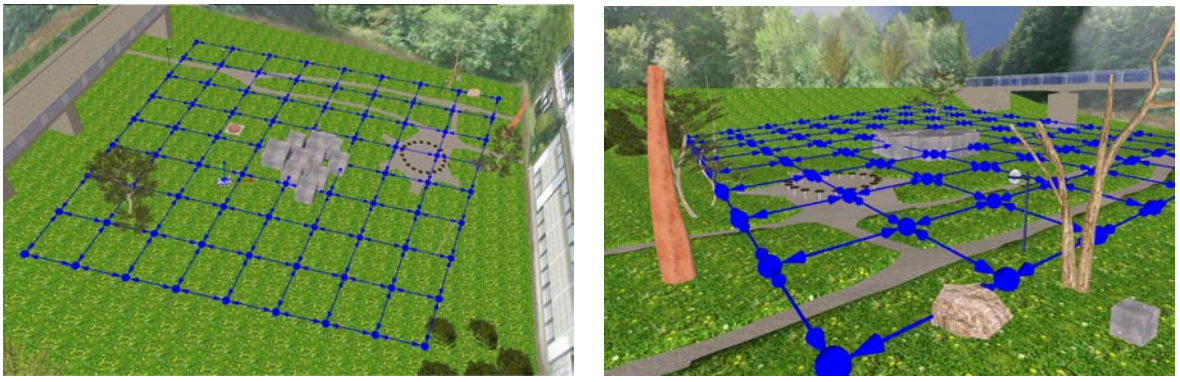


(a) Risk (b) Effort

Figure 3.26: Individual cost annotations for the real scenarios

3.7.3.1 Simulation

Figure 3.27 shows two views of the scenario, which was modeled after the testing area used for the real world experiments. As can be seen, the simulation contains several realistic obstacles such as trees, bushes, street lights or the already presented ring of seats. To further increase the challenges for the obstacle avoidance layer, the center of the scene has been obstructed with a set of (not so realistic) large stone blocks forming an irregularly shaped but approximately convex blocked area.



(a) (b)

Figure 3.27: Initial state of the qualitative evaluation scenario

The overlaid topological map initially contained only speculative (blue) nodes and edges without cost annotations. It has been generated from supplied corner nodes through recursive midpoint subdivision of the corner node connection edges. The grid structure contains 273 short edges (with lengths between 5-7 meters) and thus allows to learn a relatively fine grained estimate of the travel costs. Its regular layout does not introduce any additional knowledge about suitable routes or good placement of topological nodes. The initial map consequently happens to contain two unreachable nodes, one inside the central stone group and the other inside the ring of seats. As can be seen especially well in figure 3.27b, the scene is not entirely planar, but contains a significant slope close to

one side of the bridge. Also, the walkway embedded into the grass lies a little bit lower than the surrounding terrain. The effective offset varies along the outline of the path and forms a negative obstacle for the robot at some places, while other areas are passable.

During the experiment, the developed navigation scheme has been used to generate path traversal commands between randomly selected map nodes. In order to collect edge cost annotations as homogeneously as possible, the motivational state $(\alpha, \beta, \gamma, \delta) = (0, 0, 1, 0)$ has been used for the path planner, so that an edge's cost was maximally dependent on its familiarity value, i.e. the number of already attached annotations. Thus, paths along edges with few annotations have been preferred. As further parameters, a minimal node distance of 3 m for node fusion, a target arrival distance of $d_{arrival} = 3$ m and a traversal failure timeout $T_{fail} = 40$ s was used.

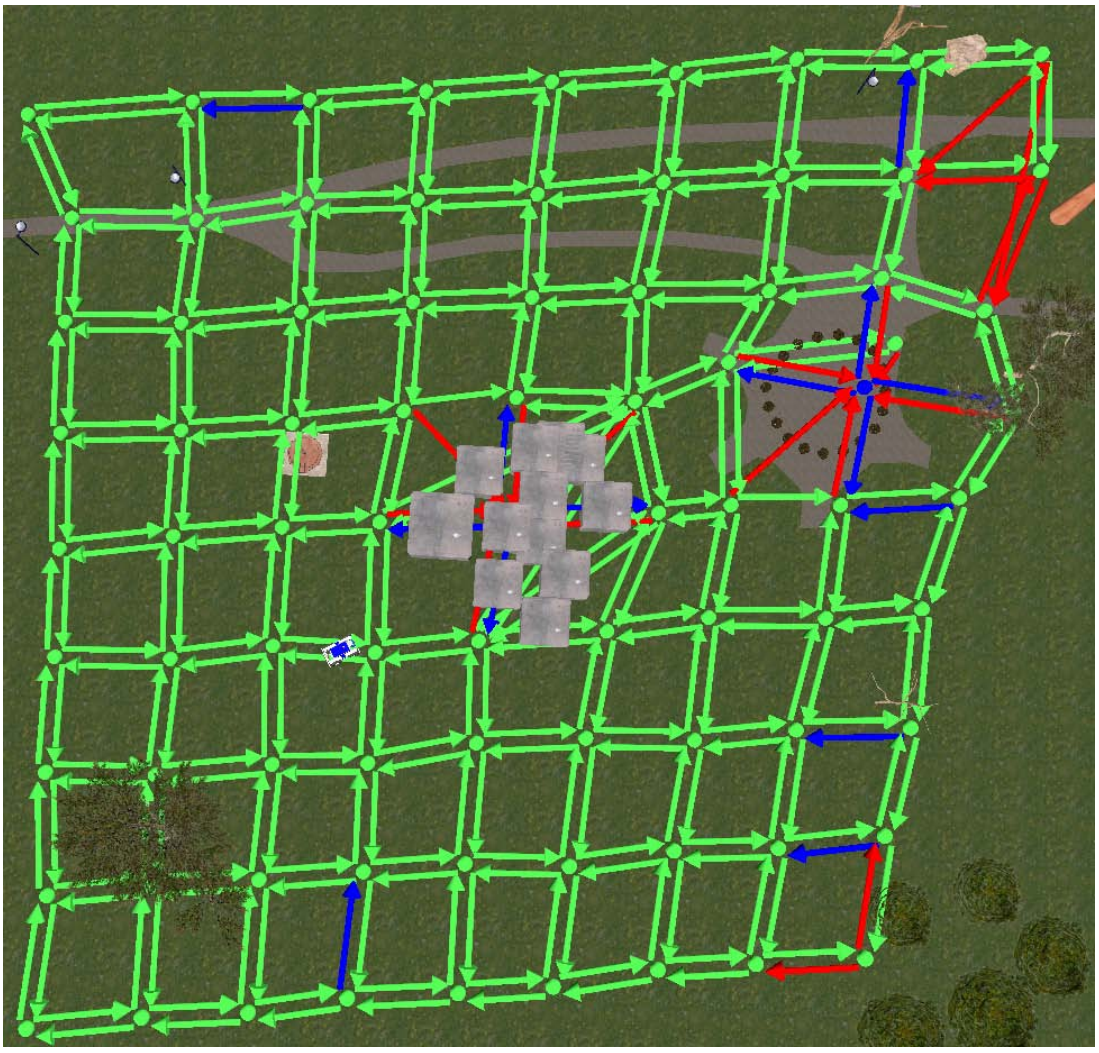


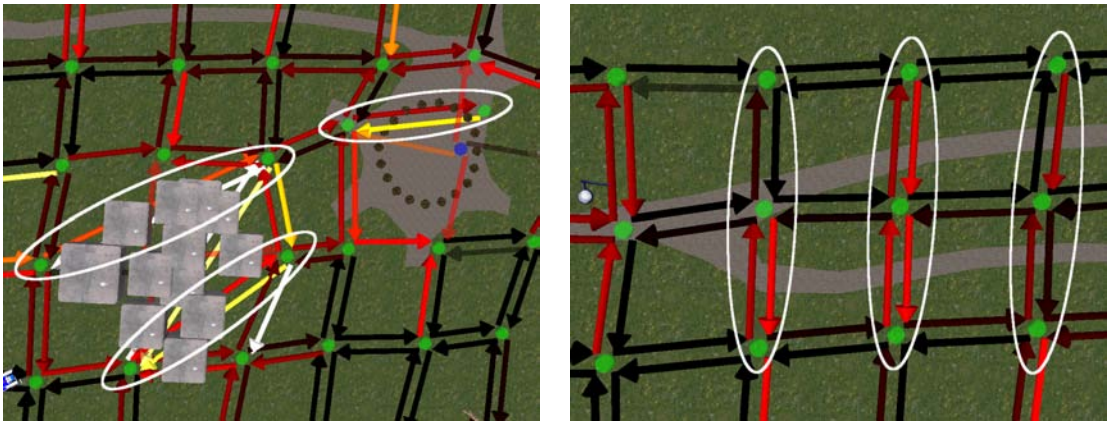
Figure 3.28: Map layout after 1479 edge traversals

Figure 3.28 shows the map layout after a total of 1749 edge traversals (equivalent to ≈ 6.6 traversals per edge, contrast and edge thickness has been altered for better viewability). Evidently, most of the nodes were found to be reachable (green). Node relocalization has displaced the speculative nodes around the stone blocks and the seat ring, as they could not be reached well at their initial locations. The other nodes have remained at

their original positions, as the robot could approach them precisely using the DetectTargetArrivalWithPostponement algorithm. Most edges have also been identified as traversable (green), with several interesting exceptions. As the two nodes inside the stone block and the seat ring are not reachable, all attempts to travel along an edge leading to them have failed. Consequently, these edges have been marked as untraversable (red) by the RepresentFailure algorithm. Normally, this method also inserts a new node at the current robot location. However, with the exception of one node near the upper-right part of the seat ring, the introduced node was placed too close to an already existing node and has therefore been fused with it by the node fusion step. The only remains of these fusions are several new edge connections between previously unconnected nodes (e.g. from the upper right to the lower left part of the stone block). This ultimately lead to the edge layout that can be seen in figure 3.28. Finally, the area around the two rightmost corner nodes was not well accessible for the robot because of some narrow obstacle configurations and kinematic constraints when the pilot was requested to turn around in a tight spot. Thus, some of the initial edges have also been marked as impassable.

Figure 3.29: Hot-cold map of learned risk costs

Figure 3.29 shows a hot-cold map of the learned risk cost factors. Increasing risk costs are visualized using 'hotter' colors, e.g. risk values increase for the color sequence black, red, yellow, white. Overall, it can be seen that the higher estimates are distributed around obstacles, whereas free space is measured as having no risk (the edges have black color). This qualitatively correct impression of the map exhibits two interesting peculiarities. First, the new edges that have been introduced by the failure representation and node fusion steps (marked in the close-up figure 3.30a) have been judged as extremely high risk connections. This can be explained by the fact that these edges are generally over twice as long ($\approx$ 12 m) as the initial edges and run across the most prominent obstacle configurations. Thus, although the pilot has apparently been able to travel along these edges, they are not a good alternative to using the already known sequence of original edges when risk is a significant cost factor. Second, the edges that cross the walkway in the upper part of the test scenario are predominantly marked with a medium risk, although no prominent proturing obstacle is visible in the vicinity (figure 3.30b). Upon closer examination, it turns out that the navigator has observed the pilot's obstacle avoidance reactions to *negative* obstacles at these places. This reaction has been triggered by the fact that the walkway is placed somewhat *below* the level of the surrounding grass (see figure 3.27b), creating steps that need to be circumvented in the otherwise apparently smooth terrain.



(a) New high risk edges (b) Risky walkway crossings

Figure 3.30: Two risk peculiarities

Similar to the presented risk estimates, figure 3.31 shows a hot-cold map of the learned effort costs. Because the edges have similar lengths, the energetical effort estimates are relatively uniform. A noteworthy exception to this is the lower left area of the test environment. Here, the effort estimates exhibit a pronounced asymmetry between edges running in opposite directions. As can be seen in the close-up (figure 3.32), this part of the test environment is rather sloped, and the topological cost learning algorithm has correctly observed that edges running downhill are energetically much cheaper than those leading upwards.

Apart from that observation, some correlation between higher risk and a higher effort can be made out from the hot-cold map. The high-risk edges around the stone block also exhibit a high effort, similar to the edges close to the badly accessible corner nodes. However, little additional effort seems to be caused by the negative obstacles of the embedded

Figure 3.31: Hot-cold map of learned effort costs

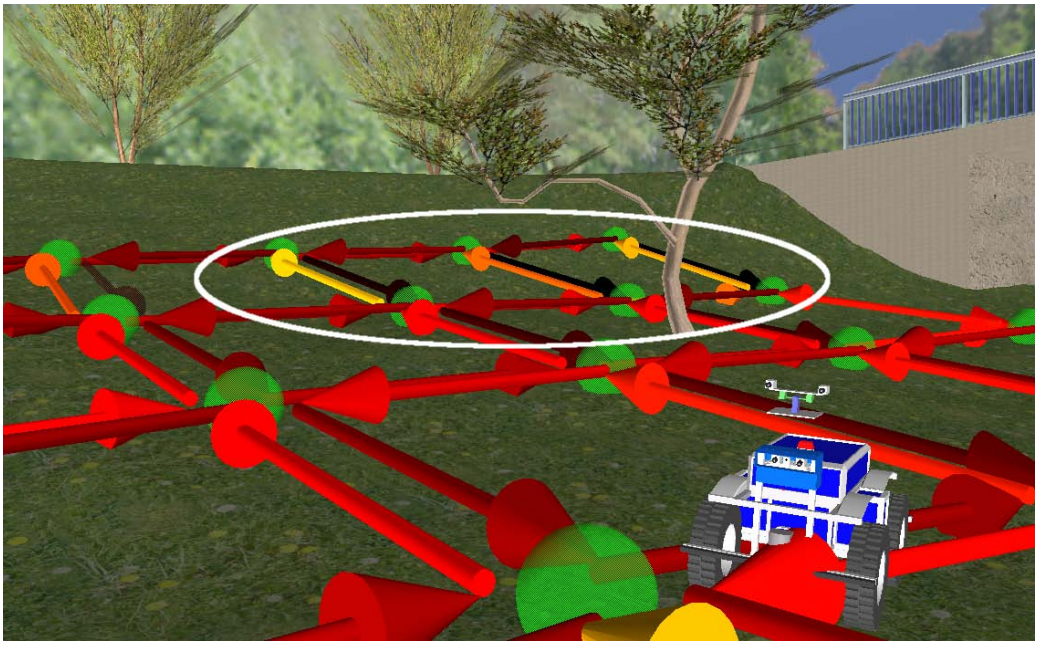


Figure 3.32: Asymmetrical effort estimates on sloped terrain

walkway. Evidently, these obstacles could be circumvented with less additional movements (and thus, less effort) than the more pronounced obstacle configurations mentioned before. Visual inspection during the traversal of these edges confirm this interpretation.

		Terrain encountered along Edge		
		Any	Free ($\widehat{R} < 1$)	Obstructed ($\widehat{R} \geq 1$)
Number of Edges	n	266	189 (71%)	77 (29%)
Total Risk	$\sum \widehat{R}$	267	26	240
Total Effort	$\sum \widehat{W}$	73192	49384	23807
Total Familiarity	$\sum \widehat{F}$	1749	1223	526
Avg. Risk	$\sum \widehat{R}/n$	1.00	0.14	3.13
Avg. Effort	$\sum \widehat{W}/n$	275.6	261.3	309.2
Avg. Familiarity	$\sum \widehat{F}/n$	6.6	6.5	6.8
Median Risk Variation Coeff.	$med\left\{\frac{\sigma_r}{\mu_r}\right\}$	-	-	**61%**
Median Effort Variation Coeff.	$med\left\{\frac{\sigma_w}{\mu_w}\right\}$	13%	13%	21%

Table 3.6: Cost statistics for simulation scenario

Table 3.6 summarizes some interesting statistical properties of the cost annotations inserted into the map. The first column lists the total and average of the three cost factors for the 266 edges of the entire map. In the second and third columns, the edges are separated according to the estimated risk $\widehat{R}$ into edges that run along predominantly free terrain ($\widehat{R} < 1$) and edges that contain more severe obstructions ($\widehat{R} \geq 1$). It can be observed that the simulation scenario is made up of about 70% low-risk edges, which have an average risk of only 0.14 and account for only 10% of the total risk costs. The 30% high-risk edges consequently cover the remaining 90% of the total risk with an average of 3.13 per edge. From these figures, it can be concluded that risk is indeed distributed rather unequally across the map. In contrast to this, the effort cost variations between free and obstructed edges are more moderate, with average values of 261 and 309, respectively. Thus, effort is somewhat correlated with risk (as observed before), but still retains a significant amount of independence from the other cost factor.

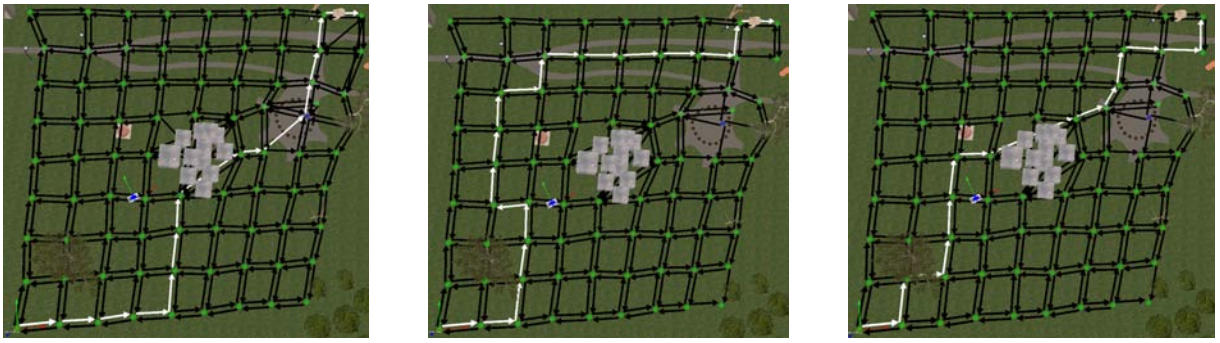
A further interesting piece of information can be learned from the *median variation coefficients* listed in the bottom part of the table. The **variation coefficient** of a distribution is defined as the distribution's standard deviation divided by its mean. In this way, it is a measure of the *relative* dispersion of the underlying distribution, independent of the numerical scale of the mean value. Since one variation coefficient can be computed for each edge, the table presents the *median* of all variation coefficients to give an impression of the 'typical' spread of the cost annotations for that edge subset. The median is used here because it is more robust against outlier values than the arithmetic mean of the variation coefficients.

From the presented values, one can gain an impression of the overall cost consistency between multiple traversals of the same edge. For the effort cost factor, the median

variation coefficient of 13% indicates that the overall effort cost observations are typically distributed rather tightly around a well defined mean value. This supports the idea of the proposed cost learning scheme and also eases the prediction of effort costs (a topic that will be addressed in the next chapter). That benign property even holds for the obstructed edges where trajectories tend to be disturbed more strongly and deviate more from the ideal line of sight. Here, the median variation coefficient rises to 21%, but still remains comparably low.

Unfortunately, the same cannot be stated for the median risk coefficient of 61% of the obstructed edges. Apparently, the risk costs fluctuate much more wildly between cost annotations for these edges than the effort costs. A possible reason for this is the impact of sensor noise on the obstacle detection algorithms which are the foundation of the avoidance behaviors reactions and thus, the observed risk. The measurements for these exterioceptive behaviors probably contain much larger noise than the data for the proprioceptive behaviors responsible for effort costs. As a final note, the missing entries for the low-risk edges or the complete set are caused by the fact that the variation coefficient becomes meaningless for distributions with a mean close to 0 (as the divisor approaches this value, the coefficient approaches infinity) and are therefore omitted.

In the last part of the qualitative evaluation, the effect of the learned edge traversal costs on path planning itself has been examined. For this, the topological path planner has been prompted to generate paths from the the low left corner node towards the upmost right node that minimize either a) plain metrical distance, b) the learned risk costs or c) the learned effort costs. The paths were generated by using either the initial, unannotated map for planning (as will be presented in chapter 4, cost estimation for a completely unannotated map falls back to a distance based metric) or setting the motivational states $(1, 0, 0, 0)$ and $(0, 1, 0, 0)$ on the map containing cost estimates.



(a) Minimal distance path (b) Least risk path (c) Least effort path

Figure 3.33: Path planning using learned cost estimates

Figure 3.33 shows the resulting paths (marked white) of the three planning requests. As can be seen, both the minimal risk and effort paths differ significantly from the one that minimizes metrical distance. The minimal risk path is rather long and avoids all (positive and negative) obstacles in the map by a safety margin. In contrast, the minimal effort path is much shorter and exploits the steep downhill slope in the lower left part of the scenario to conserve energy. However, it comes close to the tree in this area, skirts closely around the central stone block and also takes an apparently more risky connection to

cross the embedded walkway in the upper scenario part as the previous path. Overall, this example demonstrates that the proposed cost learning strategy has really enabled the robot to take both the occurring obstacles and energetically relevant terrain features into account, without performing any direct sensor data interpretation. The gained cost information can indeed be used to improve path planning and compute paths that are advantageous compared to purely distance optimal solutions.

3.7.3.2 Real World

A final experiment has been conducted to evaluate the effect of cost learning on path planning in a large real world scenario. Analogous to the last simulation experiment, three different paths between the same start and end node have been planned using different cost metrics. However, in order to decrease the amount of time needed for edge cost learning, the initial map layout was substantially smaller than the regular grid structure used for simulation. Figure 3.34 shows a panoramic image of the test area, overlaid with the topological map that was provided to the robot prior to cost learning. To obtain the path with shortest metrical distance, the path planner was then commanded to plan a path from node 13 to node 7 using the unannotated, initial map. The result is marked in the figure with thick, white arrows. Then, the robot was issued several dozens of random edge traversal commands in order to build up the cost estimates.

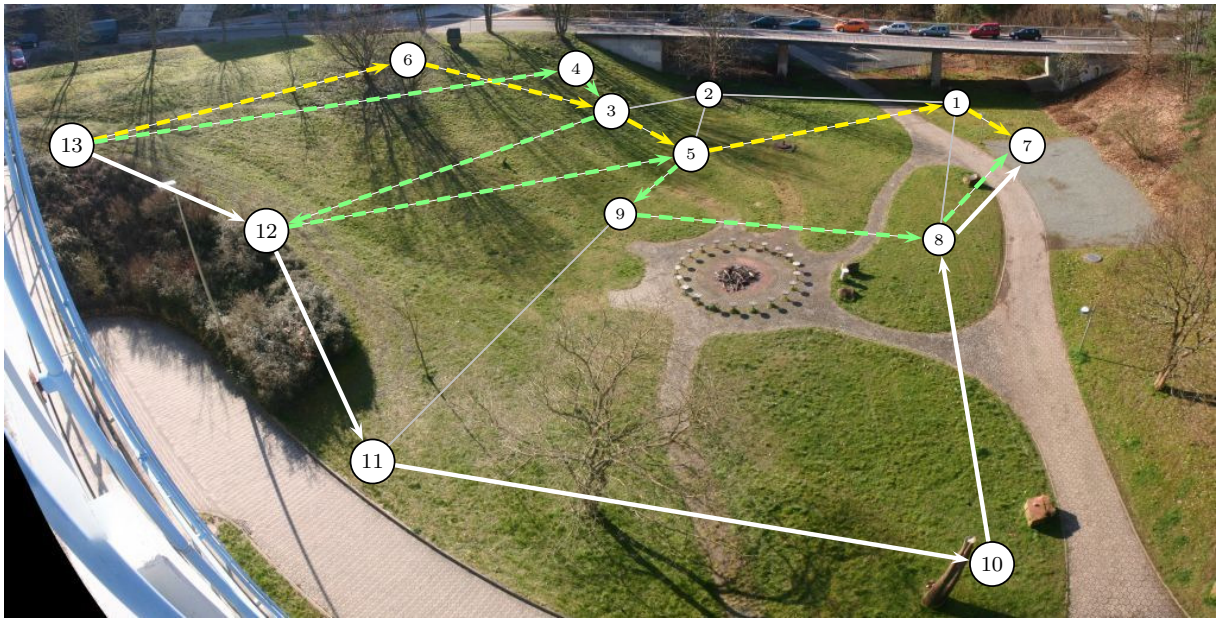


Figure 3.34: Fish-eye view of testing area and overlaid topological map
The testing ground covers approximately 100 by 100 meters and exhibits a maximum height difference of about 7 meters. The steepest part of the testing grounds (slope more than 20°) is located around node 3, while the most problematic obstacle configurations are below the bridge around nodes 1 and 2.

After cost learning, the path planner has been requested to generate the connection between 13 and 7 that minimizes either the risk or effort cost sum. The resulting paths are indicated in the picture with dashed green (minimal risk) and dashed yellow (minimal effort) edges. The green path is the result of tuning the cost function towards an extremely 'fearful' state with $(\alpha, \beta, \gamma, \delta) = (1, 0, 0, 0)$, while the yellow, dashed path is based on maximal 'impatience' with $(\alpha, \beta, \gamma, \delta) = (0, 1, 0, 0)$. Both differ substantially from the purely distance-based path. The yellow path saves energy by exploiting the steepest

slopes around node 3 and using a relatively direct connection. This comes at the price of traversing difficult terrain around node 1. In contrast to this, the green path contains a lot of lengthy detours in order to avoid this area and the vicinity of the hedge $(11 - 13)$. Both paths are intuitively plausible in the context set by the motivational state.

3.8 Conclusion

This chapter introduced a topological map variant containing nodes and edges with type information as fundamental data structure for large scale navigation in rough outdoor terrain. For this map, algorithms have been developed to effect movement along a series of map edges with minimal swaying, the robust detection of successful and unsuccessful arrivals at target nodes and techniques for map consolidation once speculative information is replaced by real experience. Furthermore, a three-dimensional cost measure suitable to record the cost factors that are most significant in outdoor terrain has been proposed. In order to estimate consistent edge traversal costs even though the topological navigation layer does not know the actual robot trajectory for a given edge in advance, a novel a posteriori learning scheme has been developed. This cost learning method observes the local piloting behaviors during motion and integrates the observations afterwards into scalar cost information. Finally, the quantitative and qualitative performance of the developed methods have been evaluated in a series of simulation and real-world experiments.

The obtained results document that the proposed method is capable of building consistent map edge costs starting from an initially unannotated topological map. The presented basic navigation capabilities enable the topological planner to generate paths that are cost optimal according to a motivational state set by the user in response to the current mission requirements. It also allows the system to refine the map during robot operation to optimize node positions, while guaranteeing a maximal node density. Untraversable connections are detected and excluded from further path planning.

With the presented capabilities, the system suffices for tasks that take place in an approximately known environment. In such scenarios, the operator can provide an initial topological map (possibly based on aerial imagery) which contains appropriate navigation-relevant nodes and viable path alternatives. The mobile robot can then start to carry out navigation tasks between nodes as required by the assigned missions, while simultaneously building up cost estimates for the topological edges and optimizing path selection over time. Possible areas of application of such a system include repetitive transportation tasks (e.g. on construction sites), patrolling of borders or large corporate estates or the continuous monitoring of environmentally relevant parameters in a large, vegetated environment.

However, the need for an initial map and the inability to extend the topological structure with new connections prevent the topological navigation system from being applicable for exploratory tasks such as autonomous reconnaissance or scouting missions. These limitations will be addressed in the next chapter.

4. Edge Cost Prediction and Map Extension

Up to now, the topological navigation system presented in the last chapter is limited to work within a given map and needs to traverse the available edges first (in order to learn cost estimates) before good path planning can be performed. These limitations do not allow the application of the navigation system for exploration and map building, or even a cost-efficient operation in environments where only a fraction of the edges has been explored and cost annotated.

In order to overcome these limitations, two questions must be answered:

- How to predict the traversal costs of speculative edges that have not been traversed yet?
- How (and where) to extend the topological map in order to reach a previously unreachable goal?

It turns out that both questions are closely related. Given the ability to estimate costs for new speculative edges correctly, map extension toward an up-to-now unreachable goal can be achieved by considering a set of possible additions of edges and nodes, estimating their costs, and finally choosing the alternative that results in the cheapest path. Thus, the remains of this chapter first treats possible solutions for the first question and then approaches map extension based on the obtained results.

4.1 Speculative Edge Cost Estimation

This section deals with speculative edge cost estimation, e.g. the task of predicting the travel costs for an edge that has not been traversed yet. As has been touched briefly in the last chapter, any potential approach to this question must be based on assumptions about the trajectory that will emerge once the navigator has commanded the edge transition. If no patterns or similarities between edges and resulting trajectories were present, the transfer of knowledge from other, already annotated links or the incorporation of metrical sensor data would not be possible.

Fortunately, experience shows that some assumptions about probable trajectories given an edge's start and end node are valid in most cases. For example, if the space between two nodes is mostly free of obstacles, the behavior-based pilot will travel along the line of sight because this is the shortest possible connection. Furthermore, the overall terrain layout often does not change abruptly, so that the experienced costs for two neighboring, approximately parallel edges are usually similar. Of course, these relations do not prevail in the presence of obstacles or other *local* terrain features that only affect some of the edges under consideration. In order to predict the correct cost in this case, it becomes necessary to obtain sensor data of the terrain which will likely be crossed during edge transition and perform a real traversability analysis. However, as the pilot's reaction to obstacles remains a 'black box' behavior and cannot be modeled well in advance, the cost estimates stemming from traversability analysis must be interpreted with caution. Their primary value lies in the capability to *qualitatively* favor edges across unblocked terrain over those that contain obstacles when considering different route alternatives.

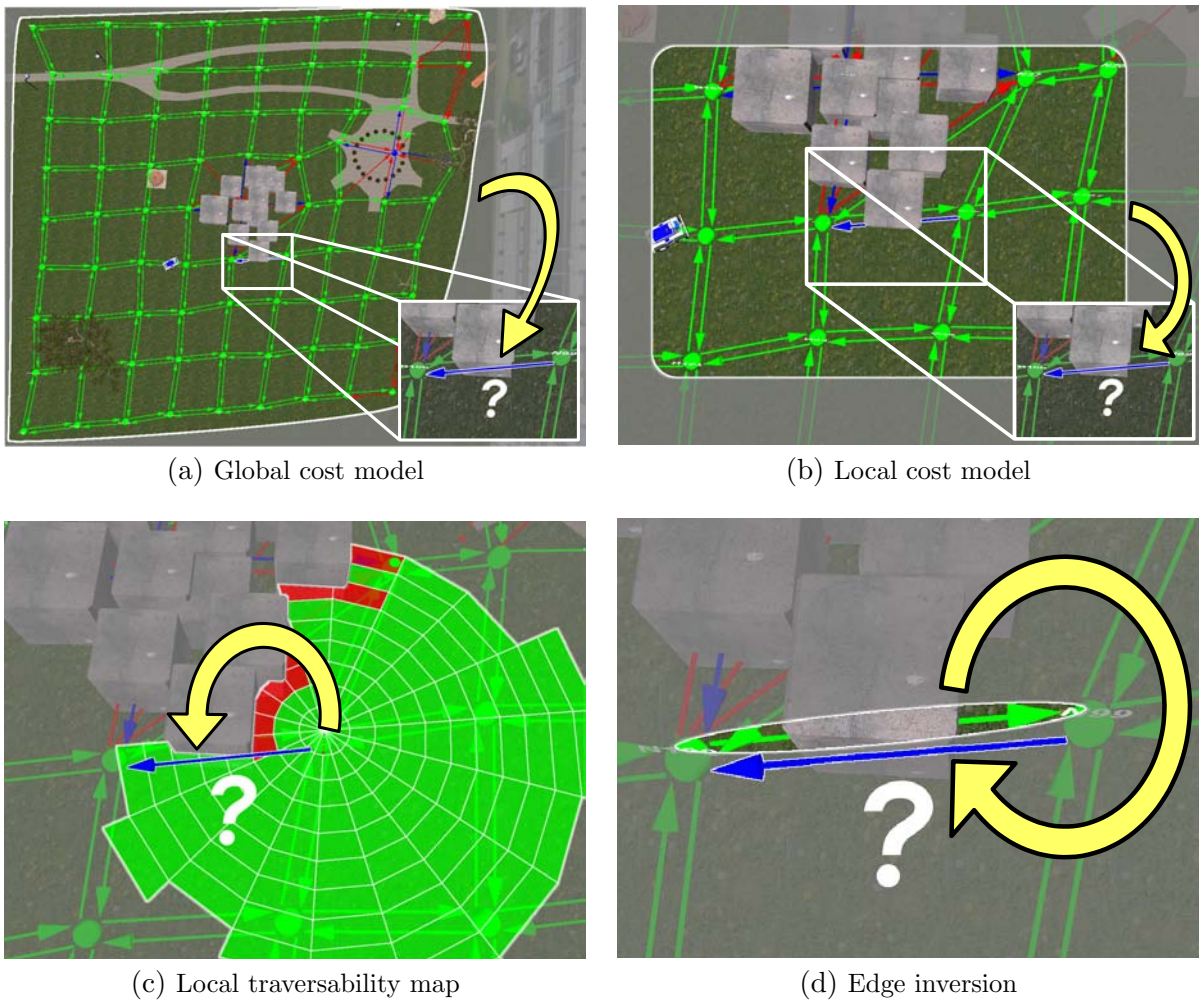
Four methods were developed in the scope of this thesis in order to predict speculative edge costs. All methods draw upon existing data stored in the topological map and extrapolate it to generate a cost estimate for the edge in question. However, the locality of the used data varies broadly, as shown in the overview in figure 4.1. While the most general estimation technique exploits all information stored in the entire map to generate a **global cost model** (figure 4.1a), the second approach builds a more restricted, **local cost model** using only cost annotations from spatially close edges (figure 4.1b). The trend toward increasingly local information continues with the remaining two techniques. The third method requires that the start node of the estimation candidate has been visited before and a metrical **traversability map** of the surrounding terrain was constructed (figure 4.1c). This traversability information is then incorporated into the cost estimate to account for local obstacle configurations that might have been missed by the global or local cost models. Finally, the **edge inversion** technique takes the known transition costs of the estimation candidate's inverse twin (running from its end to the start node) as the most definitive and local data source for cost extrapolation (figure 4.1d).

The rationale behind using not only one, but *four* different methods for speculative edge cost estimation is that all of these methods need different input data in order to work. As will be shown in the evaluation (sec. 4.1.5), the amount and locality of the required information correlates with the precision of the cost prediction. Thus, by choosing the most powerful method that is applicable given the available data, it is possible to use the best and spatially closest source of information within reach to estimate an edge in question.

4.1.1 Global Cost Model

In the worst case, the navigator needs to estimate the cost of an edge e that lies in an area which has not been explored by the robot *at all*. Consequently, no sensor data will be available for the adjacent terrain and edges in the vicinity will contain no cost annotations. Thus, the cost estimation for e cannot be based on any local source of information.

In such a situation, the only known helpful properties of e are the metrical positions of the edge's start and end node. From this, the line of sight distance $d_e = d(e.n.\vec{p}, e.n'.\vec{p})$ between the nodes can be computed. Now, *assuming* that the pilot will travel through



(a) Global cost model

(b) Local cost model

(c) Local traversability map

(d) Edge inversion

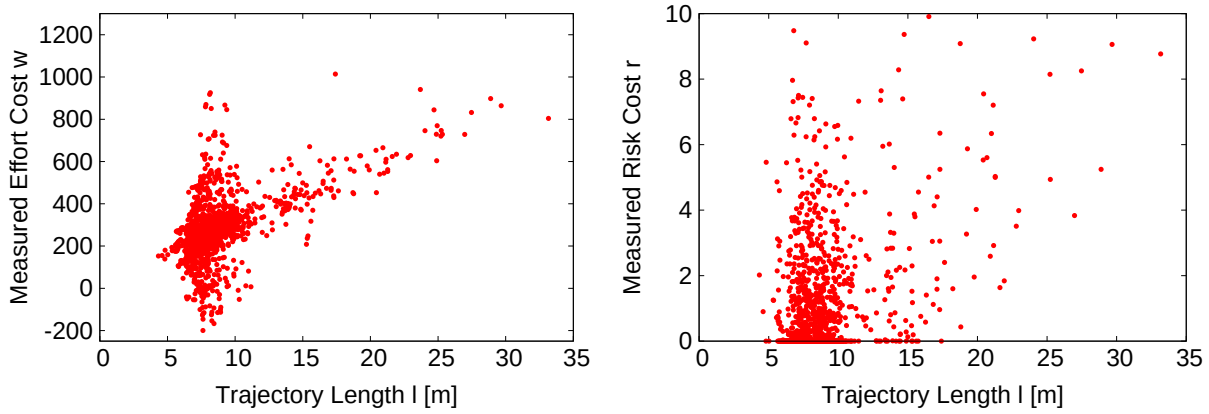
Figure 4.1: Four strategies developed for speculative edge cost estimation
All strategies extrapolate data from the topological map to predict costs for a speculative edge (blue with a white outline). However, the locality of the source data ranges from an entirely global perspective (figure 4.1a) to an extremely local scope based only on the inverse edge (figure 4.1d).

predominantly free space, the length of the emerging trajectory will be very similar to d_e. Furthermore, assuming that the pilot will 'behave normally' during transition, the resultant costs for e will be similar to those incurred by other trajectories of comparable length that have already emerged for edges in the rest of the map. Therefore, a very coarse estimate for e can be derived by establishing a global relation between trajectory length on the one side and probable risk and effort cost factors on the other side. As both the expended energy and the probability of encountering obstacles increases with a longer trajectory, at least some positive correlation of these entities should be observable.

To construct a plausible model of this relation during robot operation, the estimation algorithm needs to have access to measurements that link trajectory length with cost factors. Such measurements can be obtained by using the experience gained from previous edge traversals recorded in the topological map. For this, the edge annotation method presented in the last chapter has to be extended. In addition to storing risk and effort values for each new cost annotation, the length of the driven trajectory is recorded. This

length can be determined easily by integrating the differences between the robot poses $\vec{p}_r^{WCS}(t)$ while stepping through a behavior observation log (cf. section 3.7.1). With this extension, each edge traversal does not only produce a new cost annotation, but also generates a new sample for modeling the node distance vs. cost relation.

Figure 4.2 shows the results of applying this technique to all edge traversals performed during the experiment described in section 3.7.3.1. The two plots show the effort and risk values of all 1749 generated cost annotations plotted against their corresponding trajectory length.



(a) Effort versus trajectory length

(b) Risk versus trajectory length

Figure 4.2: Trajectory length / cost data pairs derived from edge annotations

The scatterplot in figure 4.2a gives some indication of a linear relationship between the learned effort values and the trajectory length, although the diagram also contains a significant amount of outliers. These can probably be attributed to the influence of different slopes or varying travel speeds caused by obstacles which are not accounted for in this model. The degree of linear correlation between the n measured pairs (l_i, w_i) of trajectory length l and effort w can be estimated mathematically by computing the **sample correlation coefficient** $r_{l,w}$ according to the formula

$$r_{l,w} = \frac{n \sum w_i l_i - \sum w_i \sum l_i}{\sqrt{n \sum w_i^2 - (\sum w_i)^2} \sqrt{n \sum l_i^2 - (\sum l_i)^2}} \tag{4.1}$$

where $\sum$ is a shorthand for $\sum_{i=1}^{n}$.

In general, the sample correlation coefficient between two variables can range from -1 to 1. A value of 0 indicates that the two variables are linearly independent, values of 1 or -1 are evidence for a perfect proportional or antiproportional relationship ([Bronstein 81], sec. 5.2.4). For the depicted data set of trajectory length and effort, the correlation coefficient $r_{l,w}$ is equal to 0.7829, which indicates a pronounced linear correlation between both variables. Therefore, it appears promising to fit a linear function to the obtained samples in order to determine a relation between trajectory length and effort.

Unfortunately, the plot showing risk and traveled distance in figure 4.2b does not exhibit an equally apparent linear behavior. Instead, both variables appear to be largely uncorrelated to the human eye when considering the diagram. However, the correlation coefficient

$r_{l,r}$ of 0.6085 indicates that although the relationship is weaker than with the effort cost factor and does not appear in the scatterplot (probably due to saturation of the diagram with many closely spaced points), some linear correlation *does* also exists between risk and trajectory length values. The construction of a linear model still appears to hold some merit.

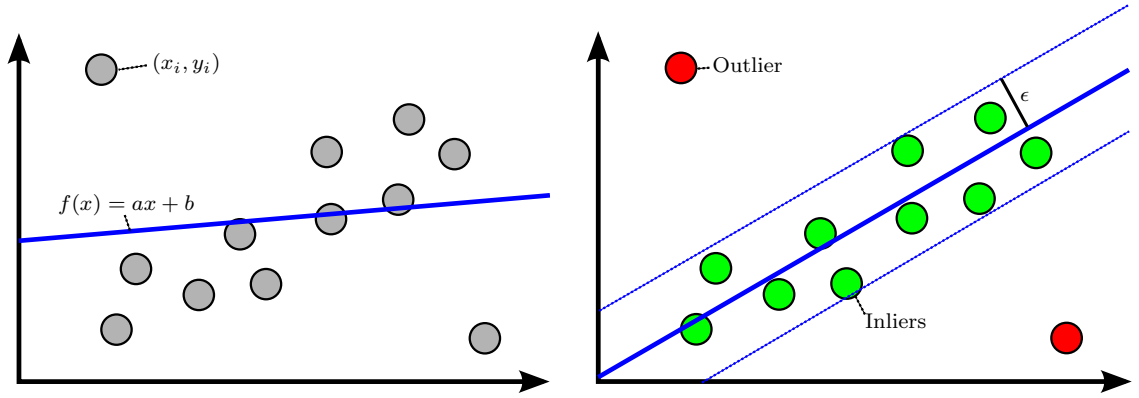
The coefficients of this linear model can be estimated using *linear regression*. For this, the function $f_w(l)$ which computes effort values from a trajectory length l is assumed to be of the standard form $f_w(l) = a_w l + b_w$. Given the data set with n pairs (l_i, w_i), the coefficients a_w and b_w can be estimated using the least-squares optimal solution given by

$$a_w = \frac{\sum_{i=1}^{n} \left(l_i - \bar{l}\right)\left(w_i - \overline{w}\right)}{\sum_{i=1}^{n} \left(l_i - \bar{l}\right)^2} \tag{4.2}$$

$$b_w = \overline{w} - a_w \bar{l} \tag{4.3}$$

with $\bar{l}$ denoting the mean length $\frac{\sum l_i}{n}$ and $\overline{w}$ the mean effort $\frac{\sum w_i}{n}$.

However, the standard least squares solution is susceptible to the influence of outliers, which can seriously distort the estimated function parameters. Figure 4.3a shows an example for this behavior. Here, two outliers that lie far away from the remaining, approximately diagonally distributed point set cause the least-squares optimal linear solution to have an incorrectly low slope. As a result, the linear regression models neither the linearly distributed points nor the outliers well.



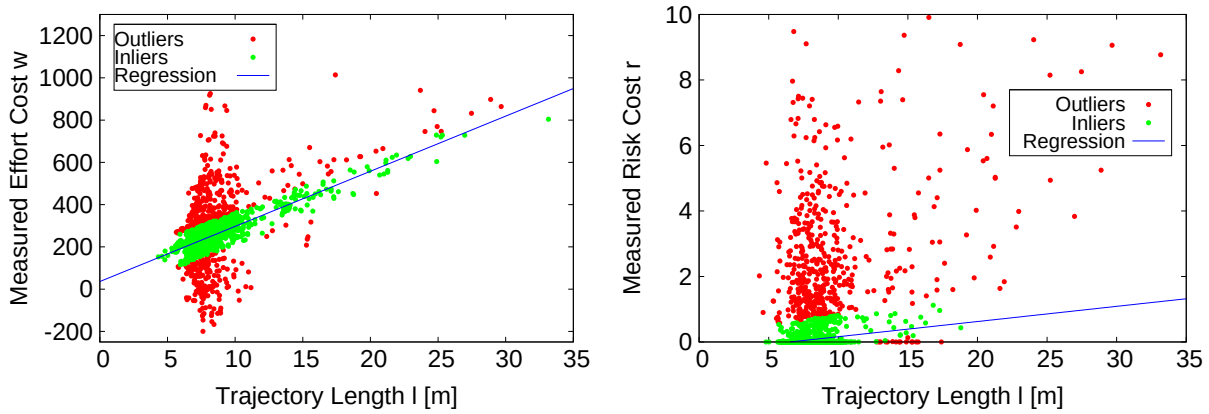
(a) Standard solution (b) RANSAC solution

Figure 4.3: Linear regression in the presence of outliers

In order to reduce the impact of outliers on the estimated linear function parameters, the use of the RANSAC algorithm is proposed [Fischler 87]. This approach proceeds by computing tentative values for the parameters a_w and b_w from two randomly picked pairs of the data set. Then, all data pairs with an euclidean distance smaller than a threshold ϵ to the line given by $a_w x + b_w$ are considered **inliers** and added to the inlier set R, all other points are classified as outliers. After several iterations of this process, the parameters supported by the *most inliers* are selected. Finally, a_w and b_w are recomputed according

to equations 4.2 and 4.3 using only the inlier set R as data source. See figure 4.3b for an example of the typical result obtained after using RANSAC on the toy data set from figure 4.3a.

The application of the RANSAC regression to the collected effort and risk samples from figure 4.2 results in the linear functions depicted in figure 4.4, with parameters listed in table 4.1.



(a) Effort versus trajectory length (b) Risk versus trajectory length

Figure 4.4: Linear trajectory length / cost models

Cost Factor	Function	a	b
Risk	$f_r(l) = a_r l + b_r$	0.08	−0.56
Effort	$f_w(l) = a_w l + b_w$	27.98	28.18

Table 4.1: Estimated parameters for global cost model

As can been seen, both estimated model slopes are positive, agreeing with the intuition that longer travels are more costly. The negative offset for the risk model is more interesting. It is a result of the fact that the source data does not contain any trajectory that is shorter than approx. 5 meters (the spacing of the initial grid used in simulation), and many of these have been annotated with a risk of 0. Thus, the linear model does indeed produce the most accurate prediction for the existing data if it crosses the x-axis right at an input value of around 5 meters. This behavior is not harmful and can be neglected as long as the navigator does not need to estimate any edge that is shorter than the shortest edges in the model's training set. In such a case, the resulting negative risk prediction must be capped to not confuse the path planning algorithm. However, after a few successful traversals of such short edges, the regression model will adapt itself automatically.

Coming back to speculative edge cost estimation, the constructed global model could now be used directly to generate a prediction for the speculative edge e by plugging the node distance $d_e = d(e.n.\vec{p},\, e.n'.\vec{p})$ between e's start and end node into the linear models, yielding the estimated costs $(f_r(d_e), f_w(d_e))$. However, this assumes that the trajectory length is equal to the node distance, which is unrealistic for most situations. Luckily, an improved connection between node distance d_e and resulting trajectory length l_e can be

established easily by using the presented outlier-robust regression technique again for this pair of variables[1]. The required data pairs (d_e, l_e) are contained in the topological map and the existing edge annotations, so that a linear function $f_l(d) = a_l d$ (an offset is not required in this case) can be constructed analogously to the computation of $f_w(l)$ and $f_r(l)$. Figure 4.5 shows the results of applying this idea to the simulation data.

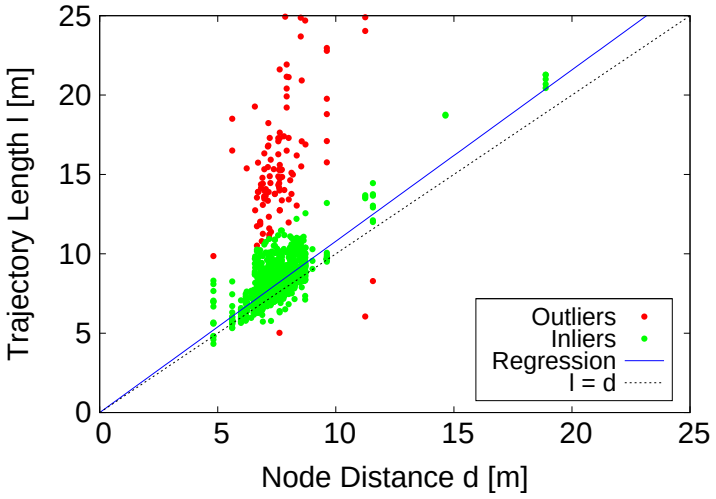


Figure 4.5: Linear trajectory length / node distance model

The optimal function has been estimated as $f_l(d) = 1.07d$, indicating that on average, the operation terrain is traversable and the emerging trajectories are relatively direct. However, the plot in figure 4.5 reveals that significantly longer trajectories do also exist, which have been classified as outliers and thus not incorporated into the model.

With the help of the regression model linking node distance and trajectory length, the 'global model' speculative edge cost estimation strategy can be finalized. Given a speculative edge e and the corresponding node distance d_e, the cost prediction for e using the global model heuristics is computed as $\vec{C} = (\mathbf{f_r}(\mathbf{f_l}(\mathbf{d_e})), \mathbf{f_w}(\mathbf{f_l}(\mathbf{d_e})), \mathbf{0})$.

4.1.2 Local Cost Model

The global cost model presented in the last section is extremely coarse, as it is based on the assumption that any map edge with similar node distance as the estimation candidate will also have similar costs. The approach suffers from two distinct disadvantages. For one, the sole use of distance values as input information cannot account for any local peculiarities such as obstacles at all. For another, it implicitly assumes that the terrain is uniform over the entire operation area. Therefore, the global model can also not cope with variations having a spatially moderate extent such as a grassy meadow containing some paths or open spaces (where travel is less risky than in the vegetated areas), or hilly terrain with smoothly changing effort costs. This lack of precision is set off by the fact that the model does not depend upon any 'special' input data and can be computed as soon as the map contains *any* cost annotations.

[1]It would also be possible to compute the relation between *node distance* and costs directly, instead of the presented approach which splits the process into two steps. However, it was decided to stick to the two-step process, as the models constructed in this fashion are more flexible and can also be used for other tasks besides speculative edge cost estimation.

With the global cost model in place as a fallback mechanism, it is possible to devise improved cost estimation methods for cases that meet certain requirements on the information available in the topological map. For example, the inability of the global model to cope with moderately large variations in terrain characteristics can be countered by constructing a **local model** that uses only cost annotations from edges that reside close to the speculative edge in question. This of course requires that nearby edges already *have* sufficiently many cost annotations and thus, that the robot has already visited that general area of the environment before.

In order to construct such a local cost model, it has to be decided which edge is 'close' enough to the estimation candidate e to supply cost annotations for the model construction. Since edges themselves do not provide metrical information, this decision is based on the positions of the edge start and end nodes instead. Given a parameter r that determines the size of the local neighborhood, two sets of nodes are constructed. The first set N is filled with all nodes that are within a target distance of r of e's start node n. Likewise, the second set N' includes all map nodes within range r of the end node n' of the estimation candidate e. Now, all edges e_i are selected for the construction of the local model which fulfill both the condition $e_i.n \in N$ and $e_i.n' \in N'$.

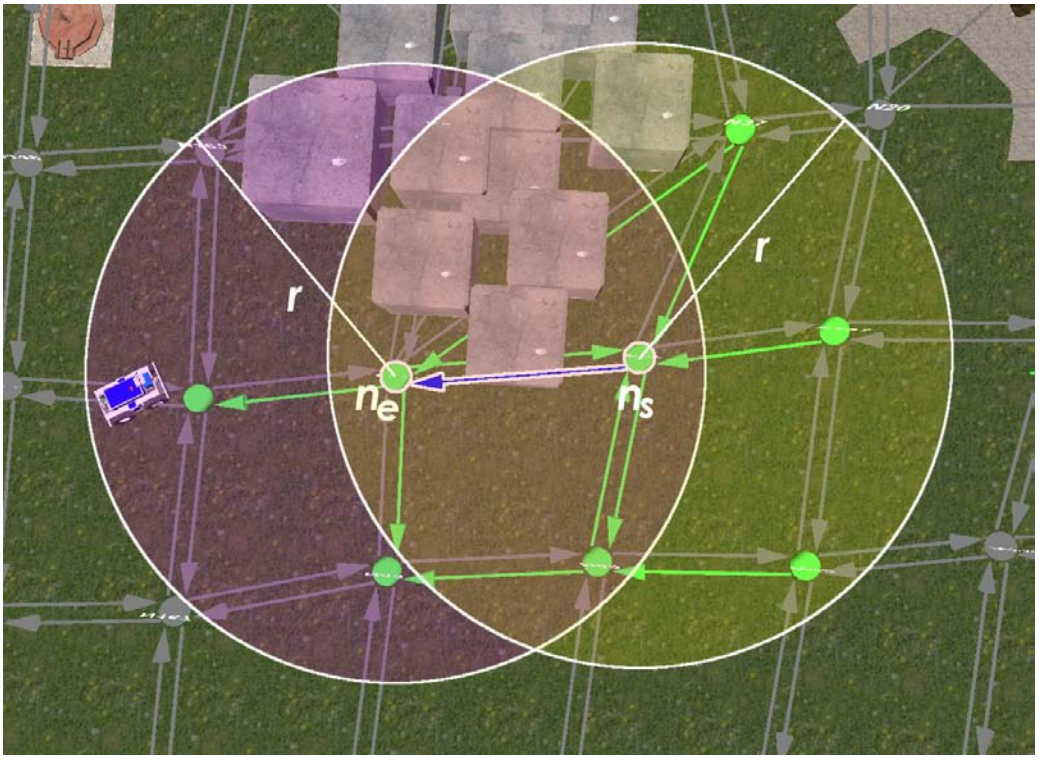


Figure 4.6: Eligible edges for local model construction

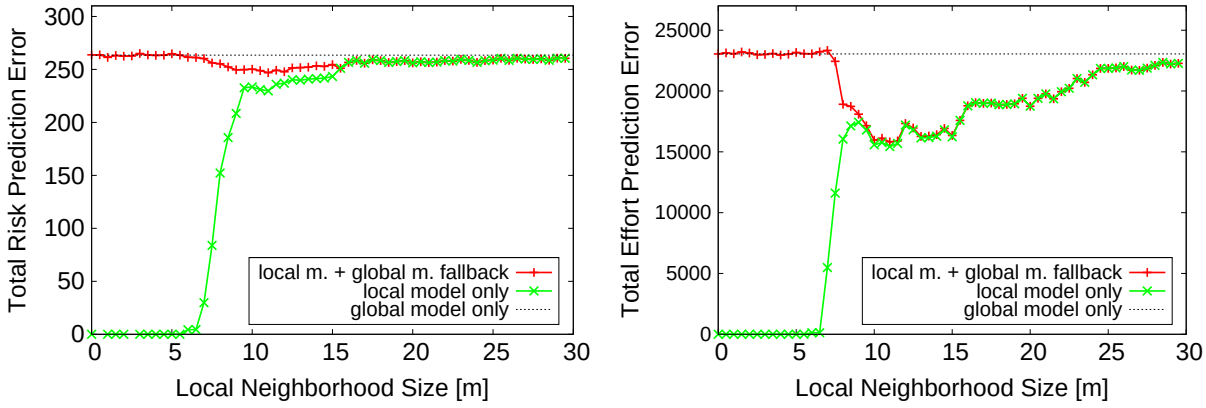
Figure 4.6 illustrates this selection process. The local neighborhood of the estimation candidate's start node n is highlighted in yellow, the surrounding of the end node n' is marked purple. To satisfy the demanded conditions for *both* edge nodes and be considered 'local', an edge must therefore start in the yellow and end in the purple area. All compliant executable edges in the figure are indicated in green, while all other edges have been grayed out.

In the next step, it is assumed that the local edges allow the construction of a stable local cost model whenever the sum of contained cost annotations exceeds a threshold Λ^{min}. In this case, linear regression models f_w, f_r and f_l are constructed just as described for the global cost model, but this time using only the annotations taken from the local edges. Finally, the transition costs for e are predicted as $\vec{C} = (f_r(f_l(d_e)), f_w(f_l(d_e)), 0)$ using the local regression models.

The last remaining question in the context of the local cost model construction is how to determine suitable values for the parameter r specifying the size of the local neighborhood and the threshold Λ^{min} specifying the minimal number of annotations. While it has been found empirically that useful values for Λ^{min} lie around $6-10$ in most cases, the selection of r is more critical. A good choice depends upon the variability of the target terrain and the overall edge density of the topological map. Thus, a truly optimal value for r can not be provided in advance.

However, it is at least possible to formulate a parameter selection method that determines good values for r given a sufficiently well annotated map to work with. Its core idea is to measure the quality of a given local model with neighborhood size r by computing the difference between the real costs of an executable map edge and the cost prediction that would be done by the model, if that edge was speculative. More precisely, the so called **leave-one-out error** for an executable edge is determined as the difference between the known edge cost vector $\vec{C}$ (for $\delta = 0$) calculated from the edge's annotation set Λ (see sec. 3.5.6) and the cost estimate obtained by temporarily setting $\Lambda = \emptyset$ and applying the local cost model prediction to that edge. After considering all edges in turn, the obtained errors are summed up into a total prediction error value. This process is repeated with different values of r and the one that corresponds to the minimal sum of the leave-one-out errors is selected as the optimum.

In the following, the proposed parameter selection method is applied to the simulation scenario data set to determine a suitable value for r. Figure 4.7 shows the total leave-one-out error sums for risk and effort cost predictions obtained using the local cost model with varying neighborhood sizes. Each graph contains three plot lines. For reference, the



(a) Risk prediction error (b) Effort prediction error

Figure 4.7: Local model cost prediction errors for varying neighborhood size

errors obtained using the global model are plotted in dashed black. The errors incurred

by using *only* the local model are marked by a green line. However, this graph must be interpreted carefully, as the local model requires sufficiently many nearby annotations and thus can not always be applied to all edges. Therefore, the total error of the local model is misleadingly low especially for small neighborhood radii, because the error is not summed over the entire set of edges. In order to compensate for this deficit, the third plot (solid red) shows the obtained errors from using the local model whenever applicable and falling back to the global model estimate otherwise. Therefore, as long as r is so small that *no* edge can be estimated using the local model, the red line coincides with the value for the global model. Then, as the local model starts to be usable for more and more edges (the distance between red and green lines drops), the total error sums fall below the global model level. This drop proves that a real benefit can be gained from using the local model cost estimation technique and also quantifies its size. At a value of $r = 16$ m, the local model can finally be computed for all edges in the topological map. Consequently, the global model fallback becomes unused and the green and red plots merge. A further increase of r causes the local model to become more and more 'global', and the error values continue to rise toward the global model level.

The plot indicates that the best choice for r in the given scenario lies between 10 to 14 meters. For $r = 11$ m, the total risk prediction error using the combined local/global cost model is 93% of the global model error, while the effort prediction error is reduced to only 68%. It is not surprising that the benefits of the local model technique are more pronounced for the effort cost factor, because the simulated terrain does contain 'effort variations' of a size comparable with r (the slope near the bridge), but no similarily scaled risk variations. The obstacles in the terrain are too small to be modeled accurately even using the local model. Nevertheless, the leave-one-out evaluation proves that the presented local regression technique can indeed generate more accurate cost predictions in environments with cost variations of medium spatial extent.

4.1.3 Local Traversability Maps

Both the global and the local cost models can only use knowledge contained in already existing edge annotations. Therefore, it is not possible with these techniques to react to local anomalies that do only influence the costs of the speculative edge under question. However, such a capability would be very helpful in many typical situations, for example when obstacles obstruct the path between start and end nodes of the estimation candidate.

Unfortunately, a cost estimation strategy capable of dealing with local anomalies cannot draw upon *any* data already contained in the current topological map. Instead, a new type of information is required which links the terrain beneath a speculative edge with a traversability measure describing how well that part of the environment can be passed. In other words, a metrical **traversability map** of the surrounding terrain is needed. Remembering the discussion of different map types from section 3.1, the use of a *global* map such as a digital elevation model encompassing the complete operation area would negate the positive properties of the topological map (such as compactness, efficiency or robustness against low localization accuracy). Furthermore, such a design decision would totally oppose the core concept of a minimal world model that underlies the whole navigation system.

Thus, an extension of the current topological map toward a hierarchical, hybrid structure similar to the approaches presented in section 3.1.3 is proposed instead. More precisely,

the idea is to store the information required for improved cost estimation in multiple *local* metrical maps, with each of them anchored at a topological node. The traversability maps are designed to remain as abstract and high-level as possible in order to maintain the current compactness and efficiency of the topological data structure. Therefore, no metrical sensor data or similar 'raw' information is stored, but only *risk / effort modificators* for the terrain surrounding the anchor node, since this data is directly relevant for cost computation. For ease of interpretation, the modificators are stored as normalized scalar values ranging from 0 for the best traversable terrain such as flat, open space up to 1 for maximally steep, cluttered areas.

In the following, the layout of the metrical traversability map is presented in detail, along with a description of how cost modificators can be retrieved from the piloting layer's local obstacle memory. Finally, the process of using the new information for speculative cost estimation is described.

4.1.3.1 Map Layout

Most established systems that use hybrid maps choose a standard grid layout for the local, metrical component. While this segmentation models the environment with constant resolution, the sensor coverage of grid cells inevitably degrades significantly with increasing distance to the robot due to the characteristics of the employed sensor systems such as laser scanners or cameras. This effect is usually tolerable because most systems are designed for indoor operation and the local maps only need to represent spatially restricted spaces such as rooms with sizes up to several meters. Additionally, these systems often use a SLAM based map building procedure to construct the local maps and synthesize data from different viewpoints until the map accuracy becomes uniform.

However, the grid layout is not an optimal choice for the envisioned application in large-scale off-road terrain. For one thing, the abstracted information that is to be stored in the local traversability maps does not support SLAM well. Thus, the map must be constructible using a single sensor snapshot, preferably taken from the central node position. For another thing, the environment to be modeled is much larger, and visibility of grid cells far from the robot is likely to be very poor. Therefore, a different map layout is proposed, which partitions space into **sectors** with constant angular extent. Each sector is further subdivided into **seclets** (or patches) of constant metrical length. The resulting map somewhat resembles a spider web and is illustrated in figure 4.8. This segmentation reduces the impact of larger distances on the sensor coverage, because each seclet is covered by an image region with constant width (if cameras are used as data source) or a fixed number of range measurements (if laser scanners are used). Also, traversability in a given direction can later be computed easily by sequential inspection of all seclets in the corresponding sector.

Each seclet stores scalar modifiers for risk and effort cost factors that characterize the underlying terrain patch. Additionally, it also keeps track of two certainty measures which express the confidence about the modifiers as estimated by the generating algorithm. These certainty values can be used well to model partial knowledge of the terrain. By setting a confidence of 0 for a seclet, missing information about that area is indicated. Definition 4.1 formalizes the components that constitute a local traversability map.

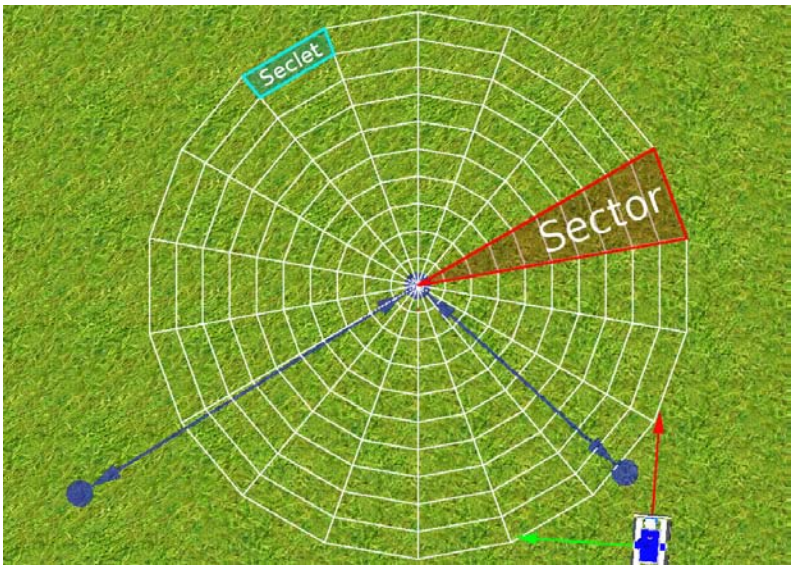


Figure 4.8: General layout of a local traversability map

Definition 4.1 (Local Traversability Map) *Given an angular sector size ϕ, a seclet width l and a number of seclets per sector L, a* ***local traversability map*** *M is defined as a list of $i = 2\pi/\phi$* ***sectors*** *$S_0 \dots S_{i-1}$, with each sector being a list of L* ***seclets*** *$s_0 \dots s_{L-1}$. Each seclet contains*

- *a pair of cost modifiers (r, w) with $0 \leq r \leq 1$, $0 \leq w \leq 1$ for risk and effort cost factors*
- *a pair of confidence values (θ_r, θ_w) with $0 \leq \theta_r \leq 1$, $0 \leq \theta_w \leq 1$ that expresses the reliability of r and w, respectively.*

To attach the local traversability map to a topological node n, the map origin is set to the node position by creating a node local coordinate system NCS at $n.\vec{p}$. The transformation M_{NCS}^{ECS} to the earth coordinate system is computed using the SetupCSWithZAxis function introduced in section 2.3.3 and executing $M_{NCS}^{ECS} = \mathsf{SetupCSWithZAxis}(n.\vec{p},\ (0,0,1),\ n.\vec{p})$ with the NCS origin position $n.\vec{p}$. Using M_{NCS}^{ECS}, the indices S and s of the sector respectively the seclet corresponding to a ECS position $\vec{p}$ in n's local traversability map is determined by:

$$\vec{p}^{\,NCS} = (M_{NCS}^{ECS})^{-1} \cdot \vec{p} \tag{4.4}$$

$$d = \sqrt{(\vec{p}_x^{\,NCS})^2 + (\vec{p}_y^{\,NCS})^2} \tag{4.5}$$

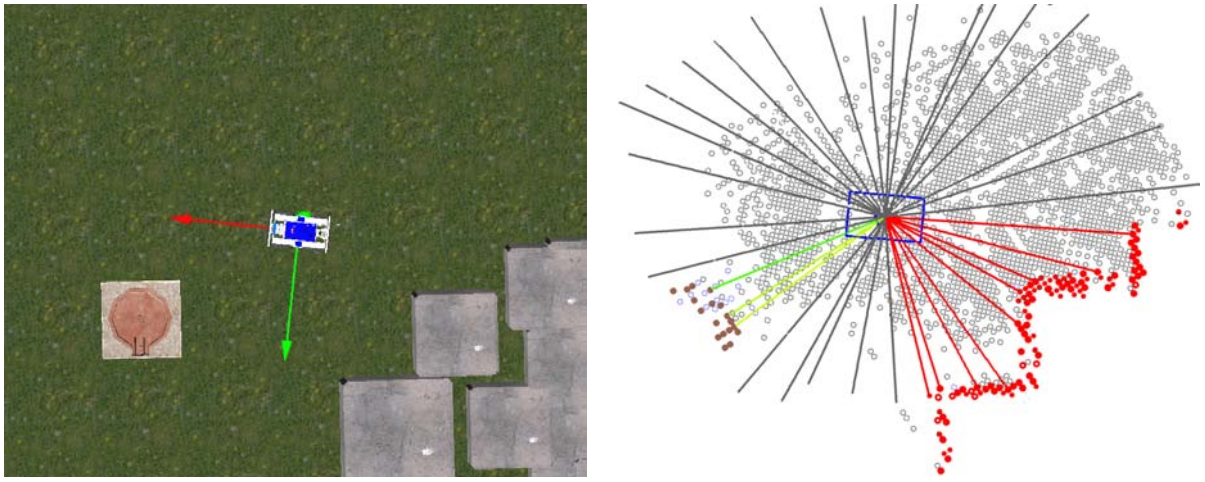
$$\gamma = \arctan \frac{\vec{p}_y^{\,NCS}}{\vec{p}_x^{\,NCS}} \tag{4.6}$$

$$S = \left\lfloor \frac{\gamma}{2\pi/\phi} \right\rfloor \tag{4.7}$$

$$s = \left\lfloor \frac{d}{l} \right\rfloor \tag{4.8}$$

4.1.3.2 Including the Local Obstacle Memory

In order to fill the local traversability maps with cost modificators, the navigator can either use the robot's available sensors and perform a traversability analysis of the terrain surrounding the anchor node by itself *or* it can reuse the data interpretation that has already been done by the pilot. The second approach seems to be simpler and more effective at this stage, because the pilot maintains a local memory which can be conveniently tapped to provide the required information about obstacles and passable areas. Figure 4.9 shows an example of this data.



(a) Scene (b) Local obstacle memory

Figure 4.9: Local obstacle memory of the pilot

Figure 4.9a shows the original scene and the corresponding memory that has been constructed by the pilot using a turnable range finder (see [Hach 07] for details) is depicted in figure 4.9b. As can been seen, the map is centered at the robot (the blue rectangle) and contains sample points indicating free space (empty black circles) as well as representations for obstacles of varying severity (filled brown and red dots).

The pilot also performs an abstraction from the raw metrical data in order to feed the obstacle avoidance behaviors with appropriate information. More precisely, it splits the map into sectors similar to those used in the navigator's local traversability map, extracts the closest obstacle for each sector and stores its position as a **representative**. In figure 4.9b, these representatives are visualized using red (severe) or yellow/green (less critical obstacles) lines connecting the robot with the obstacle position. If no obstacle is found in a sector, the longest range for which sensor data was available is stored instead (shown in gray).

These representatives form a compact basis to fill the local traversability map with the required cost modificators. First, the robot must approach the topological node whose map is to be updated. Then, the data from the pilot's memory is transferred. For this, the robot pose $\vec{p}$ and the positions $\vec{r_i}$ of each obstacle representative known to the pilot are transformed into the node coordinate system. The line connecting the $\vec{p}^{\,\mathrm{NCS}}$ with $\vec{r_i}^{\,\mathrm{NCS}}$ is sampled with a step size $< l$, and the seclet corresponding to the sample position is determined using equation 4.8. The sample positions trace the path to the nearest

obstacle for each sector. Since the pilot has determined that these areas are *free*, the calculated seclet's cost modifiers are set to $(r, w) = (0, 0)$ to indicate optimal traversability. Furthermore, as the pilot is expected to use the most recent and close-ranged information available, the confidence values (θ_r, θ_w) are maximized to $(1, 1)$. After the sampling has been done for all sectors, the seclets at the obstacle representative positions $\vec{r_i}^{\,\mathrm{NCS}}$ itself are treated. Their cost modifiers are set depending on the type of obstacle representative that the pilot has detected there. Four different cases are distinguished, as shown in table 4.2.

Obstacle Type	Cost Modifier	Description
None	$(0, 0)$	sensor range exceeded (gray lines)
Rough Ground	$(0.25, 0.25)$	increased terrain roughness (green lines)
Vegetation	$(0.5, 0.5)$	*potentially* traversable using tactile creep (yellow lines)
Critical	$(1, 1)$	definitely impassable obstacles (red lines)

Table 4.2: Types of pilot obstacle representatives

After this, all relevant information stored in the local memory of the pilot has been inserted into the node's traversability map. Figure 4.10 shows the result.

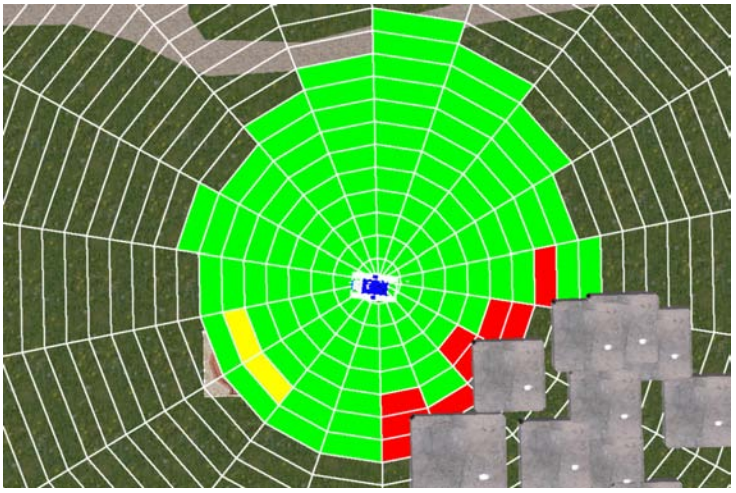


Figure 4.10: Resulting traversability map

In the picture, seclet color is used to indicate the value of the risk cost modifier, ranging from green for 0 to red for a value of 1. The opacity of the seclet corresponds to the confidence value - 0 yields a totally transparent patch (like those that were outside the range of the local obstacle memory), while a confidence of 1 makes the patch completely oblique. As can be seen, the positions of the obstacles in the local vicinity of the node have been recorded correctly as well as the presence of open space in other directions. This knowledge can now be utilized to improve the cost estimation for speculative edges.

4.1.3.3 Edge Cost Prediction using Local Traversability Maps

The prediction of edge costs using the information contained in a local traversability map must rely on relatively strong assumptions about the metrical trajectory that will emerge

upon edge transition. Since there is no better guess available, it is assumed that the robot will try to travel along the line of sight. Thus, all seclets that lie on the direct connection between the edges's starting point $n.\vec{p}$ at the map center and its end node $n'.\vec{p}$ will influence the cost estimate. Figure 4.11 shows a schematic drawing to demonstrate the arrangement of these seclets.

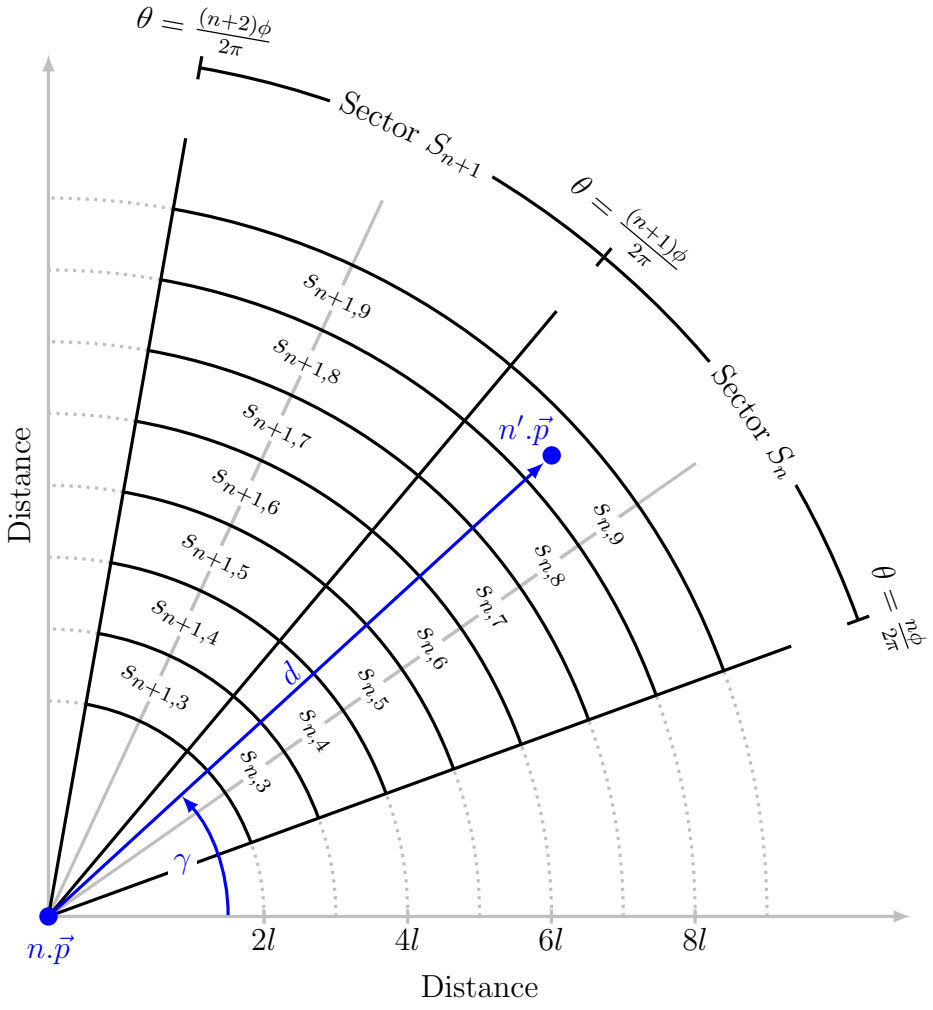


Figure 4.11: Seclets relevant for cost estimation

After projecting both start and end position of the edge under question into the start node's NCS, the angle γ and length d are computed according to equations 4.5 and 4.6. Then, the two sectors S_n, S_m that lie closest to the line of sight connection of the two nodes are determined using

$$n = \left\lfloor \frac{\gamma}{2\pi/\phi} \right\rfloor \tag{4.9}$$

$$n_{res} = n - \frac{\gamma}{2\pi/\phi} \tag{4.10}$$

$$m = \begin{cases} n+1 & \text{if } n_{res} \geq 0.5 \\ n-1 & \text{otherwise} \end{cases} \tag{4.11}$$

The inclusion of two neighboring sectors allows to perform linear interpolation according to γ and therefore avoid abrupt cost changes for small angle disturbances when γ lies close to a sector boundary. The interpolation weight w_n of sector S_n is obtained by computing

$$w_n = \begin{cases} 1.5 - n_{res} & \text{if } n_{res} \geq 0.5 \\ 0.5 + n_{res} & \text{otherwise} \end{cases} \tag{4.12}$$

Assuming that a function ComputeSecletCosts exists which returns the risk and effort costs for a single seclet, the predicted cost for the complete edge can be calculated by adding the estimates of all relevant seclets together and doing a linear interpolation between sectors. The only partially crossed last seclet has to be handled separately, as is shown in algorithm 7.

Algorithm 7: EstimateEdgeCostFromLocalMap

Result: Risk, Effort Cost Prediciton (r, w)

Int $i_{max} \leftarrow \lfloor \frac{d}{l} \rfloor$
Float last_frac $\leftarrow (d - i_{max} l)\,/l$
Vector2D cost_n $\leftarrow 0$
Vector2D cost_m $\leftarrow 0$
// add seclet costs for sectors S_n and S_m
while i $< i_{max} - 1$ **do**
  cost_n $\leftarrow$ cost_n + `ComputeSecletCosts`$(s_{n,i})$
  cost_m $\leftarrow$ cost_m + `ComputeSecletCosts`$(s_{m,i})$
  i $\leftarrow$ i + 1
end
// account for partially covered last seclet separately
cost_n $\leftarrow$ cost_n + last_frac $\cdot$ `ComputeSecletCosts`$(s_{n,i_{max}})$
cost_m $\leftarrow$ cost_m + last_frac $\cdot$ `ComputeSecletCosts`$(s_{m,i_{max}})$
// linear interpolation between sectors
return $w_n \cdot$ cost_n $+ (1 - w_n) \cdot$ cost_m

How can the function ComputeSecletCosts be implemented? In other words, what is the cost of traversing a single seclet? The solution developed in this thesis is based on three predetermined parameter vectors which quantify the costs $\vec{c} = (c_r, c_w)$ when traversing

1. a definitely **free** patch: $\vec{c}^{\,free} = \left(c_r^{free}, c_w^{free}\right)$
2. a definitely **blocked** patch: $\vec{c}^{\,blocked} = \left(c_r^{blocked}, c_w^{blocked}\right)$
3. an **unknown** patch: $\vec{c}^{\,0} = (c_r^0, c_w^0)$

These three estimates are combined using bilinear interpolation to incorporate both the cost modifier values and the accompanying confidences stored in the seclet. The interpolation is done so that a confidence value of zero results in the costs $\vec{c}^{\,0}$ regardless of the (in this case totally unreliable) cost modifier values. As the confidence increases, the

resulting costs more and more approach a linear mixture of $\vec{c}^{\,free}$ and $\vec{c}^{\,blocked}$, weighted according to the values of the cost modifiers. Mathematically, this is expressed by

$$c_r = \theta_r(r \cdot c_r^{blocked} + (1-r) \cdot c_r^{free}) + (1-\theta_r) \cdot c_r^{0} \tag{4.13}$$

$$c_w = \theta_w(w \cdot c_w^{blocked} + (1-w) \cdot c_w^{free}) + (1-\theta_w) \cdot c_w^{0} \tag{4.14}$$

where (r, w) are the seclet cost modifiers and (θ_r, θ_w) are the corresponding confidences. This bilinear interpolation is graphically illustrated in figure 4.12.

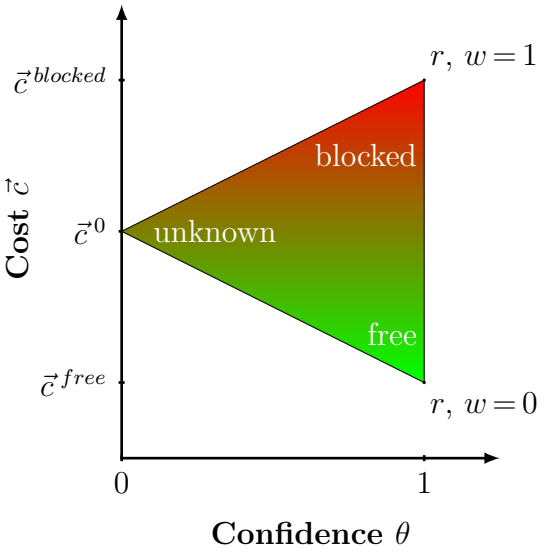


Figure 4.12: Bilinear interpolation of seclet costs

With the interpolation mechanism in place, the task of computing suitable traversal costs for a single seclet is reduced to the problem of choosing appropriate presets for the three parameters $\vec{c}^{\,free}$, $\vec{c}^{\,0}$ and $\vec{c}^{\,blocked}$.

In the case of a totally unknown patch, the seclet map does not provide any additional information. In this situation, it is sensible to fall back and rely on the best estimation methods that were available *before* the introduction of the local traversability map, which are the local and global cost models presented in the last two sections. With these, the costs of a seclet can be computed using the regression functions f_r, f_w and f_l fed with the metrical seclet width l, which leads to:

$$\vec{c}^{\,0} = (f_r(f_l(l)),\ f_w(f_l(l))) \tag{4.15}$$

This choice guarantees that a completely empty (all seclets have confidence 0) local traversability map produces the same cost estimates as if no map was present at all. Therefore, the cost estimate does not exhibit an eventually problematic unsteadiness upon switching from one prediction model to the other.

The remaining two parameters are more difficult to determine. It is intuitively plausible to assign a very low cost to $\vec{c}^{\,free}$ such as $(0, f_w(l))$ since the robot is expected to travel through open space when $\vec{c}^{\,free}$ becomes dominant. However, experimental validation using the leave-one-out error measure described in section 4.1.2 has given unsatisfactory results for such a parameter setting. In fact, the total risk error values using this choice

rise approximately 30% above the results obtained without the local traversability map. It therefore appears that the pilot's detection of free space is not very well linked with a zero risk observation. A probable cause for this is that the pilot's sensor interpretation algorithms can detect some obstacle types (such as negative obstacles like holes or steps) only within a very close range. These obstacles are consequently not always included in the local obstacle memory, but *do* cause risk accumulation every time real edge transitions occur. This discrepancy may explain the detrimental effect of an optimistic value assignment for $\vec{c}^{\,free}$. As a somewhat unappealing solution, the value of $\vec{c}^{\,0}$ can be used instead, implying that free space may contain (still undetected) obstacles just as unknown terrain. This leads to the following assignment of the second seclet cost parameter:

$$\vec{c}^{\,free} = \vec{c}^{\,0} \tag{4.16}$$

A better course of action would obviously be to improve the obstacle detection algorithms themselves in order to maximize the usefulness of the local traversability maps. However, the detection of negative obstacles is exceptionally difficult over longer distances, so that substantial improvements are unlikely to be attained with reasonable effort. It appears more promising to improve *long range* traversability estimation instead in order to raise the effectiveness of this cost estimation strategy. This topic will be addressed in the next chapters. Therefore, the proposed setting for $\vec{c}^{\,free}$ is retained for now.

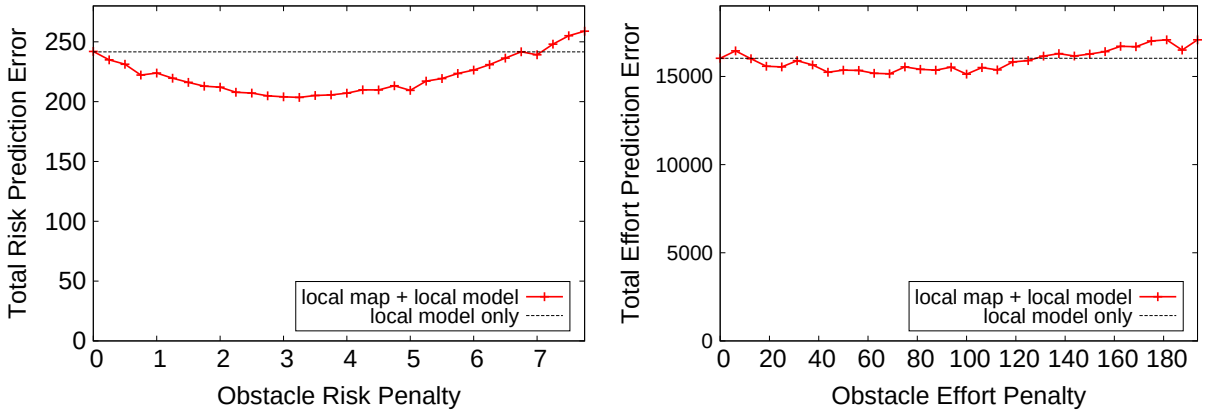
The selection of the final parameter $\vec{c}^{\,blocked}$ is complicated by the black box behavior of the piloting layer. Because no sophisticated model of the reactions to an encountered obstacle is available, the total costs for a trajectory that contains a blocked seclet cannot be predicted with high accuracy. Instead, it is simply assumed that a constant *penalty term* is sufficient to summarize the average additional risk and effort costs induced by the obstacle. Therefore, $\vec{c}^{\,blocked}$ is set to the cost for an unknown seclet plus a penalty (ρ_r, ρ_w), resulting in the definition:

$$\vec{c}^{\,blocked} = \vec{c}^{\,0} + (\rho_r, \rho_w) \tag{4.17}$$

To determine the best value for this penalty, the already established leave-one-out error optimization technique (see sec. 4.1.2) is applied. The total error that is suffered when predicting edge traversal costs based on local traversability maps is evaluated using varying values for ρ_r and ρ_w. Figure 4.13 shows the obtained results.

The figure reveals that the best predictions are obtained using a value of about $\rho_r = 3.25$ as the risk penalty and a value of $\rho_w = 60$ for the effort penalty. This completes the selection of the seclet cost parameters and concludes the description of the map based edge cost estimation technique.

Figure 4.13 also provides quantitative data on how much better the cost prediction based on local traversability maps is in comparison to the previously best estimate achieved by the local model (included in the figure for reference). With optimal parameter settings, the total risk error is only **84%** of the local model error. The maximal error reduction is not as significant for the effort cost factor, leaving an error rate of **94%** there. It can be noticed that the magnitudes of the relative error reductions are reversed in comparison to the introduction of the local model. Apparently, the local maps introduced in section 4.1.3 are better suited to treat rather small disturbances which primarily influence risk costs on a very local scale. In contrast, the local model technique from section 4.1.2 is



(a) Risk prediction error (b) Effort prediction error

Figure 4.13: Prediction errors for varying seclet penalties

better suited for more global terrain characteristics which have a greater impact on the observed effort. Thus, both cost prediction techniques exhibit complementary strengths which can be combined to obtain an improved cost estimate.

4.1.4 Cost Transfer from Inverse Edges

The last strategy that has been developed for the task of speculative edge cost prediction tries to relax the strong assumption made by the local map technique which expects the pilot to travel *directly* between start and end nodes without taking significant detours. In order to do so, the new strategy relies on the estimation candidate's **inverse twin** as the most local source of information available for a speculative edge. The inverse twin of a given edge e connecting nodes n and n' is defined as an edge that runs in the opposite direction between the same nodes, thus starting from n' and ending at n. Figure 4.14 shows an example.

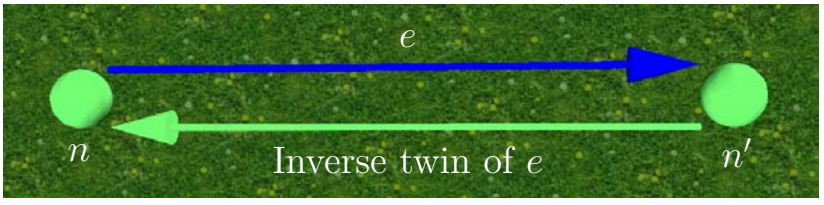


Figure 4.14: An edge and its inverse twin

As with the other strategies, an assumption must be made to justify the transfer of costs from the inverse twin to the estimation candidate. It is assumed that the trajectories which will emerge upon traversal of the estimation candidate will have a similar shape as those which led to the inverse edge annotations (of course, the travel direction will be reversed). However, if this belief is approximately correct, the cost transfer will allow accurate cost predictions even if the trajectories contain detours, loops or other anomalies that violate the inherent assumptions of the other cost estimation methods. On the negative side, this strategy can only be applied if the edge that is to be estimated really *has* an inverse twin which is already annotated.

The cost transfer itself can be implemented as follows. For one, assuming the pilot generates similar trajectories for both edges, it can be anticipated to encounter the same obstacle constellations during travel. Therefore, the risk prediction for the speculative edge can be adopted directly from the learned risk of the inverse twin, e.g. its combined risk annotation $\widehat{R}$ (see section 3.5.6).

A good prediction of the effort cost factor is more complicated, since equal effort costs can only be expected in flat terrain. For vegetated outdoor environments which are typically sloped, a simple value copy would not yield satisfactory results. To overcome this problem, the developed prediction algorithm first builds a model of how effort costs depend upon the terrain slope. It then takes the inverse edge effort cost $\widehat{W}$ and average trajectory slope α and uses the model to extrapolate this data onto the estimation candidate. The extrapolation part thereby assumes that the estimation candidate trajectories will have a slope of $-\alpha$.

In order to implement this strategy, the cost estimation algorithm first needs to build the effort / slope model. In order to accomplish this, the cost annotation method from section 3.7.1 is extended once more to record the slope α_i of each performed edge transition and include it in the corresponding cost annotation. α_i can be derived from the behavior log obtained after an edge traversal using the difference in the WCS Z-coordinate of the robot's start and end poses $\vec{p}_r^{\,WCS}(t_0)$ and $\vec{p}_r^{\,WCS}(t_1)$ and the trajectory length l that is already computed for the global cost model. With this information, the calculation of α_i is feasible using the equation:

$$\alpha_i = \arctan \frac{\vec{p}_r^{\,WCS}(t_1).z - \vec{p}_r^{\,WCS}(t_0).z}{l} \tag{4.18}$$

It is important to stress that this computation is not substantially affected by the drift of the height estimates that has been identified as a problem in the context of node arrival detection (see section 3.4.2). The main reason behind this is that the time between passing the start position and arriving at the end pose is typically rather short. In this time frame, the *difference* between the two WCS height estimates usually remains consistent with the real height difference, although the absolute values might indeed be inaccurate.

Each extended cost annotation now contains information about the experienced trajectory slope α_i, length l_i and effort cost w_i. On the basis of this data, a linear regression model can be constructed using the RANSAC algorithm introduced in section 4.1.1 to link a slope s with the incurred effort *per meter* w/l. The use of this fraction factors the influence of varying distances out of the model, which could otherwise only be accounted for by a more complicated two-dimensional function.

Figure 4.15 shows the data set collected for the simulation scenario together with the estimated regression function $f_{w/l}(s) = 199.9 \cdot s + 31.5$. The function $f_{w/l}(s)$ could be used directly to generate an effort prediction for the speculative edge by taking the inverse twin's average trajectory length l and slope α to compute $l \cdot f_{w/l}(-\alpha)$. However, this totally neglects the inverse edge's effort cost annotation $\widehat{W}$, which can contain valuable information about any particularities encountered along the driven trajectory that has been smoothened out by the linear regression function. To include $\widehat{W}$ into the prediction, $f_{w/l}(s)$ is used indirectly. The effort prediction is produced by *mirroring* the inverse twin's costs $(\alpha, \widehat{W})$ on the point $\big(0, l \cdot f_{w/l}(0)\big)$ which signifies the expected effort for a trajectory

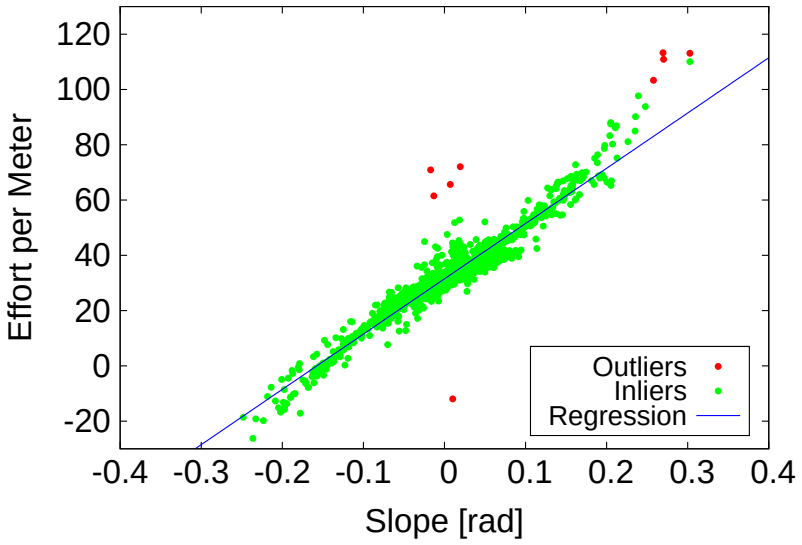


Figure 4.15: Linear effort per meter / terrain slope model

with the same length as before but zero slope. This yields an cost estimate for the slope $-\alpha$ that differs as much from the flat terrain case as the inverse twin's original cost, but deviates in the opposite direction, agreeing with the assumption that the estimation candidate has a slope of α.

Mathematically, the estimated effort costs c_w for the speculative edge is computed from l and α using $f_{w/l}(s)$ by

$$c_w = \widehat{W} + \left(l \cdot f_{w/l}(0) - \widehat{W}\right) + \left(l \cdot f_{w/l}(0) - \widehat{W}\right) \tag{4.19}$$

$$= 2l \cdot f_{w/l}(0) - \widehat{W} \tag{4.20}$$

Figure 4.16 illustrates the derivation of this formula.

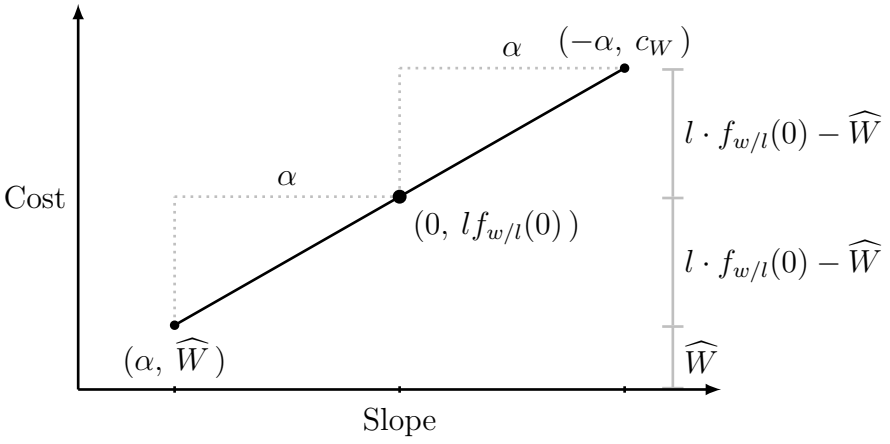


Figure 4.16: Calculation of inverse edge cost

4.1.5 Evaluation

To evaluate the performance of the four cost estimation strategies, their total and relative prediction errors have been determined using the leave-one-out technique introduced in

section 4.1.2 on the standard large simulation scenario. The resulting measures are listed in table 4.3.

Strategy	Risk Error		Effort Error	
	Absolute	Relative	Absolute	Relative
Global Model	263.5	100%	23061	100%
Local Model	242.0	91.8%	16234	71%
Local Maps	206.7	78.4%	15388	66.7%
Edge Inversion	187.7	71.2%	10639	46.1%

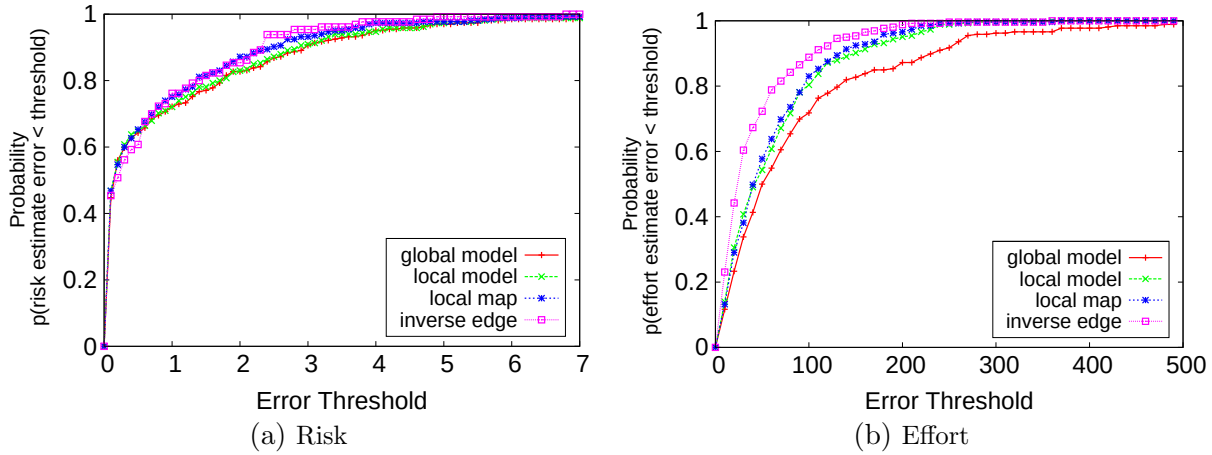
Table 4.3: Prediction errors of the presented cost estimation strategies

The shown data provides proof for the claim made at the beginning of this section, stating that the closer the source of cost information is to the speculative estimation candidate, the better the overall cost prediction. Based on this observation, the preference of speculative edge cost estimation strategies can be formulated as follows:

1. Use edge inversion, if an annotated inverse edge exists.
2. Use local traversability maps, if speculative edge's start node holds local map.
3. Use local cost model, if sufficient annotations are found in local neighborhood.
4. Use global cost model.

This order ensures that the available information is optimally exploited and the speculative edge cost estimation is done with the most accurate and powerful technique available.

Figure 4.17 shows the probability of generating a cost prediction within a given error bound to highlight the accuracy that can be expected for single cost predictions.



(a) Risk (b) Effort

Figure 4.17: Probability of cost prediction with bounded error

Considering the fact that the risk costs for the examined scenario typically range from 0 to 15, while effort variations are scaled between −200 and 1000, the prediction of accurate

risk costs appears to be notably more difficult than precise effort predictions. Since the recorded risk annotations for a given edge (see table 3.6) are themselves already much more variable, this effect is not surprising. Unfortunately, the lower stability of the risk cost factor itself cannot be compensated by *any* cost estimation technique without dropping the 'black box' assumption of the robot pilot. As this would violate one of the basic paradigms of the robot control architecture examined in the presented work, this approach is not considered as a viable solution. Instead, the limited capability to estimate the exact quantity of risk costs has to be accepted. At a minimum, the developed techniques allow to discern well traversable from less well traversable routes using the local traversability maps, and can otherwise provide a sound prediction for typical risk costs. Additionally, the cost learning guarantees that the estimated costs will be replaced with true experience as soon as the speculative edge has been traversed. This ensures long-term consistency even for the volatile risk cost factor.

Another peculiarity of figure 4.17 is the performance difference between the four estimation strategies. For effort, the reduced prediction errors of the strategy sequence 'global model → local model → local map → edge inversion' listed in table 4.3 reflects properly in higher probability rates for these estimation techniques. However, the risk probability plots do not exhibit a similarly large difference between the four methods. This gives rise to the question where (if at all) exactly the more sophisticated strategies excel at cost prediction compared to the less refined ones. The answer can be taken from figure 4.18, which is another diagram showing the probability of a cost prediction within a given error bound. This time however, only *obstructed* edges are considered, e.g. edges which have a true risk

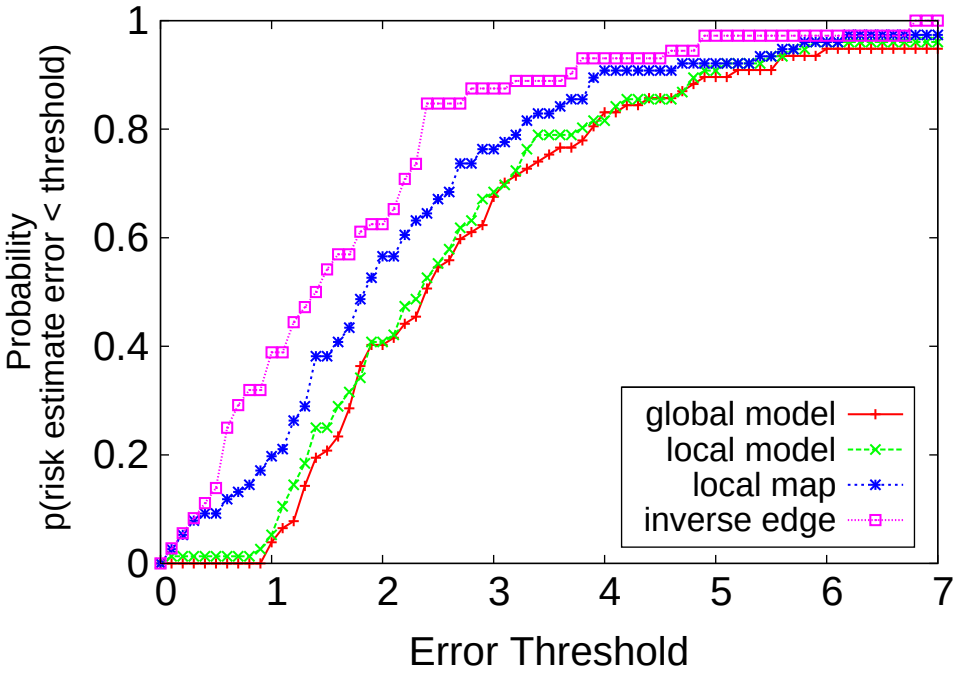


Figure 4.18: Probability of risk prediction with bounded error for obstructed edges

cost $\widehat{R}$ of at least 1. In essence, the graphs in figures 4.17a and 4.18 reveal that all four strategies are equally well capable of predicting the costs for terrain containing little or no severe risk sources. Because the corresponding edges amount to $\approx 77\%$ of the total, the prediction probabilities used in figure 4.17a is dominated by this class of edges, leading to

minimal differences between methods. However, this changes when only the obstructed cases are considered, as shown in figure 4.18. For edges that provoke obstacle encounterse, the true merit of the more sophisticated cost estimation methods becomes visible.

4.2 Map Extension

The last section presented a set of techniques that can be used to predict edge traversal costs for speculative edges, while exploiting the most accurate sources of information available. Using this capability as a building block, the second question posed in the introduction can be approached. The question addresses the topic of *map extension*, or put more descriptively, how the topological map can be extended in order to reach a previously unreachable place. This allows the robot not only to explore new parts of the environment, but also to recover from failed approaches to already known nodes. As described in section 3.4.5, such events are handled by the RepresentFailure algorithm, which turns a traversable edge into an untraversable one. If that edge was the only known way to access a certain node, the topological map must be extended with new connections in order to make that node reachable again via a different route.

In contrast to many indoor applications, the operation environment of an off-road robot is typically not enclosed by an impassable border such as a fence or a wall. Thus, 'full exploration' strategies often proposed for indoor applications cannot be applied here. Instead, map extension must be rather *goal-directed*, e.g. driven by the assigned task to reach a previously unreachable goal.

In order to achieve this using the available capabilities of edge cost learning and prediction, the developed map extension technique proceeds as follows. As input, it requires the specification of a **origin node** n_{orig} and a **goal node** n_{goal} that define the start and end points for map extension. It is assumed that there exists no valid path between n_{orig} and n_{goal} upon invocation of the algorithm. Figure 4.19 shows an example.

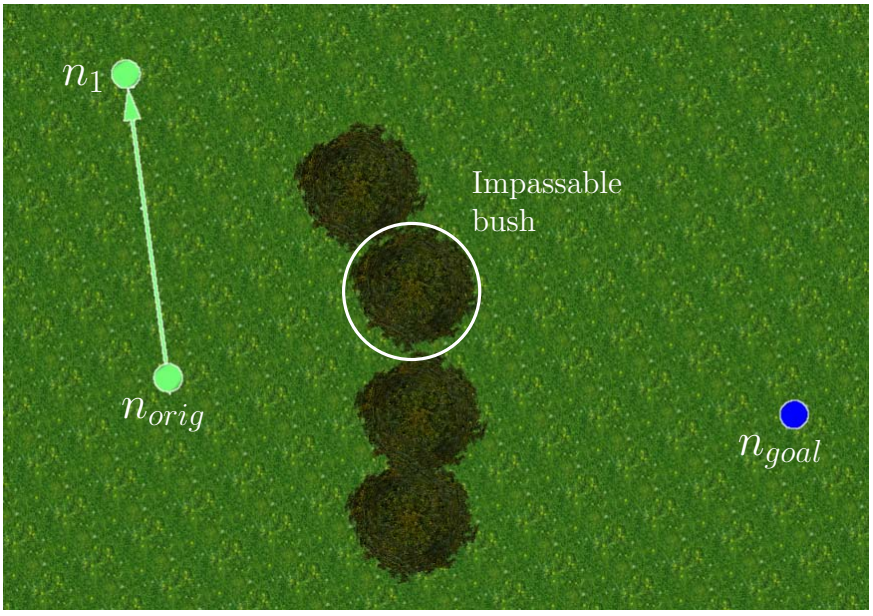


Figure 4.19: Exemplary start situation for map extension

In order to make the goal node reachable via the topological map, a set of hypothetical new routes is constructed which link these previously unconnected nodes. The routes inevitably contain at least some speculative edges, whose costs are subsequently predicted using the methods established in the last section. Finally, the cheapest new route is selected and inserted permanently into the topological map. The other alternatives are discarded. This strategy ensures that only minimal and necessary additions are introduced into the map structure. This agrees with the concept of a minimal world model underlying the thesis.

The next sections provide more detail on the task of hypothesis generation and present two possibilities for the integration of the map extension technique into the existing navigation system. At the end of this section, some experimental results are reported from both simulation and a large-scale real world experiment.

4.2.1 Generation of New Route Candidates

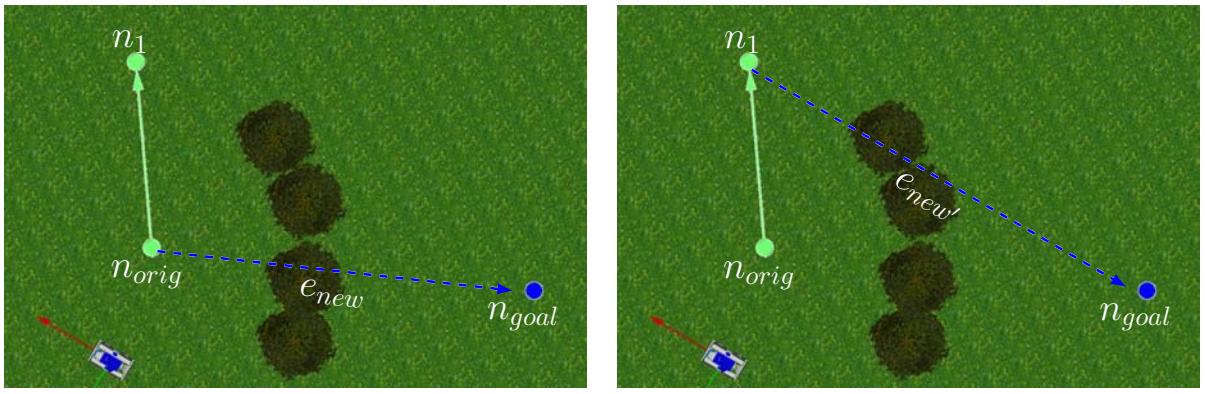
The developed map extension algorithm chooses from two different classes of possible map additions to decide which candidate realizes the 'best' new path between n_{orig} and n_{goal}. First, the insertion of a single new edge between already existing nodes is considered. For this, new edges e_{new} between the origin node n_{orig} or *any node that is reachable from it* and n_{goal} are hypothetically inserted into the map. Then, the edge that allows the construction of the path with minimal travel costs between n_{orig} and n_{goal} is determined. This process is described more precisely in algorithm 8.

Algorithm 8: Route candidate generation I - Direct connections

```
Data: Graph G = (N, E) // the global topological map
Data: Set S // the set of valid start nodes
S ← {n_orig} ∪ all nodes n ∈ N for which Path (n_orig, n) exists
Edge best_edge ← nil
Float best_cost ← ∞
foreach n ∈ S do // main loop
    if (n, n_goal) ∉ E then // don't create duplicate edges
        Edge e_new ← (n, n_goal)
        E ← E ∪ e_new
        Path P ← Path (n_orig, n_goal)
        if Cost (P) < best_cost then
            best_edge ← e_new
            best_cost ← Cost (P)
        end
        E ← E / e_new
    end
    // extension for second candidate class will be inserted here
end
return (best_edge, best_cost)
```

Path(n_1, n_2) denotes a function that computes a Path P from node n_1 to node n_2 within the topological map, if one exists. Costs(P) is defined as the accumulated cost of traversing a Path P using the current motivational state, computed using both the available

annotations and, if necessary, a suitable cost prediction technique. In order to let the Costs function compute measures which allow accurate comparison with other alternatives, the maximal cost factor values $\widehat{R}^{\max}$ and $\widehat{W}^{\max}$ used for normalization are fixated at their current values for the duration of the candidate generation. The algorithm checks before insertion of a new hypothetical edge e_{new} if an edge already exists between the nodes that would be linked. If so, that hypothesis is skipped. This ensures that the map extension does not result in duplicate edges and that routes which have been marked as untraversable are not used again. Figure 4.20 shows examples of the route candidates generated by algorithm 8.

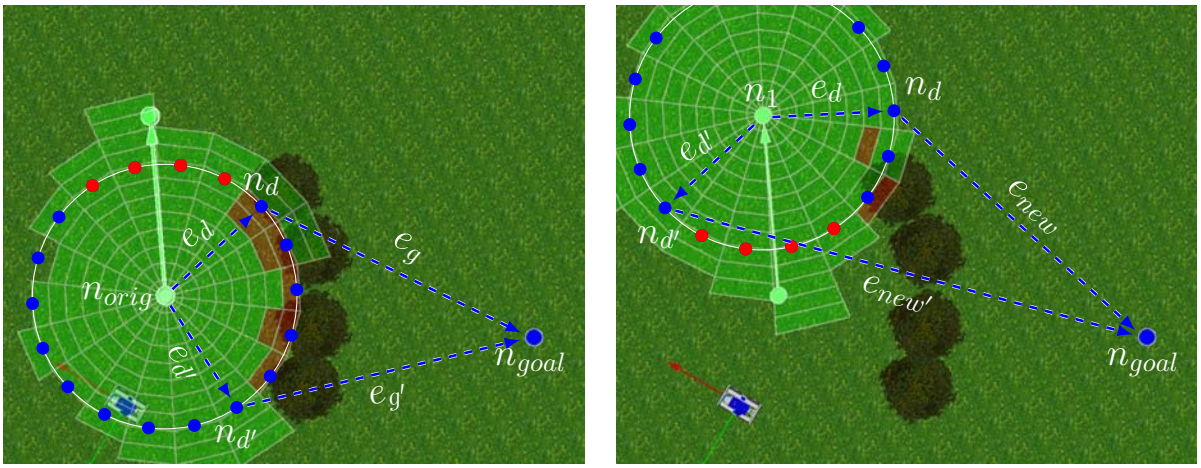


(a) Starting from n_{orig} (b) Starting from n_1

Figure 4.20: Route candidate generation I - Direct connections

The second class of route candidates is generated by splitting the edge e_{new} set up by algorithm 8 into two edges e_d, e_g and placing a **detour node** n_d in between. Thus, if e_{new} originally led from n_{orig} to n_{goal}, e_d and e_g would connect (n_{orig}, n_d) and (n_d, n_{goal}), respectively. The position of n_d provides an additional degree of freedom to construct different route candidates apart from the choice of the start node for e_{new}. In order to focus on a manageable number of candidates, the positions of n_d are chosen to lie at a fixed target distance d from the start node of e_d. Keeping this distance, different candidates are generated by rotating n_d circularly around the start node. As angular step for the rotation, the sector size ϕ of the local traversability maps introduced in section 4.1.3.1 is used. The rationale behind this approach is to take maximal advantage of the information contained in the local map of the start node for the cost estimation of e_d. By reusing the angular sector size ϕ, each sector of the local map is considered once as primary source of traversability information during the evaluation of the n_d candidates. Consequently, any well traversable passage that is recorded in a sector of $e_d.n$'s local map is detected with a high probability. Figure 4.21 exemplarily shows the construction of some candidate routes which contain detour nodes.

In order to compute the ECS position $n_d.\vec{p}$ of the detour node n_d using the current rotation angle $i\phi$, a temporary detour node coordinate system **DCS** is set up using the SetupCSWithXAxis function introduced in algorithm 2 (p. 19). The start node of the edge e_d is used as the origin of the DCS, the x-axis is determined by $e_d.n'.\vec{p} - e_d.n.\vec{p}$. The 'up' direction is chosen as the WCS z-axis $\vec{\Psi}_z$ to minimize the tilt of the DCS with respect to



(a) Two example detours starting from n_{orig} (b) Two example detours starting from n_1

Figure 4.21: Route candidate generation II - Detour connections

the gravity vector. Combining these requirements, the transformation between DCS and ECS is computed by $M_{DCS}^{ECS} = \mathsf{SetupCSWithXAxis}(e_d.n.\vec{p},\ e_d.n'.\vec{p} - e_d.n.\vec{p},\ \vec{\Psi}_z)$.

Using DCS, the detour node positions are easily calculated via rotation of a vector along the DCS x-axis with length d and transformation into ECS, formally expressed by:

$$n_d.\vec{p}^{\,ECS} = M_{DCS}^{ECS} \cdot \begin{pmatrix} \cos i\phi & -\sin i\phi & 0 & 0 \\ \sin i\phi & \cos i\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} d \\ 0 \\ 0 \\ 1 \end{pmatrix} \tag{4.21}$$

The creation process of the detour node routes is the main innovation of the second candidate generation algorithm. The other parts of the implementation, namely the iteration over all nodes reachable from n_{orig} and the selection of the extension with minimal cost, remain as shown in algorithm 8. Thus, this algorithm can be extended at the indicated location with the code shown in algorithm 9 to handle the creation and evaluation of detour routes in the same code frame.

Similar to the detection of potential edge duplicates for the direct connection candidates, the detour route algorithm checks whether a detour node is to be placed too close to an already existing node. If such a situation occurs (examples are marked as red nodes in figure 4.21), it is still possible that the current route hypothesis can be salvaged. In order to decide that, it is checked whether e_d and e_g can be connected to the overlapped node n_{over} instead of the current detour node n_d without creating duplicate edges. If this modification would cause duplicate edges, the overlapped node cannot be used to replace n_d and the current hypothesis is skipped. Otherwise, the map extension can proceed to evaluate the modified route candidate.

This check is motivated by the following deliberations. If e_d would duplicate an already existing edge that is traversable, n_{over} is reachable from the current start node and will itself be selected as start node for the map extension eventually. Thus, the modified route candidate will be generated anyway at some other time and can be skipped now without losing anything. If the duplicated edge is untraversable, the candidate would

Algorithm 9: Extending algorithm 8 to handle routes with detour nodes

```
foreach n ∈ S do // main loop
    // evaluate direct connections ...
    for i ← 0 to 2π/φ do
        n_d.p⃗ ← compute detour node position with rotation angle iφ
        if ¬∃a ∈ N.d(a.p⃗, n_d.p⃗) < min_dist then // don't overlap existing nodes
            N ← N ∪ (n_d.p⃗, speculative)
            E ← E ∪ (n, n_d) ∪ (n_d, n_goal)
            Path P ← Path (n_orig, n_goal)
            if Cost (P) < best_cost then
                best_edge ← e_d
                best_cost ← Cost (P)
            end
            reset E and N
        else
            handle node overlap special case
        end
    end
end
```

reopen a connection that has already been found unsuitable. This would cause the robot to repeat an older navigation failure, which is clearly undesirable. The same reasoning can be applied for edges duplicated by e_g. In summary, any edge duplication is a reason to skip the current route candidate in case the detour node would overlap any other node.

4.2.2 Cost Transfer between Nearby Maps

The map extension algorithm presented in the last section is able to link unconnected topological nodes using only a minimal amount of new map elements. In free terrain, the introduction of a single, direct connection usually suffices to allow travel to a previously unreachable node. In more confined spaces, the extension algorithm can exploit the available terrain information encoded in local traversability maps and build detour routes to circumvent detected obstacles. By iterating this process, even large obstacle constellations can be mastered eventually.

However, the current form of the presented method still suffers from a deficit that reduces its usefulness. The problem is encountered when the radius d used for detour node generation is comparably small and the nodes n_d are placed within an area of the start node's local map which is still filled with traversability information. In this case, the cost estimation algorithm effectively *ignores* the part of the local map that lies beyond the detour node radius. The cause of this problem is the fact that local maps are only used to estimate costs for edges that *originate* at the node to which the map is attached. Consequently, the start node's map is only used to predict the costs for the edge e_d. The cost prediction for e_g cannot benefit from the information stored in the remaining map parts. Thus, the cost prediction of detour route candidates becomes more inaccurate than forced to by the available data and the selected extension route can easily 'poke' through already known (but ignored) obstacles.

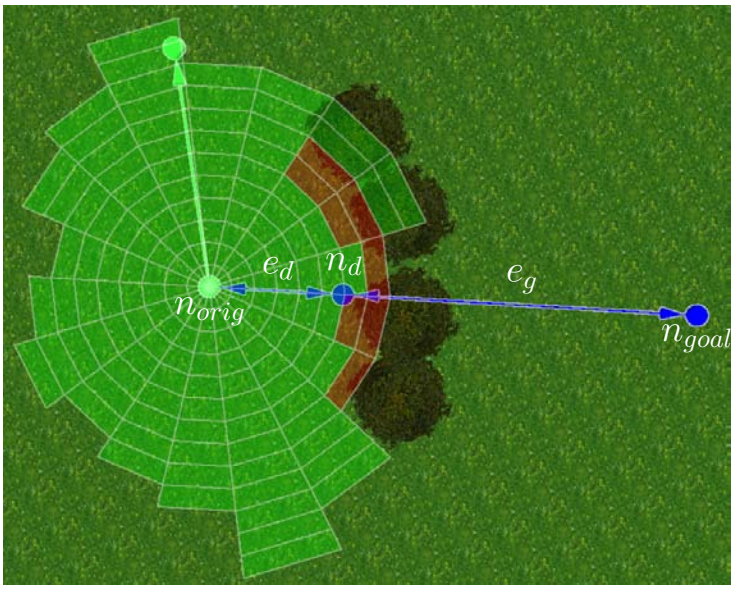
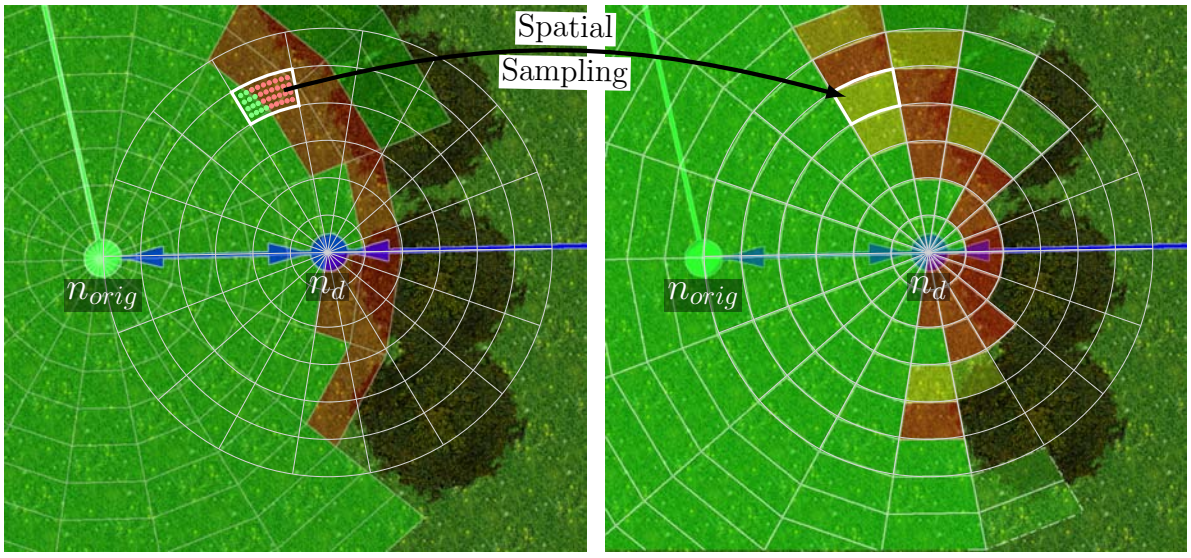


Figure 4.22: Suboptimal map extension for small detour node radii

Figure 4.22 illustrates this problem. The detour node n_d is placed directly in front of the hedge, which has been accurately detected and noted in the traversability map of n_{orig}. Unfortunately, the restricted applicability of the local map causes the shown connection to become the cheapest possible map extension to reach the goal node n_{goal}. For cost estimation, this route consists of only a low cost edge e_d traversing free space and the shortest possible edge e_g which is optimistically estimated using the local or global cost model, *without* including any obstacle penalties.

In order to improve the situation, it is crucial to use *all* available terrain information for the selection of a suitable map extension route. The easiest approach given the current map extension strategy is to choose a sufficiently large detour node radius. However, this workaround is not overly elegant and also requires a priori knowledge about the maximum range up to which the local maps can contain traversability information. A second and more powerful solution is to give up the independence between local maps, so that information transfer becomes possible. In the general case, an unrestricted data exchange between local maps is not desirable, because such a step would ultimately lead to a globally referenced, metrical world description. As already discussed, this requires a relatively exact robot localization to work reliably, which cannot be guaranteed easily for outdoor environments. However, for the special case of map extension, the temporary transfer of terrain information from the local map of the current start node to the detour node's map suffices to let the route selection algorithm choose the best alternative. After reaching the extension decision, the transferred map information can be discarded to restore the independence of the local maps. In this way, the hybrid maps remain as robust to localization errors as they were before.

As a result of the previous discussion, the transfer of cost modifiers from one local map to the map of another node has been implemented in order to augment the map extension algorithm. However, its application is restricted to detour nodes and the transferred information is discarded after selection of the best route candidate. The implementation is based on **spatial sampling** of the source map at the seclet positions of the destination



(a) Sampling of n_{orig}'s map (b) Resampled map for n_d

Figure 4.23: Spatial resampling of a local traversability map

map and is visualized in figure 4.23. For each seclet of the destination map attached to the detour node n_d, a set of sampling points is generated which uniformly covers the seclet's area. The sampling positions are then translated into the NCS of n_{orig}'s source map and used to look up the cost modifiers (r, w) and reliability scores (θ_r, θ_w) of the corresponding seclet within *that* map. All samples of (r, w) and (θ_r, θ_w) taken for the current destination seclet are then averaged to yield an estimate for that seclet. Finally, a decay factor $\kappa = 0.75$ is multiplied with the reliability scores to indicate the loss of spatial accuracy incurred by the resampling step.

After performing the sampling procedure for all seclets of the destination map, the map contains an image of the original map, translated into the frame of reference of the new map. In the current example, the transferred information for n_d contains the impassable area of the detected hedge. Consequently, the cost estimation for the connection shown in figure 4.22 can now correctly include the obstacle penalty for the edge e_g. As shown in section 4.3, this ultimately leads to the preference of another, longer route through free terrain.

4.2.3 Integration

The simplest way to integrate the presented map extension technique into the existing navigation system is to trigger its execution every time the user requests the travel from a map entry node to a map exit node (cf. section 3.6) for which no valid path can be found. In this case, it is straightforward to choose the origin node of map extension as the map entry node, and the goal node as the map exit node. After map extension, the path planner can select the newly created path and traverse it. Further causes which require calling the map extension algorithm are *traversal failures* that occur during path traversals. If the RepresentFailure method breaks the only connection to the current map exit node, the map extension algorithm can be used to create a new hypothetical connection and let the robot continue operation. In this case, the extension origin is set to the node inserted at

the current robot position during the failure handling. The extension goal remains equal to the map exit node as before.

The map extension algorithm can also be used in a less obvious way to provide improved guidance for the behavior-based pilot during the traversal of speculative map edges. If this technique is used, the path planner checks the length of a speculative edge e that is to be traversed against a predetermined maximum length $l_{max} > d$. If the edge is too long, it is removed from the map and the map extension algorithm is called using $e.n$ as origin and $e.n'$ as the goal node. In order to avoid reinsertion of the removed edge, the usage of direct connections as route candidates is disabled in this case. Now, based on the local traversability map of $e.n$, a new detour connection is constructed, which preferably runs through free space. This eases the job of the piloting layer and makes the emergence of a well predictable line-of-sight trajectory more likely. It is important to note that this **Speculative Edge Split** method will be usually applied several times for a given initial edge, as each split only reduces the length of the second inserted edge e_g by the detour node radius d. Thus, e_g is potentially still longer than l_{max} and will be split again once n_d has been reached. In effect, this leads to the replacement of a long edge with a sequence of shorter segments that follow a path of locally optimal traversability. In essence, this technique can be used to implement a local path following strategy on the basis of the terrain traversability information available to the navigator.

4.3 Experiments and Results

Figure 4.24 shows the final speculative route that has been selected as the best possible map extension to reach n_{goal} from n_{orig} given the original situation of figure 4.19 and a comparably short detour node radius of $d = 8$ m. Agreeing with intuition, the chosen

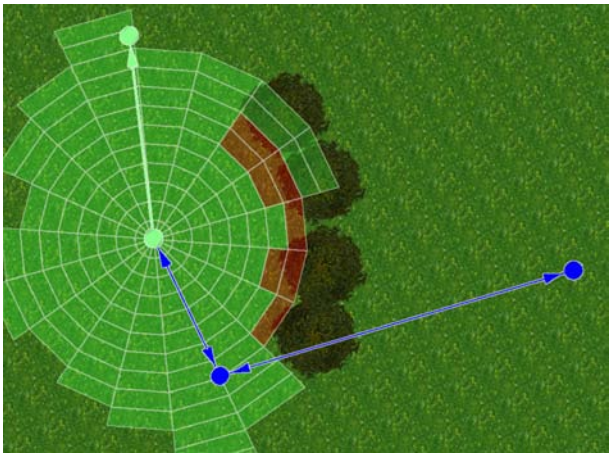


Figure 4.24: Extension route for simulation scenario

route is the shortest possible connection to n_{goal} that still circumvents all detected parts of the hedge. The extension can be deemed an appropriate choice, as it appears extremely likely that the pilot will be able to successfully traverse the new connection. The skirting of the lowest bush by the edge originating from the detour node (a result of missing information in the local traversability map) is easily corrected by the local obstacle avoidance mechanisms.

The usefulness of the presented algorithms has also been demonstrated in a real world situation during the European Land Robot Trial (ELROB) 2008 robot competition in Hammelburg, Germany (http://www.m-elrob.eu/). Both the map extension and the speculative edge split techniques have been key factors for the success of the RAVON robot during the qualification run for the reconnaissance scenario of ELROB 2008. The goal of this qualification run was to travel $\approx$ 1 km through a priori unknown terrain, with both start and end point of the track made public just before the start. Because the use of freely available tools to generate intermediate waypoints was permitted within the competition's time limit, the GOOGLE EARTH™ software was used to quickly set up an initial path from the start to the end of the qualification track. The robot RAVON, equipped with the combined behavior-based / topological navigation system described in this thesis, was then used to traverse this path in a fully autonomous fashion. The robot successfully arrived at the second to last waypoint within the premises of a mockup military camp that contained the final destination of the trial before the time limit was reached. With this performance, the robot qualified itself as one of 4 competitors (out of 11 teams) for the final run of the reconnaissance scenario that was held during night. It also was the *only* fully autonomous vehicle that passed the qualification.

Figure 4.25 shows a GOOGLE EARTH™ screenshot of the terrain covered during the competition together with an overlay of the final map constructed by the robot's navigation system. The small circles indicate topological nodes and the connections between them represent edges. The color scheme is equal to the standard used thorough this thesis: green lines indicate traversable edges and reachable nodes, while blue signifies speculative elements and red untraversable edges.

Initially, the robot was commanded to travel from the node `P0` on the extreme right of the image to the goal node `P16` visible on the left side, along the route specified by the intermediate `P` nodes. The pilot successfully followed this route until the connection of `P3` and `P4` had to be traversed. Here, a mixture of debris and high vegetation, both not visible in the aerial image, blocked the path. Consequently, the edge traversal failed. In response to that, the RepresentFailure algorithm marked the connection edge as untraversable and inserted a new node `E0` at the current robot position. Now, because this broke the only connection to the goal node `P16`, the map extension algorithm was called for the first time. It inserted a new direct connection from `E0` directly to `P16`, which starts off in the direction of a well traversable dirt road. Although this road is the best track to the goal, it had not been used for the original route setup because the other, more difficult road would have provided a higher score for the competition. Upon initiating the traversal of the new connection, the speculative edge split strategy was invoked repetitively and split the extremely long direct connection iteratively into shorter segments with a length of 20 meters (the detour node radius used in this scenario) each. Using the information obtained from the local obstacle memory, the new detour nodes were placed accurately along the well traversable dirt road, slowly progressing in the direction of the goal. The edge splitting terminated once one of the picked detour nodes coincided with the `P13` node, which subsequently replaced that node. This way, the path planner 'jumped back on track' and could follow the originally set waypoints to continue its travel and enter the military compound. At this time, the qualification run was ended by the jury because the time limit was reached. Nevertheless, this successful demonstration of global navigation skills led to the qualification of the team for the final scenario of the competition. This

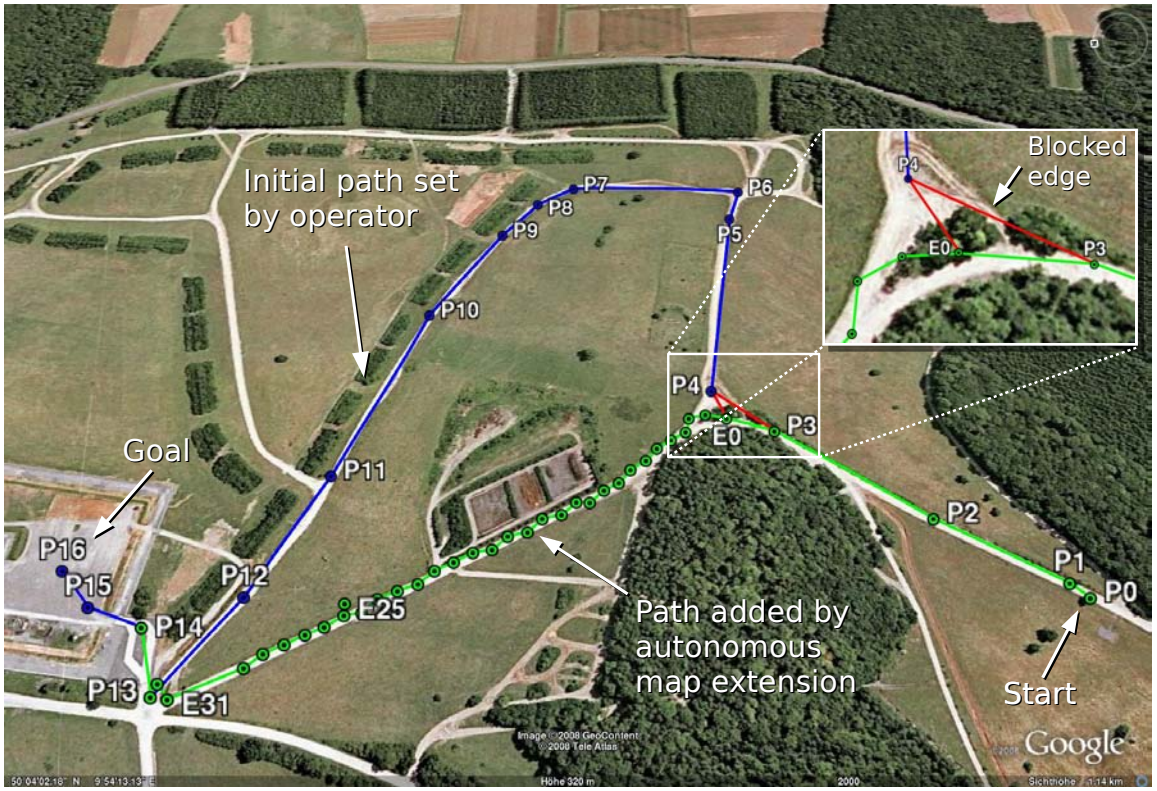


Figure 4.25: Autonomous map extension during European Land Robot Trial 2008
The image was produced using Google Earth™.

was accomplished by just 3 other teams out of 11 contestants. RAVON also achieved the highest possible scores for navigation autonomy in both the qualification and the final run.

In summary, the qualification run was an impressive demonstration of the robustness and flexibility of the developed navigation and map extension system. The robot successfully recovered from a traversal failure and autonomously constructed a path along a well traversable road without relying on *any* previous information about that road or even an explicit road detection or following strategy. After successfully generating a long and sensible path segment that led the robot safely toward the goal, the path merged again with the original track by coincidence. The path planner then switched back seamlessly and used the provided track information which had not been replaced by the autonomously generated shortcut.

4.4 Conclusion

In this chapter, four methods were presented which allow the prediction of traversal costs for up-to-now untraversed edges. Three of these methods reuse information stored in the cost annotations of the topological map edges, but incorporate data on different levels of locality. The coarsest prediction technique relies on a global cost model constructed from all available cost annotations using an outlier-robust linear regression of risk and effort cost factors. This model correlates the estimated travel length of the hypothetical edge with its probable cost by extrapolating global, overall terrain cost characteristics. A local

(a) At the starting point (near P1)

(b) Along the course (near P2)

(c) Along the course (near E25)

(d) Near the camp entrance (near P13)

Figure 4.26: Images from the qualification route of ELROB 2008
Image source: Deutsche Bundeswehr (German Army). Reproduced with kind permission of the copyright holder.

cost model is built by the second prediction method, which restricts the spatial extent of edges eligible for cost transfer. This allows to better model local fluctuations of terrain properties, improving risk and especially effort predictions. The third approach predicts costs based solely an edge's direct inverse twin, which is the spatially closest source of information available.

In order to account for terrain properties which are not captured by the extrapolation of topological map information, a fourth cost prediction algorithm was proposed which uses the local traversability maps of the hybrid world model as information source. These maps are filled by extracting traversable free space and untraversable obstacles from the obstacle memory of the local pilot. By combining the learned costs in the topological edges and the modifiers stored in the local metrical map, the accuracy of cost prediction can be greatly improved. This is especially relevant for the risk cost factor, which depends on the amount of obstacle evasions during edge traversal.

Experimental validation of the cost prediction algorithms revealed that increasingly accurate extrapolation techniques can be selected as the available cost information accumu-

lates. The overall performance of the prediction strategy was verified and each method was quantitatively analyzed based on an extensive large-scale simulation test.

Based on the developed cost extrapolation methods, an exploration strategy was proposed which generates new connection hypotheses from two sets of possibilities. The first set of extension hypotheses adds direct connections between a reachable node and the previously unreachable goal node. The second extension strategy inserts additional detour nodes, which are placed according to the sector sizes of the local traversability maps. This allows to optimally exploit the traversability information contained inside. After evaluation of all valid hypotheses, only the candidate with the lowest predicted costs is actually incorporated into the map. This keeps the map as small as possible, in accordance to the formulated objective to retain a compact and scalable world model.

The devised map extension approach was tested in a real-world, competitive scenario posed during the ELROB 2008 trial. In this trial, the global navigation system was capable to recover from an impassable obstacle that blocked the predefined route and generated an alternative route which led the robot safely along a pathway over 1 km, until the alternative route merged again with the predefined one. This successful demonstration of global navigation skills led to the qualification of the team for the final scenario of the competition. This was accomplished by just 3 other teams out of 11 contestants. RAVON actually achieved the highest possible scores for navigation autonomy in both the qualification and the final run.

5. Shape-Based Terrain Traversability Estimation

The last chapter presented several techniques to predict the costs of speculative edges and extend a topological map in order to reach new targets. It has become apparent that the use of local traversability maps is vital both to optimize the risk estimation for speculative edges and to select suitable routes for the detour node based map extension. However, the traversability maps are up to now filled solely with information taken from the local obstacle memory of the piloting layer. As the pilot's main task is *local* trajectory generation and obstacle avoidance, this memory has a rather limited spatial range. Thus, the navigator can only use short-range information to choose a route that leads the robot through obstacle free terrain. As a consequence of this restricted knowledge horizon, exploration decisions are likely to be myoptic and not optimal when considered in a larger scope.

In order to improve the performance of the navigation system, long range information about the traversability of the surrounding terrain must be acquired. Once suitable data is obtained, it can be inserted into the (now less) local traversability maps in the same way as the local obstacle memory data which is already used. Thus, the main difficulties that still need to be solved are a) the acquisition of appropriate sensor data covering a large area of the environment and b) the analysis of this data in order to extract traversal cost modifiers for the local map.

In the following, a new approach for terrain traversability estimation is presented which has been developed for this thesis by the author in cooperation with H. Bitsch [Bitsch 08]. In summary, this approach obtains high resolution *color images* of the terrain within a range of up to ≈ 30 m around the robot using a turnable stereo camera system. Based on the distance measurement derived using stereo reconstruction, a *geometric model* of the terrain surface is generated. Finally, a traversability analysis of the surface model is performed and a local traversability map is filled with the results. The approach has been published to the scientific community in [Braun 08b].

The rest of this chapter is structured as follows. After a short survey of already existing methods, the developed techniques for the tasks of data acquisition and traversability

estimation are described in detail. Afterwards, experiments are presented to document the benefit for the cost prediction and exploration steps that can be obtained using the now available long range information.

5.1 Related Work

Traversability estimation is a key issue for path planning in outdoor terrain. Therefore, a lot of different approaches have already been published. To remain focused, the following survey concentrates on data acquisition and terrain evaluation techniques suitable for outdoor vehicles in an off-road or desert like terrain.

5.1.1 Data Acquisition

Established techniques for outdoor traversability estimation typically use either 3D LIDAR range sensors [Vandapel 04] or stereo systems [Simond 06] [Huertas 05] [Howard 06] [Kim 07]. While LIDAR systems can quickly produce accurate 3D point clouds distributed along the terrain surface, they do not provide visual information useful for terrain classification such as terrain color or texture.

In contrast, stereo cameras capturing high resolution color images can support appearance-based terrain type inference. However, these systems have to derive the distance information for 3D point cloud generation algorithmically, which increases computational load and limits range accuracy. Even worse, stereo reconstruction depends heavily on suitable input data. On problematic image areas (e.g. having low contrast), stereo matching may fail.

There are different ways to obtain depth information from stereo images. Aside from the 'common' technique of generating a dense disparity map using algorithms such as described in [Birchfield 99], some research groups use methods based on optical flow calculation between consecutive frames ([Kagami 05], which used the KLF tracker by [Tomasi 91]) or specialized techniques like [Gemme 05].

In order to combine their benefits, several researchers have proposed to use both sensor types simultaneously and perform data fusion (e.g. [Braid 06] [Cremean 06]). However, consistency over long ranges requires an extremely accurate mutual sensor registration and a mechanically rigid construction to prevent misalignments caused by vibrations during robot operation. Thus, sensor fusion has not been widely adapted for sensor systems with moving parts such as pan/tilt units. Nevertheless, a general overview of registration techniques can be found in [Brown 92].

In [Rasmussen 02], the combination of a laser range-finder and color/texture images of an outdoor scene is used for road detection. To avoid the registration problem, the laser scanner was registered to the stereo rig by using the algorithms introduced in [Elstrom 98]. The combined data is used as training models for a neuronal network in order to detect road like areas.

In [Cremean 06], a different approach was taken. For every sensor, the acquired data was transformed into grid of fixed size and orientation, the desired calculations were performed and the results were pessimistically fused by taking the minimal resulting value for each cell.

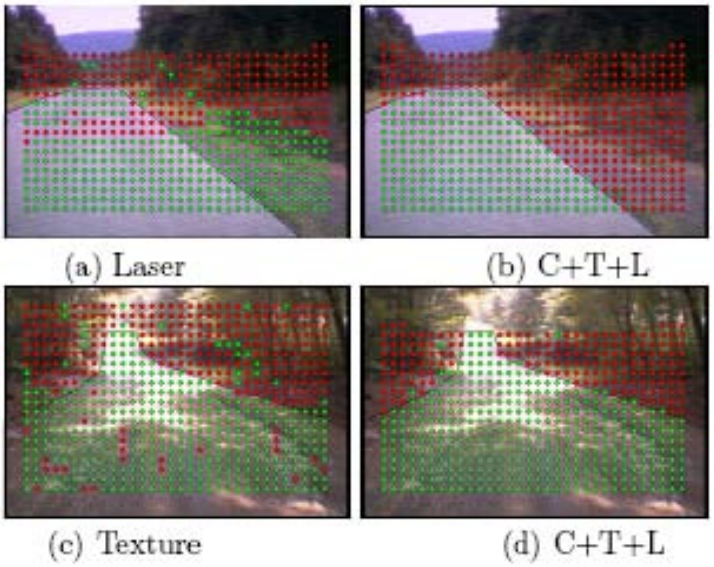


Figure 5.1: Road detection using Laser (L),
Texture (T) and fused data from Laser, Texture and Color(C) (Source: [Rasmussen 02])

5.1.2 Traversability Estimation

Two main concepts can be found to extract traversability information from the obtained raw sensor information. One idea is to *reconstruct* the environment in the form of a geometrical (3D) model and derive traversability information from the model in a second step. The other one is to use *classification* algorithms to group elements in the scene directly and reason upon the classification results.

Surface Reconstruction

The 3D reconstruction of the environment is often based on a cloud of surface points generated by the applied sensors. The idea of almost all methods presented here is to segment the point cloud into smaller parts and analyze those. In some cases, based on the analysis, a region growing of similar areas is performed as a form of split and merge algorithm. Various types of segmentations such as voxels or patches are available. A rather novel approach is the superpixel representation which is compared to normal patches in [Kim 07].

Figure 5.2: Octree representation from different angles of a point cloud (Source: [Nuechter 07])

For mapping of an area or SLAM (simultaneous localization and mapping) purposes, a truly three-dimensional representation of the scene is desirable. An example using

a LIDAR system can be found in [Nuechter 07]. There, an octree based segmentation process of the point cloud is introduced (see figure 5.2). This octree representation of the scene is updated with new scanning data from different locations using a kd-tree enhanced version of the ICP registration algorithm as introduced in [Besl 92].

For outdoor navigation and driving, a full 3D map of the area contains more data than actually needed and is expensive to generate. Several approaches therefore use digital elevation maps (DEM) or height maps as a sufficient representation of the area in a 2.5D space. One implementation of this technique is discussed in [Vergauwen 01], where a DEM is used in the navigator component of a Mars rover equipped with a stereo camera setup. The traversability estimation of an area is based on a metric using the gradient of the desired area in the DEM. In this approach, the DEM is directly calculated out of the stereo data and the disparity image without the diversion of generating a point cloud. Instead, it uses a technique based on virtual vertical lines and their shadows in the scene.

In [Weingarten 03] a different approach using voxels and plane fitting is introduced. Based on LIDAR data, the cloud of point is segmented using voxels. For every voxel, an average plane is estimated by using RANSAC and least square plane fitting. By not making any assumptions regarding the topology of the input data, it can be applied to various 3D sensor data.

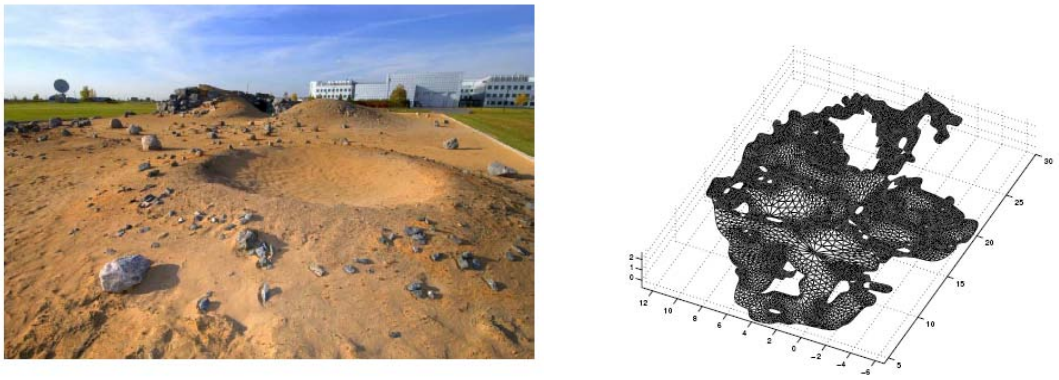


Figure 5.3: Surface reconstruction using a triangular mesh (Source: [Gemme 05])

Triangular meshes can also be used for surface reconstruction as shown in [Gemme 05], where such a mesh is fitted on a cloud of points received from a LIDAR system (see figure 5.3). On the resulting mesh, traversability is estimated by geometrical considerations for each triangle. Designed for a planetary rover, this approach is mainly suitable for obstacle-free terrain, but can detect holes in the ground and steep slopes.

Another approach to sensor fusion and evaluation of digital elevation maps (DEMs) regarding traversability can be found in [Cremean 06]. Their outdoor vehicle Alice features several mid range laser scanners (up to 25 m) as well as a long range stereo vision system with a baseline of 1 m and a visibility of up to 50 m. In a first step, a separate DEM for every sensor is generated. The map is of fixed size, grid dimension and orientation relatively to the vehicle. Incoming 3D point data is transformed according to the current GPS and inertial system measurements and overlaid with the grid. A height value is calculated for every cell in the grid. In a second step, a traversal speed limit is calculated for every cell (see figure 5.4). This is done by considering the variance in elevation. A larger variance means a larger spread in elevation measurements and a higher probability

the cell contains an obstacle, reducing the resulting speed to 0. A second source for speed limit estimation is derived from calculating a Gaussian weighted average of the differences in height between the current cell and its neighbors and transforming this value into a speed limit. The smaller one of the two is used. Every time new sensor data appears, the DEM is updated and the affected cells are recalculated.

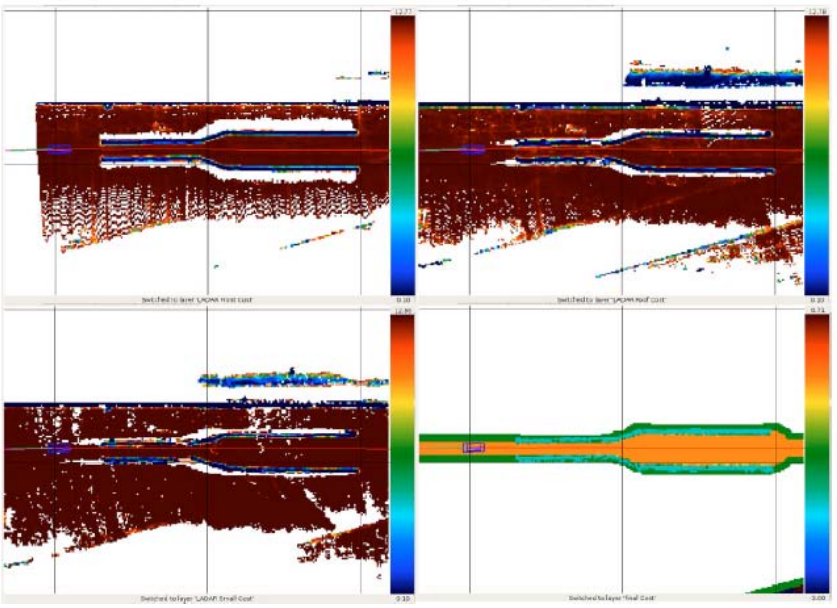


Figure 5.4: Speed maps from three different LADAR sources and the combined speed map enhanced with race corridor information (Source: [Cremean 06])

To fuse the different sensor DEMs into a final DEM, the speed limits for every cell are merged using a modified weighted average algorithm, considering the range and characteristics of each sensor. This is expected to reduce the registration problem. In a last step, the final DEM is overlaid with data from a road following algorithm in order to remove speed information from cells outside the estimated race corridor.

Classification Based Techniques

The algorithms introduced in the last section generate a detailed geometrical representation of the environment to reason upon, but do not include knowledge about the actual type and material properties of the observed structures. In contrast to this, classification based methods try to actually classify areas into roads, bushes, trees or fences and use trained or preconfigured knowledge to estimate traversability from this classification. Since the focus of the current chapter is on *geometry* based approaches, methods relying on different kinds of features such as color and texture will be skipped at this point. The next chapter contains a more in-depth review of these approaches.

An example for classification based on predefined criteria can be found in [Hebert 03]. Based on LIDAR data, the acquired 3D point cloud is segmented by a voxel grid. For every point a shape matrix characterizing the local distribution of all points in the same voxel is calculated. In a second step, every point is classified into different classes named scatter, surface and linear (poles) using an automatically trained Bayesian classifier (see figure 5.5). A proper registration of different scans is crucial for this approach. Additional research

effort has been put into this idea and it has been advanced as shown in [Lalonde 06] in terms of classification performance and accuracy.

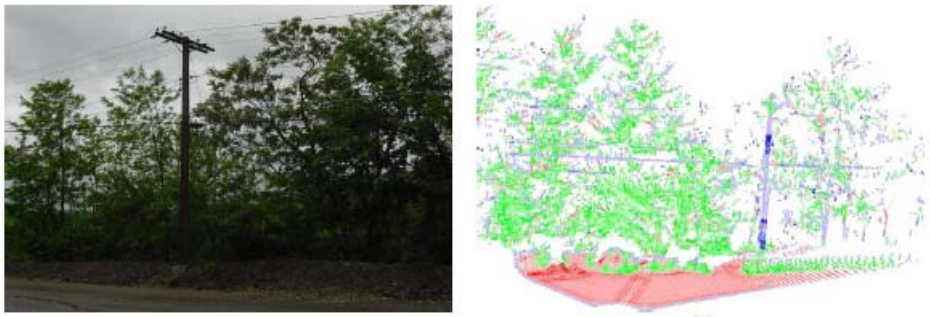


Figure 5.5: Classification of LADAR data including wires (Source: [Lalonde 06])

In [Braid 06], the TerraMax autonomous vehicle is introduced. It is equipped with multiple LIDAR scanners and a trinocular vision system with a baseline between the outer cameras of 1.5 m. The vision system is used for obstacle- as well as path detection. For both aspects, the vehicle calculates the average terrain slope in front of the vehicle first. Any area deviating too much from these estimates are marked as an obstacle. Its actual position is calculated by stereo triangulation and refined with LIDAR data. To estimate the drivable path in front of the vehicle, the smoothness of the slope in front of the machine is fused with similarities in texture, color and shape to constitute a free way recommendation to the path planning module.

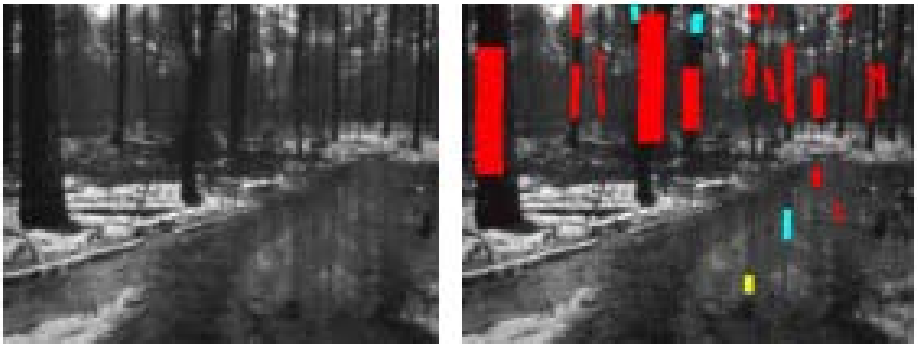


Figure 5.6: Tree detection in a winter forest (Source: [Huertas 05])

A different approach to estimate the traversability of a forest-like area is presented by [Huertas 05]. It aims at the detection and assessment of tree trunks. Huertas tries to detect fragments representing trunks by matching edges of opposite contrast. The area in between can be approximated by an ellipse, which was found to be sufficient for encoding shape, position and orientation of the trunk. The distance of the detected trunks is computed by calculating the average stereo range data of the area the fragment covers. The method works for normal and infrared images, but is focused on vertical shapes and the described version only works on trees which can be clearly distinguished from the background. An example is given in figure 5.6.

5.1.3 Selection of an Appropriate Sensor System

The survey of related work above showed that traversability estimation is most frequently based on LIDAR sensors, stereo camera systems or a combination of both. Out of these options, a **turnable stereo color camera system** has been selected as exclusive information source for the approach developed in the scope of this thesis. This choice is motivated by several factors:

- The intended range of up to 40 m is close to the technical limit of commercially available laser scanners (such as the SICK sensors) and cannot be extended algorithmically if desired. In contrast, the range of the stereo vision system can easily be extended by increasing the distance between the cameras. The fact that this also increases the minimum distance for object detection can be tolerated, as close range objects are already covered by the pilot's local obstacle memory.
- The angular resolution of a camera can be much higher ($\approx 0.025°$ per pixel for 40° FOV at 1600 pixels) than of comparably expensive laser range finders ($\approx 0.5°$). Even after combining multiple pixels into one depth value during stereo reconstruction, the angular resolution usually remains superior.
- The color images contain additional information like color or texture which can be used for *terrain type classification* based on visual appearance (e.g. using texture, feature or color analysis). Because vegetation discrimination is a very important topic for the envisioned application domain, this is a major benefit of using cameras.
- The use of a single camera system avoids registration issues that plague sensor fusion approaches, while the panning degree of freedom can be used to increase the field of view of the system. [1]
- The speed penalty that results from using a mechanically rotating system and the need to recover depth information using stereo algorithms is acceptable, because the navigator does not need to perform the long range traversability analysis frequently. It is sufficient to start this analysis whenever difficult map extension or route planning decisions have to be made.

Considering the presented approaches for traversability estimation, both the reconstruction of a geometric model and the appearance-based terrain classification have distinct benefits and drawbacks. The modeling techniques can determine the layout of the terrain surface without relying on an initial training phase. Thus, they can be readily used in *any* environment to produce a traversability prediction based on predefined, shape-based criteria encompassing terrain slope or the presence of abrupt steps. On the negative side, these approaches assume that the reconstructed surface is also load-bearing, which can be a serious misjudgment in terrain covered with vegetation.

Consequently, many groups use a terrain type classification approach for robots expected to operate in vegetated surroundings. These techniques can distinguish between load-bearing, solid surfaces and soft vegetation. While this allows to obtain qualitatively

[1] Some previous experience of the author with the problematic mutual registration of camera and laser scanner systems has been documented in [Braun 05].

superior traversability estimates than model based techniques, image-based classification algorithms typically require training data in order to work reliably. This data must either be supplied manually prior to robot operation, or generated online by the robot itself using a self-supervised learning technique. Manually provided data limits the robot's operation environment to a priori known terrain types and thus severely restricts the flexibility of the system. Alternatively, the self-supervised generation of training data requires that image areas can be matched spatially with sensor 'experiences'. However, the three-dimensional information about the visible terrain that is needed for this already constitutes a geometric terrain model. Thus, it appears that a classification based approach which uses self-supervised learning must be considered as an *extension* of a geometrical modeling technique, rather than an alternative to it.

In summary, the discussion indicates that the construction of a 3D surface model and subsequent traversability estimation based on the terrain shape seems to be the most promising approach to extend the current navigation system. The obtained geometrical information can be analyzed directly to improve the performance of the path planning system. Later, the long range traversability system can be extended on the basis of the collected three-dimensional information with a classification based approach that implements both terrain type classification and spatially consistent online learning capabilities. This topic will be addressed in the next chapter.

5.2 A New Approach for Shape-Based Traversability Estimation

Motivated by the deliberations presented above, a new approach for terrain traversability estimation using stereo cameras and shape-based traversability estimation is proposed. The algorithm proceeds in the sequence of steps shown in figure 5.7.

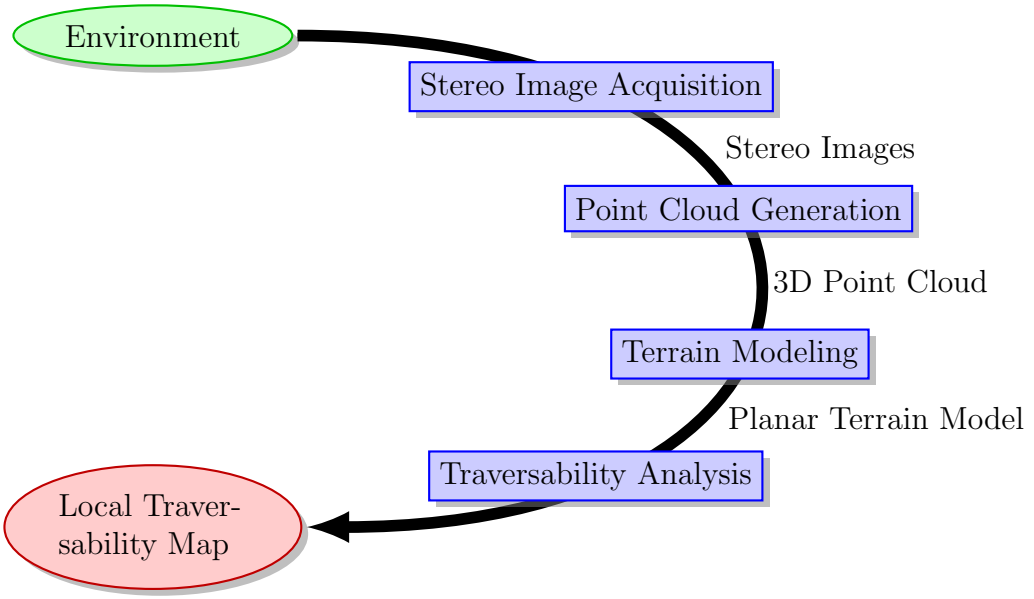


Figure 5.7: Workflow of proposed method

Using the camera system, well exposed stereo images are recorded from the terrain which is to be evaluated. Then, a stereo reconstruction is performed to produce a three-dimensional

point cloud outlining the terrain surface. Subsequently, a piecewise planar terrain model is fitted to the point cloud in order to allow reasoning about the traversability of the corresponding planar patch. Finally, the results of the traversability analysis are transferred into a local traversability map in order to make the data available for the cost prediction algorithm.

Each of these steps will be discussed in detail in the following sections.

5.2.1 Stereo Image Acquisition

The accurate reconstruction of the geometrical terrain shape over the target range of $\approx$ 40m requires a camera system with very high resolution. Therefore, two Point Grey Research Scorpion SCOR-20SOC-CS CCD cameras as depicted in figure 5.8b have been selected as primary sensors. The cameras are connected via an IEEE1394a firewire serial bus [IEEE 00] to the host PC and provide images with a resolution of **1600 x 1200** pixels. Similar to the human eyes, the usage of two cameras with a mutual horizontal displacement allows to recover the three dimensional position of features visible in both camera images using a **stereo reconstruction** algorithm, described in section 5.2.2.4. The selected models automatically synchronize themselves on the hardware level to guarantee that both images are taken within at most a 125 μs time interval. This feature substantially eases the task of feature matching required by the stereo algorithms. As lenses, two manual focus mega-pixel models from Computar have been chosen. They provide a horizontal field of view of 56°. This is perceived as an acceptable compromise between covering a large area in a single image and achieving a high spatial resolution.

In order to extend the angular coverage of the cameras, the rigid stereo system is attached to a pan/tilt unit shown in figure 5.8a[2]. The unit is a purpose build device actuated by one stepper motor each for the pan and tilt axis. The motors are controlled by a DSP circuit board, which communicates with the host computer via CAN bus. This interface allows to issue initialization commands, or provide absolute values for the pan and tilt angles.

In the current configuration, a pan angle of approximately 250° (−125° to 125°) and an tilt angle of approximately 130° (−65° to 65°) can be covered. A direction of 0° turns the camera head to the front direction with a horizontal alignment of the tilt unit. At the end of the turning range, stop switches prevent the head from overwinding.

Coping with Variable Lighting Conditions

A major challenge when capturing images outdoors are rapidly changing lighting conditions and the high dynamic range of the visible environment. The two main camera control settings that can be used to deal with both are the **shutter** and **gain** settings.

The shutter value defines the time during which the cameras internal Charge-Coupled-Device (CCD) sensor accumulates electrical charge (which is proportional to the amount of incoming light). A higher value allows the CCD sensor to accumulate more charge during light exposure, which can compensate e.g. dim lighting conditions. On the other hand,

[2]As seen in the figure, the cameras are enclosed in a copper-covered chassis. This shielding became necessary after tests revealed that the cameras emit electromagnetic radiation which strongly interferes with the main GPS unit placed behind the cameras.

(a) Pan-tilt unit (b) Camera sensor

Figure 5.8: Image acquisition hardware

a lower value prevents the image from getting too bright or even saturated when taking images in bright conditions. In digital cameras, the shutter is commonly implemented as an electronic circuit which removes the charge from the CCD just before the exposure starts. After the time specified by the shutter value, the sensor is read out.

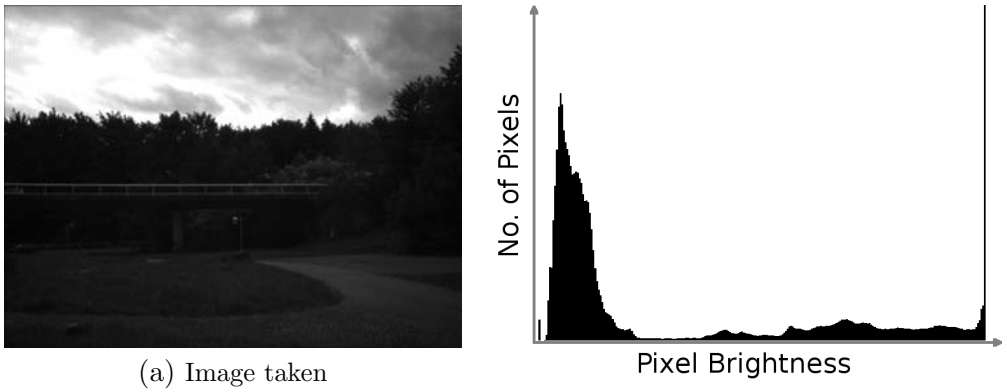
The gain property controls the internal signal amplifier, which boosts the voltage of the accumulated charge to usable levels before it is digitized. High gain values can overcome dark lighting conditions, but introduce amplification noise into the final image, resulting in random speckles of slightly different color or brightness.

The camera has algorithms to control the gain and shutter values automatically. This **auto-exposure** feature can be used to achieve a certain brightness of the entire image. In order to focus research effort more on novel traversability estimation techniques, the camera's build-in automatics for shutter and gain was initially used in order to acquire images with acceptable illumination.

Unfortunately, the standard automatic control turned out to be unsuitable for outdoor situations which exhibit a large dynamic range. Figure 5.9 illustrates the problem. Although the photographed scene is partly cloudy, the sky is much brighter than the ground plane. This causes the camera automatic, which is designed to optimize the *overall* brightness of the entire image, to decrease shutter in order to reduce the over-exposure of the sky. However, as a consequence, the lower part of the image is rendered too dark and details in the texture-rich ground area are lost. The corresponding brightness histogram is shown in figure 5.9b. As can be observed, the histogram is very unbalanced. Many pixels have a very low brightness value, while another large group of over-exposed pixels is capped at maximum brightness.

To overcome this problem, the camera automatic has been replaced by an algorithm specifically designed to be robust against large brightness variations which occur frequently in outdoor environments. Instead of optimizing the overall brightness value of the entire image, the new algorithm's goal is to keep the majority of the image *well-exposed* (within a reasonable brightness range).

In the outdoor setting that is the primary concern here, large brightness variations often emerge between image parts that show sky and ground. Fortunately, the sky contains



(a) Image taken with automatic exposure control

(b) Brightness histogram

Figure 5.9: Automatic exposure control in outdoor environments

no traversability information which needs to be captured, so the new shutter and gain control can safely sacrifice (over-expose) the sky in order to record the other parts of the image well. In order to achieve this, the algorithm exploits the fact that the two parts are usually arranged horizontally. Thus, the algorithm starts by segmenting the image into *horizontal bars* as shown in figure 5.10.

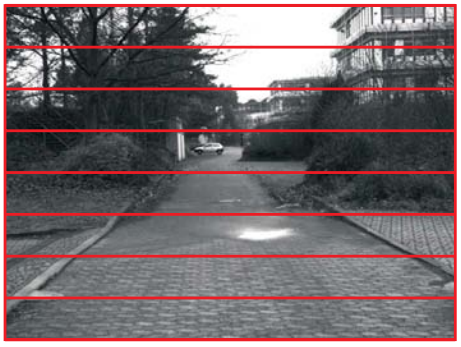


Figure 5.10: Image brightness evaluation
The algorithm evaluates the image brightness using eight equidistant bars. Mean values (highest to lowest bar): 159, 131, 69, 65, 78, 123, 130, 124.

Instead of using the mean value of the whole image for shutter adjustment, the mean value of every single bar is calculated and rated separately as too dark (d), valid (v) or too bright (b). During test runs, a valid brightness rating in the range of $70-150$ (out of the possible values between $0-255$) has yielded satisfactory results. If the number of valid bars exceeds a configurable threshold (typically 65% of all bars), the image is considered usable and processed further. If the threshold is not reached, the image is regrabbed with a different shutter value. It is sufficient to adapt the shutter value by simply adding or subtracting a fraction of the old value, depending on whether the number of dark bars n_d exceeds the number of bright bars n_b as shown in equation 5.1.

$$shutter_{new} = shutter_{current} \cdot \begin{cases} 1.3 & n_d > n_b \\ 0.7 & n_d \leq n_b \end{cases} \tag{5.1}$$

Under certain dim lighting conditions (sunset, indoors), increasing the shutter value alone is not sufficient to produce well-exposed images. In this situation, the shutter value will

eventually reach its upper limit. Then, the optimization algorithm starts to modify the gain parameter using a similar adaption rule as presented in equation 5.1. However, as this adds noise to the image, high shutter values are always preferred to high gain values (→ shutter before gain strategy). Therefore, if image brightness is to be reduced again, gain is lowered first, and only after gain values reach the lower limit, the reduction of shutter times is started.

Finally, in order to handle cases where there is no chance to acquire a well-exposed image, such as in complete darkness or directly facing the sun, the number of regrabbed images is counted. If eight images are not accepted in succession, the image with the highest number of valid bars is used regardless of the threshold.

An example of a real outdoor scene with difficult illumination conditions is depicted in figure 5.11. Using the presented shutter and gain control strategy, a well-exposed image foreground showing bushes and grass is obtained despite the brightly visible sky in the upper image parts.

Figure 5.11: A well exposed foreground despite direct sun exposure

5.2.2 Point Cloud Generation

After capturing a well-exposed pair of images, the depth information of the image content needs to be recovered in order to build a three-dimensional model of the visible terrain. As already mentioned, this can be achieved using *stereo reconstruction*, a technique which calculates distances between the viewer and objects by evaluating the apparent displacement of these objects in images taken from slightly different viewpoints. Stereo reconstruction is thus based on *parallax*, akin to the stereopsis process of the human visual system.

In order to perform stereo reconstruction based on two captured images, the so called *correspondence problem* needs to be solved. Objects or visually distinct image parts (**features**) that correspond to each other must be found in both images and correctly matched. Then, the two feature positions can be fed into a mathematical model that contains both the imaging properties of each single camera (their **intrinsic** parameters) and the mutual orientation between them (the stereo system's **extrinsic** parameters). With this model, the three dimensional position of the features with respect to a camera coordinate system can be computed [Hartley 03].

Although it is possible to perform stereo reconstruction with two arbitrarily aligned cameras, the feature matching algorithms that will be used later presume that image pairs are captured using a **canonical** stereo system. Ideal canonical stereo systems feature identical cameras with parallel image planes, parallel optical axes and colinear scan lines. This simplifies the mathematical model and allows to restrict the search space for corresponding features to *horizontal scan lines* in each image (see [Hartley 03] for more information on this topic). As a result, stereo reconstruction becomes much more efficient.

5.2.2.1 Camera Calibration and Image Rectification

In practice, perfect canonical stereo systems cannot be built. All lenses exhibit optical deficiencies and even cameras of the same type always have slightly different internal geometries. In addition, mechanical tolerances inevitably cause slight errors in the desired parallel camera alignment.

However, those deficiencies can be compensated up to a certain degree by performing *camera calibration* and *image rectification*. Camera calibration corrects non-linear image distortions caused by the optical system. A very drastic case of *radial* distortion (sometimes called a 'fish-eye' effect) and its removal is shown in figure 5.12. The correction of such a lens distortion can be performed according to the plumb bob model introduced by Brown [Brown 71]. This model approximates the entire lens system of a camera as a radially distorting lens combined with a thin prism, which causes a tangential distortion component. In reality, such a distortion occurs when a compound lens contains an imperfectly centered component.

(a) Distorted image (b) Undistorted image

Figure 5.12: An example for undistortion

The plumb bob model characterizes distortion using a set of two radial and two tangential distortion coefficients. As described in [Brown 66], known coefficients can be used to compute a non-linear pixel transformation which compensates the estimated lens distortions.

Once the non-linear optical distortions are removed, the extrinsic parameters of the stereo rig can be estimated. This allows to perform *image rectification*, e.g. to compute a linear transformation of the captured images that corrects non parallel alignments of the cameras. The mathematical details of this process are skipped, exhaustive material on this standard procedure has already been published e.g. in [Pollefeys 02] or [Hartley 03].

In the scope of this thesis, the *DLR Camera Calibration Toolbox* has been used to estimate the lens distortion coefficients and the intrinsic and extrinsic parameters of the stereo system. This toolbox was published by the Institute for Robotics and Mechatronics at the Deutsches Zentrum für Luft- und Raumfahrt e.V, Köln[3]. It is based on the work of [Weng 92] and estimates the system parameters using a series of stereo images showing different poses of a chessboard with known dimensions and many easily detectable corners.

As a result of the calibration, a rectification map is generated for each input image that contains new coordinates for every pixel of the original image. This map can be used to transform a pair of grabbed images in real time into rectified images that *appear* as if they have been produced by a canonical stereo system. Most importantly, the projections of any object are mapped onto similar horizontal lines, and thus have the same height in both images. An example of a rectified image pair is given in figure 5.13. In the figure, two differently shaped black borders can be seen around the images. These borders result from the undistortion and remapping of the original images. Also, it can be observed that corresponding elements are really aligned along the same horizontal line. The red lines provide some assistance for verification.

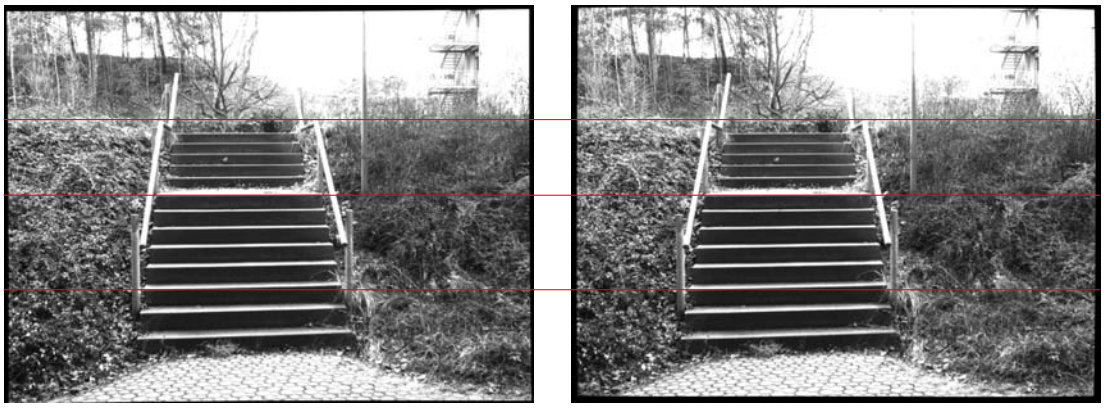


Figure 5.13: A rectified image pair

The rectified images are suitable input images for the feature matching algorithms that will be presented in the next section.

5.2.2.2 Feature Matching and Disparity Calculation

In rectified stereo images, a scene point projected at a position (x, y) in one image is projected at a position (x', y) in the second image (unless it is occluded, which is neglected for now). To recover the three-dimensional position of a scene point visible in both images, the coordinate x' must be determined by finding the best match of the projected scene point's image at (x, y) somewhere along the same horizontal scan line in the second image (see figure 5.14). If a good match is found, (x, y) and x' are sufficient for stereo reconstruction. Alternatively, (x, y) and the horizontal position *difference* $d = x - x'$ can be used. In this context, d is commonly called the **disparity** of the projected scene point.

There are two main classes of methods that allow to find matching elements in a pair of stereo images and compute the corresponding disparity values for these matches.

[3] http://www.dlr.de/rm-neu/desktopdefault.aspx/tabid-3925/6084_read-9201/

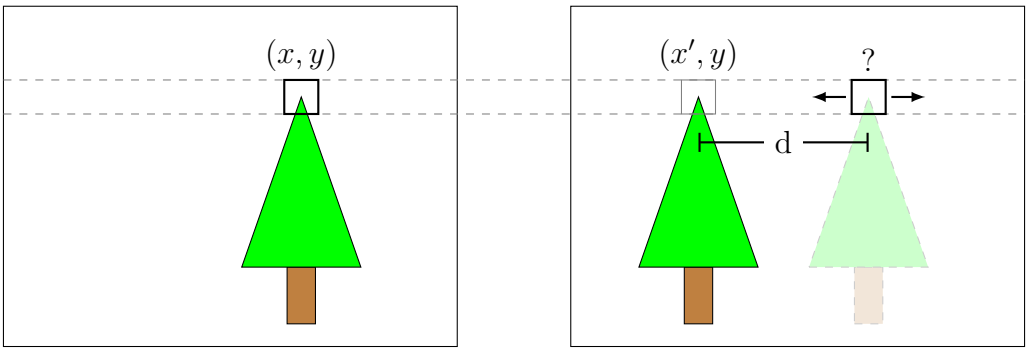


Figure 5.14: Feature matching and disparity calculation in a canonical stereo system
(Image inspired by Allan Ortiz)

Sparse disparity algorithms only compute disparities for easily recognizable image features such as object corners. Because these features are so well localized, highly accurate disparity values can be produced, often down to a sub-pixel accuracy of 0.2 pixels (using linear interpolation). Also, the algorithms are comparably fast since they only treat a subset of all image pixels. On the downside, these methods do not work at all in regions with low contrast or few identifiable features, possibly leaving out larger image areas like uniformly colored roads or walls.

In contrast to sparse stereo methods, algorithms for *dense* disparity computation try to calculate a disparity value for *every* pixel in the image plane. Compared to sparse techniques, these algorithms are typically less accurate and computationally more taxing due to the large number of pixels that have to be matched. Also, even though disparity values are produced everywhere, the matching quality can become rather low in image regions with little contrast or indistinct patterns. However, some techniques allow to incorporate reliable disparity values from adjacent regions into areas which are difficult to estimate accurately themselves. This allows to obtain at least somewhat accurate disparity data instead of none at all as provided by the sparse approaches. Table 5.1 summarizes the listed properties of both algorithm classes.

Algorithm Class	Sparse	Dense
Typical Method	Feature detection and tracking	Window based pattern matching
Coverage	Distinctive features	All matchable pixels
Covered Pixels	0.5% - 5%	≈ 100 %
Accuracy	Very High	Medium
Speed	High	Low
Treats problematic areas	No	Yes
Robust against Decalibration	Possibly	No

Table 5.1: Typical properties of sparse and dense disparity algorithms

The complementary strengths of sparse and dense disparity computation algorithms and the fact that computational effort is not a critical issue in the context of this work led to the

decision to use *both* classes of algorithms simultaneously. To construct the best possible terrain model for traversability estimation, the developed modeling algorithm preferably uses surface points detected by an accurate sparse method. If there are not sufficiently many of such points available (such as in images regions with low contrast), additional data from a dense disparity algorithm which is specifically designed to incorporate reliable information from neighboring regions if necessary is included as a fallback. By combining both methods, the probability to obtain a reasonable number of three-dimensional features in *all* areas of the image are increased.

The sparse and dense disparity algorithms that have been chosen for the developed traversability estimation approach are presented in the following sections.

Sparse Disparity Calculation

To compute sparse disparities, a two-step method is used. First, features with high contrast in two orthogonal directions (such as shown in figure 5.15) are detected in one image. Then, a method for optical flow estimation is applied to match the detected features in

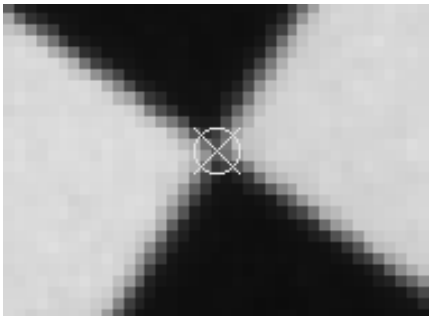


Figure 5.15: A *very* good feature to track

the second image. Once the feature correspondences have been established, several quality checks are performed to remove outliers. The remaining feature pairs are used for disparity calculation.

The first step, feature detection, is done using a method introduced by [Shi 94], appropriately called GoodFeaturesToTrack. The method detects high contrast image corners. This is done by iterating over the pixels p of the left input image and calculating the eigenvalues λ_1 and λ_2 of the covariance matrix M

$$M = \begin{pmatrix} \sum_{S(p)} (\frac{\delta I}{\delta x})^2 & \sum_{S(p)} \frac{\delta^2 I}{\delta x \delta y} \\ \sum_{S(p)} \frac{\delta^2 I}{\delta x \delta y} & \sum_{S(p)} (\frac{\delta I}{\delta y})^2 \end{pmatrix} \tag{5.2}$$

with $I(x, y)$ being the intensity channel of the input image and $S(p)$ a square neighborhood around the pixel p. After that, a non-maxima suppression step is performed on the computed eigenvalues, leaving only local maxima in a 3x3 neighborhood intact. Keeping pixels with larger eigenvalues ensures a greater distance to the image noise-level. Because

image corners with high contrast are characterized by two large eigenvalues, the remaining pixels are filtered using a quality parameter $q < 1$ and requiring that

$$min(\lambda_1, \lambda_2) > q \cdot max(\lambda_1, \lambda_2) \tag{5.3}$$

As a last step, corners too close to each other are removed, keeping the stronger corner.

q is chosen so that 0.5% - 5% of the image pixels are selected as features. If these bounds are exceeded, q is adapted accordingly to increase or decrease the amount of accepted features. However, q is not allowed to fall below a minimum of 0.01; if this yields too few features, the input image is simply unsuitable for sparse disparity calculations.

Figure 5.16 shows an example of detected features for a typical outdoor scene. It can be observed that many features are placed in vegetated regions due to their high contrast, while uniformly colored building walls are not well covered.

Figure 5.16: Typical result of sparse feature detection

In the next step, the features are matched with corresponding points in the other image. This is done using an optical flow algorithm described by Lucas and Kanade in [Lucas 81]. For a given point $(u_x, u_y)^\mathsf{T}$ in the left image plane I_l, the algorithm finds the corresponding point $(u_x + \delta_x, u_y + \delta_y)^\mathsf{T}$ in the right image plane I_r that minimizes the intensity difference ϵ of the local neighborhoods around the feature pair:

$$\epsilon(\delta_x, \delta_y) = \sum_{x\,=\,u_x - w_x}^{u_x + w_x} \sum_{y\,=\,u_y - w_y}^{u_y + w_y} \Big(I_l(x,\ y) - I_r(x + \delta_x,\ y + \delta_y) \Big) \tag{5.4}$$

This general tracking algorithm can deal with two-dimensional feature shifts and is therefore also capable of working on images that have not been rectified. It is not far-fetched to hypothesize that improved matching performance could be achieved by selecting a different algorithm that exploits the horizontal line constraint more fully and performs only an one-dimensional search. However, when using rectified images, the two dimensional tracking capability can be exploited as a very powerful filter criterion to reject bad disparity values. In fact, every feature pair that is *not* located on approximately the same scanlines in both images is most likely erroneously matched and difficult to localize correctly *in general*. Thus, the resulting disparity for that feature is probably inaccurate regardless of the used matching algorithm. Consequently, the terrain model quality can be improved by ignoring such a match.

The horizontal match constraint is checked by computing the match angle $\alpha = \arctan(\delta_y/\delta_x)$ and requiring $|\alpha| < \mu$ with a typical error threshold $\mu \approx 3°$. By allowing a certain deviation from the zero angle, the sparse disparity extraction becomes robust against small image rectification errors and minor changes in the cameras' geometrical setup. These changes can be introduced e.g. through vibrations during operation or non-rigid components inside the lenses. In experiments, such effects have been found to decrease the quality of the calibration over time and thus quickly limit the effectiveness of purely one-dimensional matching strategies.

The minimal intensity difference ϵ found for each feature is another good criterion to filter mismatches. Thus, all matches with $\epsilon > \epsilon_{max}$ are discarded.

During empirical testing, the tracking algorithm yielded approximately 75% successfully tracked points. $1-5\%$ of the detected features could not be found in the other image plane and were diagnosed as missed detections by the algorithm itself. The rest of $20-24\%$ is filtered by the μ or ϵ constraints introduced above. This rather high value results from erroneously tracked features and numerical inaccuracies for tracked features further away from the cameras, having a small disparity and therefore little room for matching errors. This percentage has also proven itself as a good indication for the quality of the camera calibration. Rising percentages indicate increasing displacements in the stereo setup and quantify the urgency of recalibration.

Dense Disparity Calculation

The desire to use the dense disparity algorithm in cases where the sparse approach is unable to provide sufficient information places a high priority on selecting an approach well capable to deal with low-contrast and difficult to match image regions. This capability is especially pronounced in the dense disparity algorithm introduced by Stan Birchfield in [Birchfield 99], which was therefore chosen for dense disparity calculation. As shown in [Schäfer 05], it is quite robust against different lighting conditions like cloudy sky, bright sunlight or even weak reflections, and there are special optimizations for handling untextured regions like roads. Also, post-processing algorithms exists which propagate reliable disparities to regions of unreliable or unavailable disparity [Birchfield 99].

An example for a *disparity map* generated with the Birchfield algorithm is shown in figure 5.17. In such a disparity map, the measured pixel disparity is encoded as brightness, rendering closer objects brighter.

Figure 5.17: Undistorted left image and resulting disparity map

Taking a closer look, it can be observed that the main objects in the scene have been analyzed correctly and even the trees in the background are visible. Although parts of the grass surface in the foreground exhibits rather low contrast, it is mapped correctly as a result of the special treatment for untextured regions. However, some errors are also introduced. Especially on the left and the top sides of the map, there are regions with bad or missing disparity information, resulting from pixels present only in one image plane or not matched in the other one. Furthermore, there is some noise around prominent object edges such as the person in the map, resulting from improper disparity propagation/post-processing. However, these image areas are distinct enough to allow a good sparse disparity analysis.

5.2.2.3 Feature Filtering

The features produced by the sparse and dense disparity algorithms contain a significant amount of incorrectly matched outliers. Many of these outliers stem from a relatively small number of typical problems related to the input images. Four main issues can be identified:

- The images have irregular black borders caused by undistortion. Features detected at or close to these borders have a very high risk of being erroneously matched and should be removed.
- Image areas with extremely low contrast are difficult to analyze even for the more tolerant dense algorithm, resulting in tracking mismatches.
- Larger, coarsely connected areas of low contrast can indicate sky, water or other areas lacking usable information. Features detected in areas with many neighboring low contrast areas have a higher risk of also being erroneous themselves.
- The used dense disparity algorithm requires a range of visible pixels before disparity matching becomes possible. This causes a tattered left border of invalid (zero) disparity values in the produced disparity map.

To filter outliers resulting from one of the problems outlined above, it is necessary to check the *image neighborhood* of the features. Since it is prohibitively expensive to construct and analyze the neighborhood of each feature one by one, 'generic' neighborhoods are

created by subdividing the image into a regular grid of **tiles** instead, as shown in figure 5.18. Each tile is quadratical with a typical size of $t_s = 40$ pixels for a 1600 by 1200 pixel image.

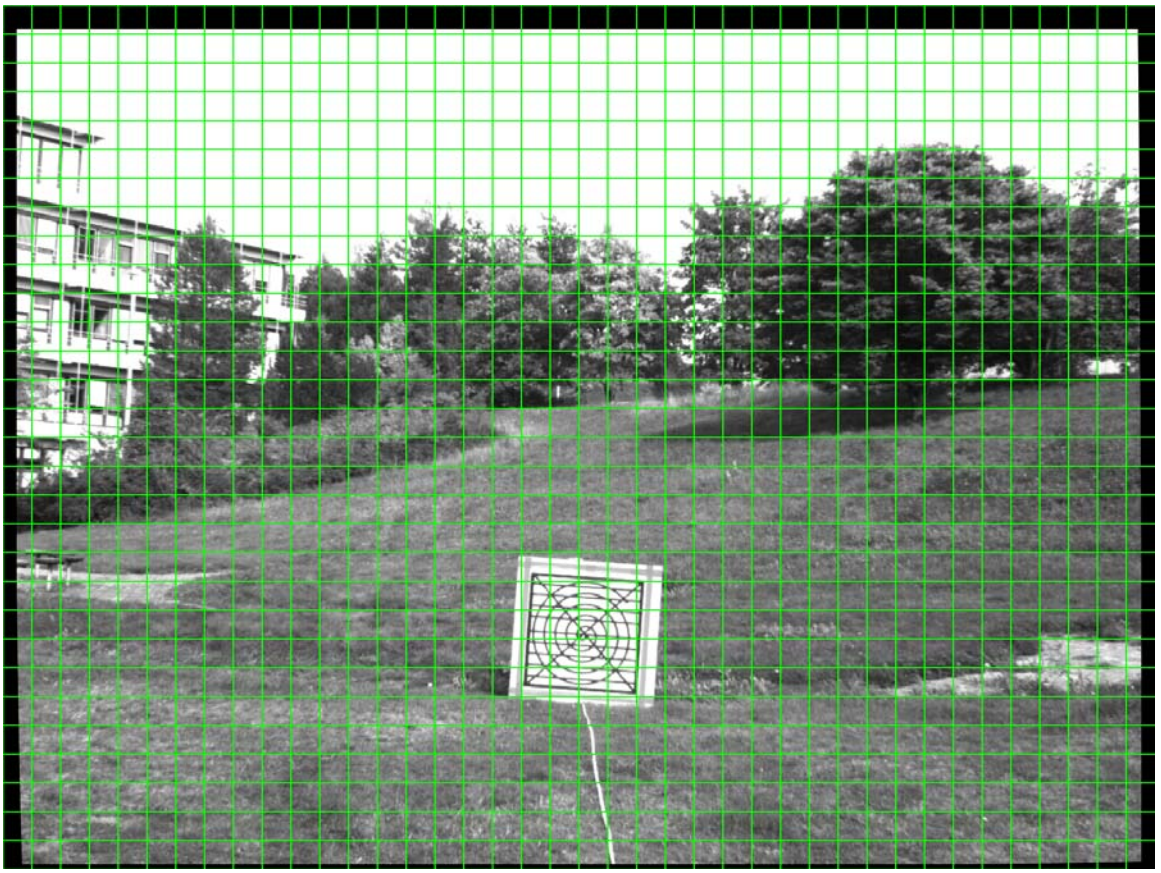


Figure 5.18: Example image subdivided into tiles

Each tile is checked against a set of quality criterions derived from the list of problems presented above. If one of these condition is not met, the tile is marked as 'unsuitable' and the features located within the tile's image region are discarded.

Undistortion Border

First, tiles overlapping with the border introduced by the undistortion process are marked as unsuitable. Because the shape of this border only changes during recalibration, a mask which excludes the tiles affected by the undistortion border can be precomputed.

Low Contrast

Correct feature matching in image areas with very low contrast is often even beyond the capability of the dense algorithm. Thus, extremely low contrast tiles are also excluded from feature generation. To detect those tiles, the standard deviation σ of the pixel intensities within an image tile is estimated. It is defined as:

$$\sigma = \sqrt{\frac{1}{t_s^2 - 1} \sum_{x=x_0}^{x_0+t_s} \sum_{y=y_0}^{y_0+t_s} (I(x,y) - \hat{I}\,)^2} \tag{5.5}$$

with

$$\hat{I} = \frac{1}{t_s^2} \sum_{x=x_0}^{x_0+t_s} \sum_{y=y_0}^{y_0+t_s} I(x,\ y) \tag{5.6}$$

being the mean value of the tile's brightness and (x_0, y_0) being the base position of the tile. A low contrast tile can be identified by comparing σ with a threshold. During empirical testing, a requirement of $\sigma > [3 \dots 7]$ provided good results as a quality constraint.

After checking each tile for sufficient contrast, isolated tiles which are marked differently than all of its neighbors are smoothened out using a single morphological opening and closing operation with a 3x3 tile structural element (see [Foreman 07] for more details on morphology operators). This step is aimed at regularizing larger areas of unusable tiles and is led by the assumption that a valid tile which is completely enclosed by unsuitable tiles is probably also error-prone and should better not be considered. A good example are the edges of darker clouds in the sky, which exhibit sufficient contrast but do not contribute any usable information due to their context.

Invalid Disparity

The last common source of outliers which needs to be treated are invalid disparity values produced by the dense disparity algorithm as a result of missing feature correspondences close to the left image border (well observable in figure 5.17, for example). In order to remove these features, which typically exhibit almost zero disparity values d, *all* features are required to have a disparity within given minimum and maximum range d_{min} and d_{max}. As will be discussed in section 5.2.2.4 (in which appropriate values for these bounds will also be derived), this limits the 3D distance of valid features down to a range which ensures a reasonable depth accuracy.

In order to filter the dense disparity values even more rigorously and also exclude matches that lie close to features with invalid disparities, any tile that contains more than 1% of features violating these bounds is marked unsuitable as well. This tile criterion is only applied for the dense disparity algorithm.

In effect, the disparity constraint filters out all of the left border tiles that contain invalid data, plus all tiles which contain features beyond the sensor range, e.g. the 'background' of each stereo image pair.

The performance of the presented tile-based filtering criterions can be observed well in figure 5.19. Tiles that have been found unsuitable are marked with different colors, depending on the criterion that was violated. Red tiles have been excluded due to the boundary constraint, yellow tiles exhibited insufficient contrast, and blue tiles contained too many features with invalid disparities, so that dense feature extraction was skipped.

To conclude this section, the different criterions introduced to filter erroneous and incorrect feature matches are summarized in table 5.2.

5.2.2.4 Triangulation

After filtering most of the erroneously detected features, the remaining match locations in both image planes and the corresponding disparity information can be used to compute the three-dimensional feature positions through *triangulation*. Initially, the 3D coordinates

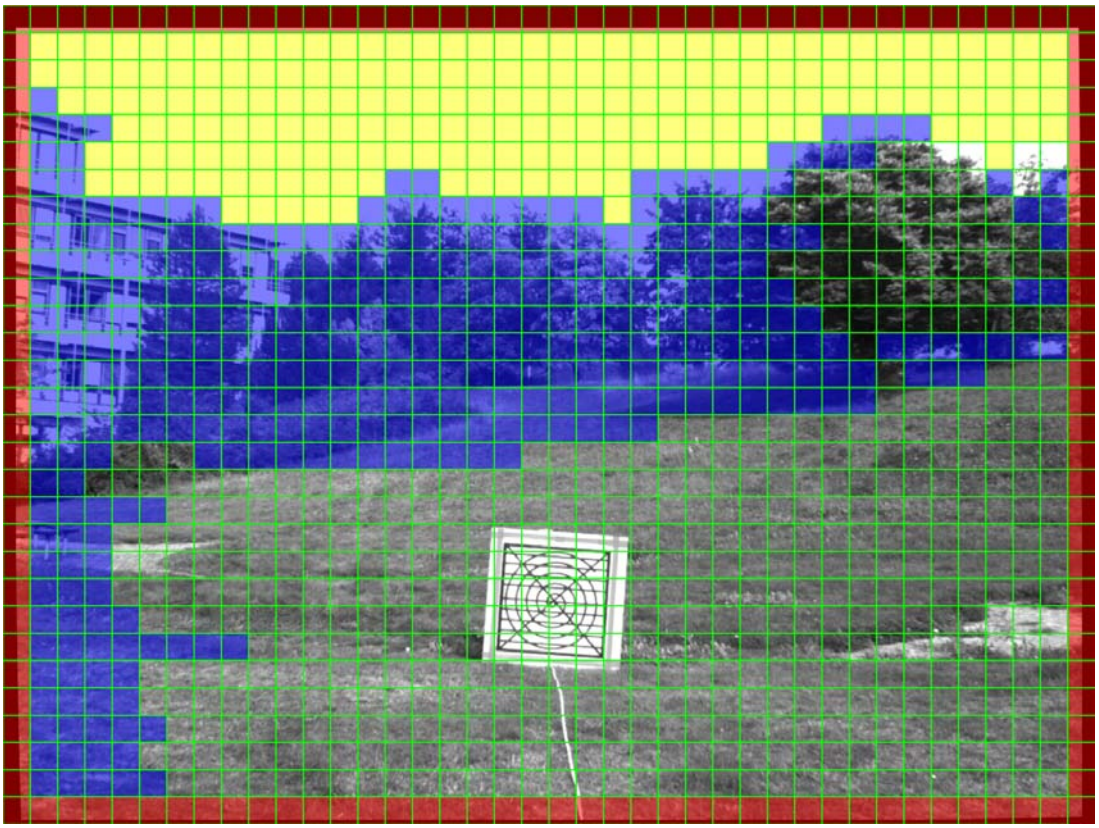


Figure 5.19: Tiles unsuitable for stereo reconstruction

are calculated in the Camera Coordinate System (CCS), which originates in the center between both cameras and whose z-axis is orthogonal to the connection line of the two cameras, pointing away from the robot. The x-axis is chosen to point horizontally to the right (from the robot's perspective). Figure 5.20 shows a schematic birds eye view of the canonical stereo camera system and the CCS.

Relevant parameters for triangulation are the horizontal displacement b of the two camera projection centers C_l and C_r, commonly called the **baseline** of the stereo system. The distance of the centers from the image planes (red) is called the **focal length** f, while the intersection of a camera imaging axis with the respective image plane determines the **principal point** of that camera (P_l, P_r). Given the projections of a scene point $S(x, y, z)$

Filter Criterion	Applicability Sparse	Dense
Sufficient match similarity ($\epsilon < \epsilon_{max}$)	Yes	No
Low vertical match slope ($\|\alpha\| < \mu$)	Yes	No
In tile outside undistortion border	Yes	Yes
In tile with sufficient contrast ($\sigma > [3 \dots 7]$)	Yes	Yes
Valid feature disparity ($d_{min} < d < d_{max}$)	Yes	Yes
In tile with ≥99% valid disparities	No	Yes

Table 5.2: Summary of filter criterions for matched feature pairs

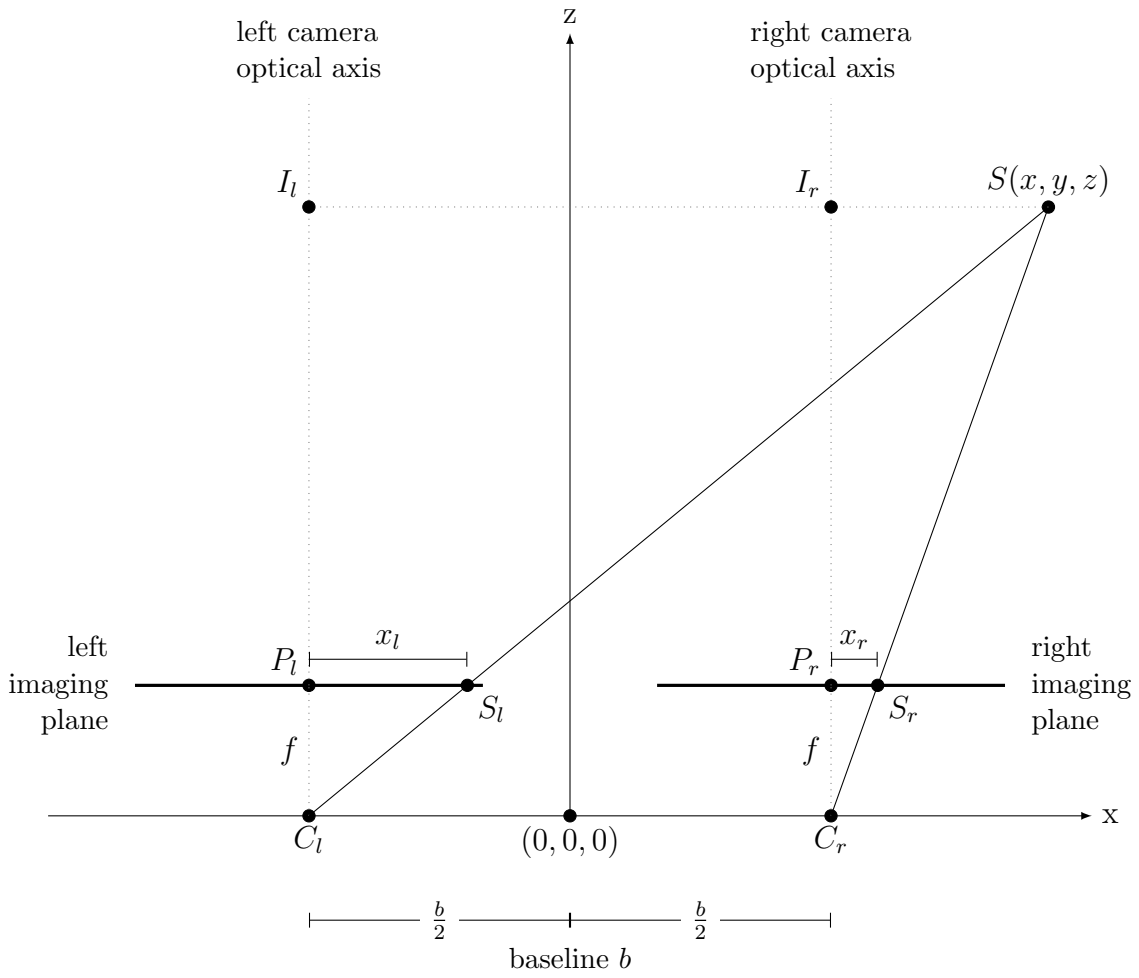


Figure 5.20: Birds eye view of a canonical stereo system

in the left image plane S_l and the right image plane S_r with coordinates (x_l, y_l) and (x_r, y_r) relative to the principal points, the already introduced disparity d of the point S is found again as $d = x_l - x_r$.

Given the canonical setup in figure 5.20, the depth z of S can be calculated by exploiting *triangle similarities*. For the left camera, the similar triangles $S_l P_l C_l$ and $S I_l C_l$ give

$$\frac{x_l}{f} = \frac{x + \frac{b}{2}}{z} \tag{5.7}$$

For the right side, the similar triangles $S_r P_r C_r$ and $S I_r C_r$ result in

$$\frac{x_r}{f} = \frac{x - \frac{b}{2}}{z} \tag{5.8}$$

The combination of equations 5.7 and 5.8 yields the distance z:

$$z = \frac{bf}{x_l - x_r} \tag{5.9}$$

Substituting equation 5.9 into eq. 5.7 allows to calculate the x coordinate of the scene point S as

$$x = \frac{b(x_l + x_r)}{2(x_l - x_r)} \tag{5.10}$$

For the y coordinate, the application of the triangle similarities in the y-z plane yields equation

$$\frac{y_l}{f} = \frac{y}{z} = \frac{y_r}{f} \tag{5.11}$$

Combining equations 5.11 and 5.9, the y coordinate for S can be calculated as

$$y = \frac{by_l}{x_l - x_r} \tag{5.12}$$

In summary, the three dimensional position (x, y, z) of the scene point S can be derived from the known projections S_l and S_r with two dimensional coordinates (x_l, y_l) and (x_r, y_r) relative to the image planes by

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \frac{b(x_l + x_r)}{2(x_l - x_r)} \\ \frac{by_l}{x_l - x_r} \\ \frac{bf}{x_l - x_r} \end{pmatrix} = \frac{b}{x_l - x_r} \begin{pmatrix} \frac{x_l + x_r}{2} \\ y_l \\ f \end{pmatrix} \tag{5.13}$$

Sensitivity Analysis

The derived formulas reveal one major drawback of stereo reconstruction. Unfortunately, the depth resolution (depth sensitivity) decreases rapidly with growing distance to the camera head. This becomes obvious when taking the partial derivative of the depth z with respect to the disparity $d = x_l - x_r$ in equation 5.9, which results in:

$$\frac{bf}{d} \frac{\delta z}{\delta d} = -\frac{bf}{d^2} = -\frac{z^2}{bf} \tag{5.14}$$

According to this, the absolute sensitivity of the depth calculation decreases with z^2. As an example, let's assume a stereo system with a baseline of 20 cm and a focal length of 200 pixels [4]. For objects at a distance of 1 m, the depth difference per pixel of disparity is about 2.5 cm / pixel. Doubling the distance to 2 m, the depth difference per disparity step is *quadrupled* to about 10 cm / pixel. In a distance of 10 m, it is already as high as 2.5 m / pixel, effectively making precise depth recovery impossible. Thus, the depth accuracy of a stereo camera system is distributed rather unequally over the range of the system. Also, even disparity differences that are still easily resolvable by the matching algorithms will lead to significant depth jumps. Any terrain modeling approach will need to cope with this behavior.

[4] Given a calibration object with known dimensions, the camera calibration process allows to determine the metric size of a single image pixel and thus, to convert focal lengths given in metrical units into an equivalent length in pixels. This eases computation, as all other measures are also available in pixels.

For the available camera setup with a baseline of ≈ 0.5 m and a focal length of ≈ 1893 pixels, the maximum working distance with a still reasonable sensitivity < 1 m / pixel can now be estimated as

$$z_{max} \approx \sqrt{1\frac{\text{m}}{\text{Pixel}} \cdot 1893 \text{ Pixel} \cdot 0.5 \text{ m}} \approx 30.7 \text{ m} \tag{5.15}$$

The disparity corresponding to this maximum distance is ≈ 31 pixels, and thus the minimal acceptable disparity d_{min} can be set to 31. To increase the maximum range, the baseline b or the focal length f could be increased, but this introduces new complications such as higher algorithm runtime, a decreased probability of finding corresponding pixels in the image plane for close objects or mechanical problems for the current pan/tilt unit. Therefore, the maximum upper limit is retained.

The limit in the other direction is dictated by the fact that the utilized Birchfield implementation has a maximum disparity search range d_{max} of 255 pixels. With equation 5.9, this value can be used to calculate the minimum z distance that the algorithm can handle with the used stereo setup:

$$z_{min} \approx \frac{1893 \text{ pixel} \cdot 0.5 \text{ m}}{255 \text{ pixel}} \approx 3.7 \text{ m} \tag{5.16}$$

All objects closer than z_{min} are thus not considered during dense disparity calculation. This poses no significant problem, as close range obstacle information is available from the local obstacle memory up to a range of ≈ 8 meters.

Transforming CCS coordinates into RCS

The last step of point cloud generation is to transform the obtained three dimensional CCS point coordinates into the Robot Coordinate System RCS. From there, the transformation matrices into ECS or a suitable node coordinate system NCS are already known and can be performed easily.

Figure 5.21 shows a schematic drawing of the robot and the kinematic chain of the stereo camera system. Included are the physical dimensions of each link (in the same unit as the point coordinates) plus the source coordinate system CCS (red) and the target coordinate system RCS (blue).

Based on this diagram, the homogeneous transformation matrix M_{CCS}^{RCS} can be constructed using a concatenation of elementary transformation matrices by:

$$M_{CCS}^{RCS} = T_1 \cdot T_2 \cdot R_{pan} \cdot R_{tilt} \cdot T_3 \cdot R_{axis} \tag{5.17}$$

In this sequence, R_{axis} is responsible for rotating the CCS axes into alignment with the RCS axes, effectively mapping CCS $\vec{z}$ onto RCS $\vec{x}$, CCS $\vec{y}$ onto RCS $-\vec{z}$ and CCS $\vec{x}$ onto RCS $-\vec{y}$. R_{pan} and R_{tilt} represent the rotation effected by the pan/tilt unit and are

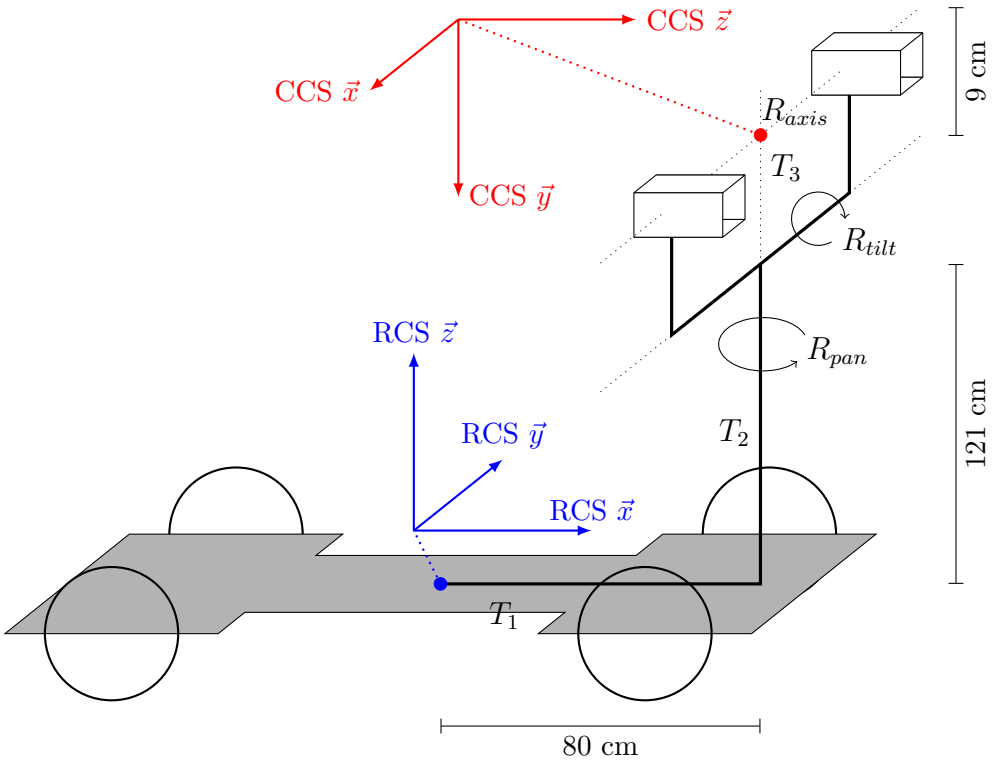


Figure 5.21: Camera coordinate system CCS and robot coordinate system RCS

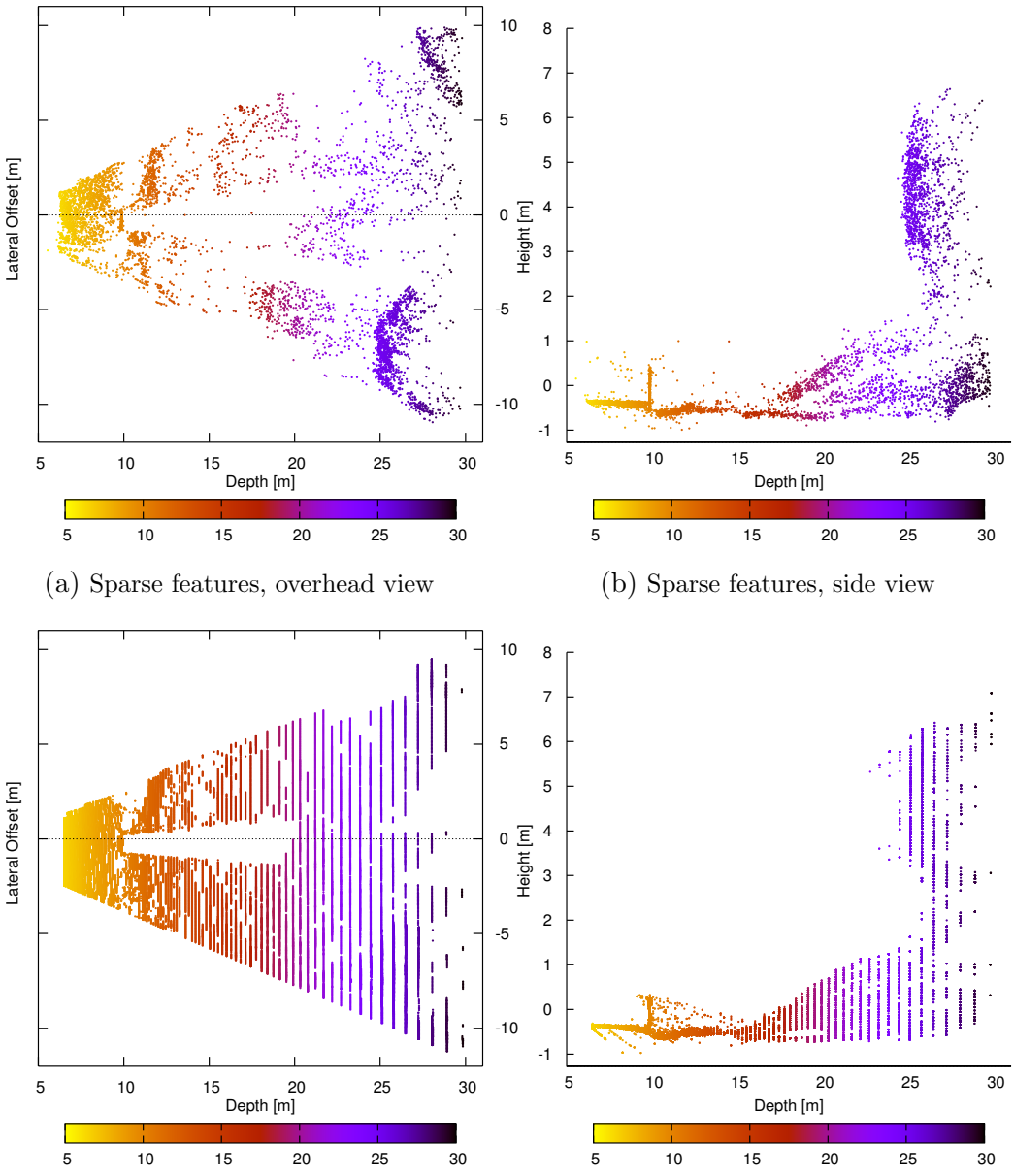
parameterized by the pan angle γ and the tilt angle β. Explicitly written, the matrix concatenation appears as:

$$M_{CCS}^{RCS} = \overbrace{\begin{pmatrix} 1 & 0 & 0 & 80 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 121 \\ 0 & 0 & 0 & 1 \end{pmatrix}}^{T_1 \cdot T_2} \cdot \overbrace{\begin{pmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}^{R_{pan}} \cdot \overbrace{\begin{pmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}^{R_{tilt}} \cdot \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 9 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{T_3} \cdot \underbrace{\begin{pmatrix} 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{R_{axis}} \tag{5.18}$$

For a point S in CCS, the RCS coordinates S' can now be calculated by applying the homogeneous transformation matrices

$$S' = M_{CCS}^{RCS} \cdot S \tag{5.19}$$

Figure 5.22 shows the point cloud reconstruction of the example scene depicted in figure 5.18. The reconstructed features are depicted separately for each reconstruction algorithm,



(a) Sparse features, overhead view

(b) Sparse features, side view

(c) Dense features, overhead view

(d) Dense features, side view

Figure 5.22: Reconstructed features

and a view from above the scene and a side view is presented. In all plots, the robot coordinate system originates at $(0, 0, 0)$.

In both side view plots, the vertical calibration chart in the image foreground can be identified clearly, along with the rising terrain behind it and the treetop close to the limit of the system range. Furthermore, the different accuracies and outlier rates of the sparse reconstruction algorithm compared to the dense algorithm becomes visible. Especially at longer ranges, the smallest possible disparity changes form distinct depth steps in the features reconstructed by the dense disparity algorithm. Nevertheless, the merged point cloud, created by combining all reconstructed points from both sparse and dense

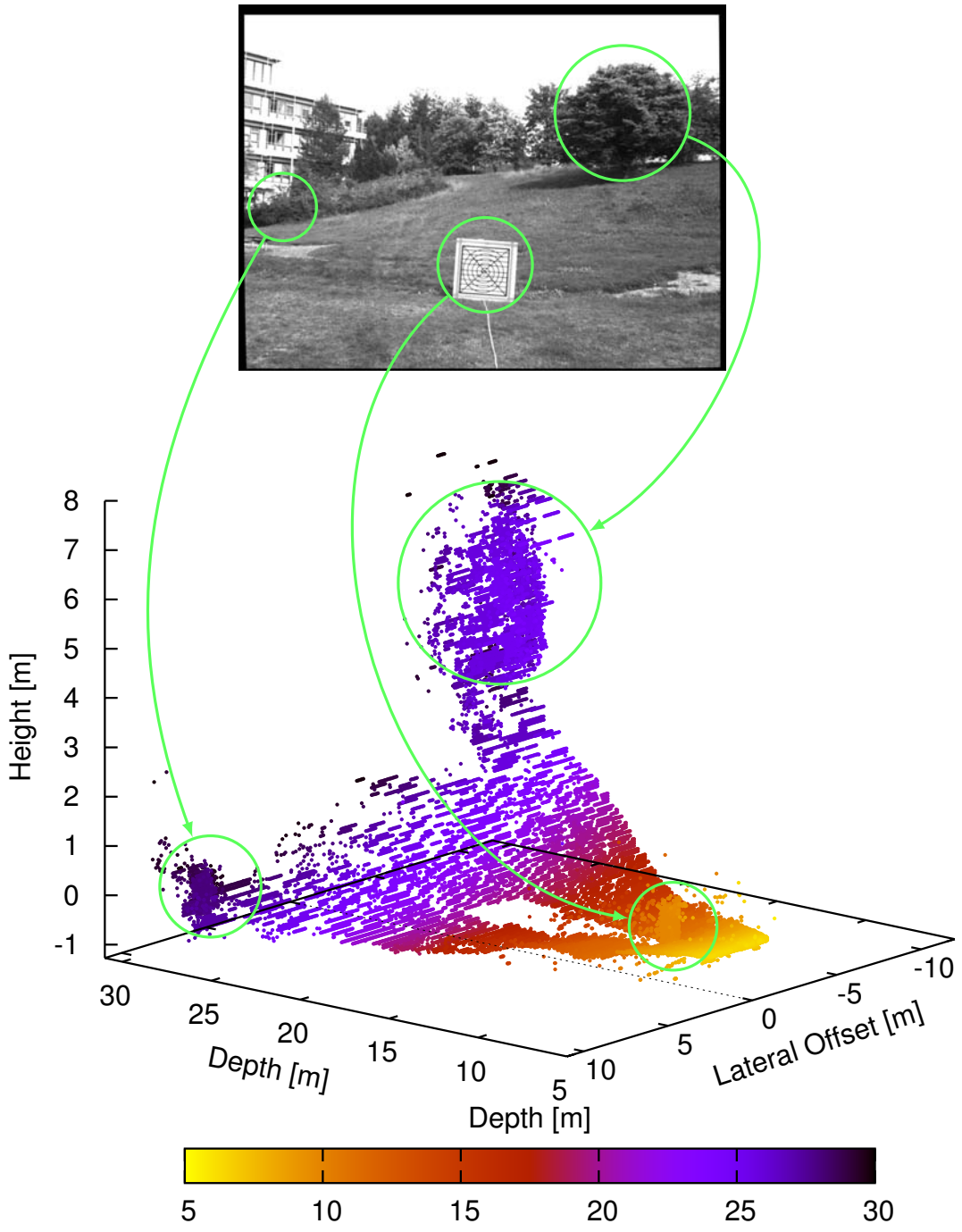


Figure 5.23: Point cloud containing sparse *and* dense features (view from top left)

reconstruction, shown in figure 5.23 captures the relevant terrain outline quite precisely, in a range from ≈ 5 to 30 meters.

Merging Point Clouds from Multiple Images

The process presented above produces a cloud of points as a result of a single image acquisition. To cover a larger area of the surrounding terrain, several images with different settings of the pan/tilt unit can be reconstructed and merged with each other in the RCS coordinate system. The following traversability estimation steps can then process a single,

large point cloud and abstract from the sequential construction. The number of images can be chosen as desired. For the used stereo system setup, a value of 7 images per sweep has proven sufficient to cover the available working space of the pan unit without introducing gaps between the different shots.

As a final remark, the transformation values as well as the angles calculated by the pan/tilt units motor movements naturally include a certain measurement error. Consequently, the 3D position of each reconstructed point differs from the exact position of the corresponding feature in the scene. These effects are not considered further, because traversability estimation does not require an overly exact, *absolute* reconstruction of feature positions. Instead, the relative arrangement of points in relation to each other are much more interesting. Subjectively, for the working range up to 30 m, these relations are sufficiently well captured with the presented approach. An objective, quantitative evaluation of this impression requires a carefully surveyed or constructed outdoor scene and is deemed future work.

5.2.3 Terrain Modeling

After a three-dimensional point cloud outlining the terrain surface has been constructed, the reconstructed points can be abstracted into a representational form which supports reasoning about terrain traversability. A popular model for this is the digital elevation map (DEM) [Cremean 06] [Pollefeys 01]. Recapitulating the exposition in section 3.1.1.1, DEMs are grid maps storing the overall terrain height of a certain metrical patch (e.g. 1 m x 1 m). By considering height differences between neighboring cells, traversability scores can be computed. However, details such as the 'real' slope of the patch are lost and a distinction between patches containing abrupt steps and those representing continuous slopes becomes impossible.

A more accurate terrain model can be produced by fitting planes [Weingarten 03] or triangular meshes [Gemme 05] to the terrain surface. These planes can approximate gently sloped terrain much better than elevation maps. However, the plane fitting requires more accurate information for reliable operation than the construction of a DEM patch. Therefore, this model has been predominantly used for indoor applications [Weingarten 03], which require shorter ranges and thus obtain point clouds with higher densities.

The terrain model developed in this thesis extends the state-of-the-art methods and adaptively combines *both* terrain model types in order to benefit from their respective strengths. More precisely, areas exhibiting a high density of accurately reconstructed points are approximated with **sloped planes** that retain the crucial information about predominant slopes. In areas unsuitable for this procedure due to a lack of points or inherent non-planarity, the method falls back to elevation map modeling using **elevation planes** without slope information. The goal of this strategy is to increase the range of the vision-based traversability estimation system in comparison to a pure plane fitting approach. At the same time, nearby terrain is modeled more accurately than a pure DEM technique would allow, enabling the robot to estimate terrain traversability with higher precision.

Spatial Segmentation

Both representational forms model the surrounding terrain piece by piece and therefore depend on a spatial segmentation of the obtained point cloud. Because the integration

of the final results into the existing cost estimation mechanism is best accomplished by generating an additional *local traversability map* with the layout described in section 4.1.3.1, the point cloud is segmented according to the sector / seclet structure introduced there. Given a topological node to which the derived traversability information shall be attached, the obtained point cloud is thus first transformed from RCS (via ECS) into the node coordinate system of the selected node. Then, all reconstructed 3D points are assigned to their corresponding seclets according to the points' x and y NCS coordinates. For each seclet s, two point sets are created: one set M_s of features stemming from the accurate, *sparse* feature reconstruction algorithm and another set M_d with features that were derived with the less accurate, *dense* reconstruction method.

Figure 5.24 exemplarily shows the segmentation of the sparse feature point cloud depicted in figure 5.22a. For this segmentation, the associated topological node (marked with a black dot) is placed directly at the position where the image pair was originally captured, so that RCS and NCS origins coincide. The back 180° of the map and the seclets with distance $0 - 2$ m are omitted to increase readability.

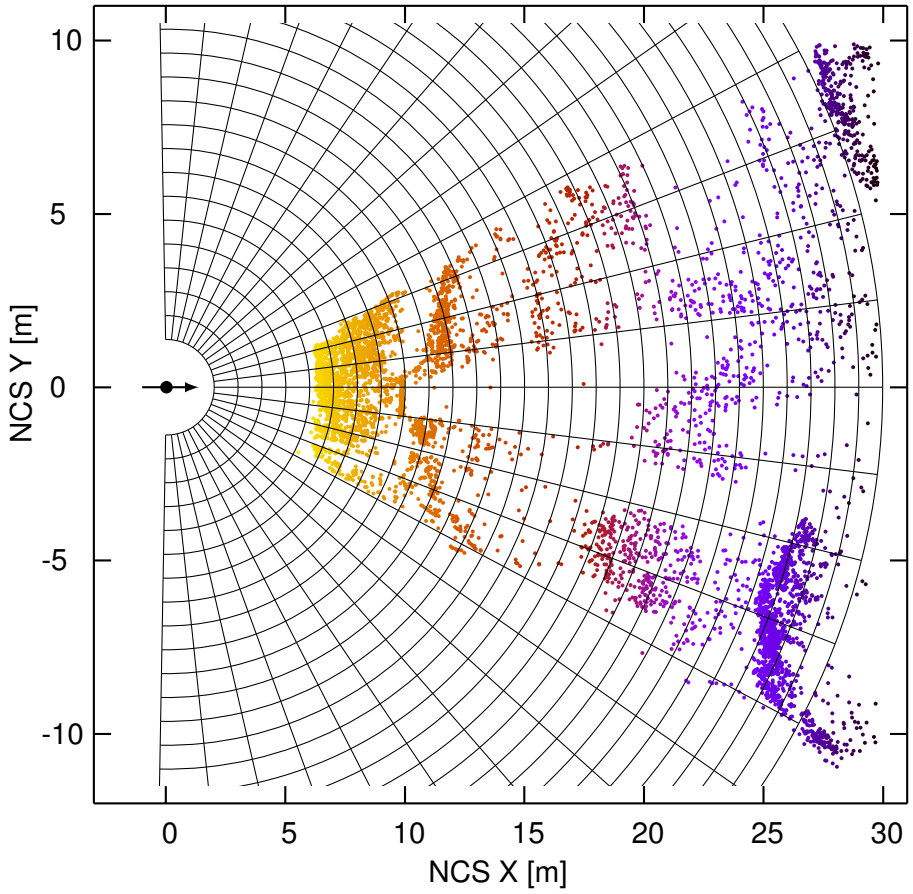


Figure 5.24: Spatial segmentation into seclets

5.2.3.1 Seclet Approximation with a Sloped Plane

After point distribution, each seclet is considered in turn. Seclets with a sparse point set $M_s = \{\vec{p_0}, \ldots, \vec{p_n}\}$ containing more points than a threshold κ_s are considered as candidates for approximation with a sloped planar surface. In order to guard against outliers

in M_s, the approximation is implemented using a plane fitting approach that employs the RANSAC algorithm (which was already used for outlier-robust linear regression in section 4.1.1). This technique has already been used for planar representation of walls in indoor environments and is extensively described in [Weingarten 03] and [Wettach 07]. The RANSAC formulation of the planar approximation technique proceeds as shown in the pseudo-code algorithm 10.

Algorithm 10: ApproximatePointSetWithSlopedPlane

```
Data: Point3DSet M_s of features from sparse reconstruction
// planes are represented as a pair:
// (Point3D center_of_gravity, Vector3D plane_normal)
Result: (Point3D center, Vector3D normal) of the best fitting plane for M_s
begin
    Point3DSet R_max ← ∅
    Int r_max ← 0
    for i = 0 to num_ransac_iterations do
        Randomly select three points p1, p2, p3 ∈ M_s
        if p1, p2, p3 are not colinear then
            Construct plane (c, n) from p1, p2, p3
            R = all points ∈ M_s with distance to plane (c, n) < d_max
            if |R| > r_max then
                R_max = R
                r_max = |R|
            end
        end
    end
    // construct least-squares optimal plane from largest inlier set R_max
    (c_best, n_best) = FitPlaneToPointSet (R_max)
    return (c_best, n_best)
end
```

Algorithm 10 considers points with a distance smaller than a threshold d_{max} are considered inliers and added to the inlier set, all other points are classified as outliers. After several repetitions of this classification process, the plane supported by the biggest inlier set R_{max} is selected. Finally, its pose is recomputed using a least-squares fitting of *all* inliers.

Parameter Selection

In addition to the input point cloud, the RANSAC algorithm depends upon the two additional parameters `num_ransac_iterations` and d_{max}, which need to be set appropriately.

A higher number of iterations for the RANSAC algorithm increases the probability of finding planes with a high number of inliers, but also increases the total runtime costs. Given a desired probability p of taking at least one sample with all three points $\vec{p_1}$, $\vec{p_2}$, $\vec{p_3}$ being inliers, a value for `num_ransac_iterations` can be calculated as discussed in [Hartley 03]. The main prerequisite for this is the proportion ϵ of *real* inliers to outliers

in the dataset. If this ratio is known, the number of iterations N to perform can be calculated as

$$N = \frac{\log(1-p)}{\log(1-(1-\epsilon)^3)} \tag{5.20}$$

The proportion of inliers to outliers varies strongly depending on the geometrical shape which is represented by the points in a given seclet. While a flat groundplane is expected to contain only a few outliers, a bushy treetop will strongly increase the probability of randomly choosing an outlier for the sample set.

Since 'correct' estimates for ϵ are difficult to obtain, a conservative value of $\epsilon = 50\%$ is assumed. According to this assumption and equation 5.20, a total of 35 iterations is sufficient to construct a valid plane with a probability over 95%.

The inlier distance threshold d_{max} has been estimated empirically. A range of $15-40$ cm yielded the best results during experiments.

Implementation of the FitPlaneToPointSet Method

The FitPlaneToPointSet method constructs a plane $(c_{best}, \vec{n}_{best})$ which minimizes the sum of the squared distances between the plane and all of the inlier points $\vec{p_i} \in R_{max}$. First, the plane is expressed in its Hesse Notation

$$\vec{u} \cdot \vec{p_i} - d = 0 \tag{5.21}$$

which can also be noted as

$$u_x x_i + u_y y_i + u_z z_i - d = 0 \tag{5.22}$$

with $\vec{u} = (u_x, u_y, u_z)^\mathsf{T}$ and $\vec{p_i} = (x_i, y_i, z_i)^\mathsf{T}$.

By requiring $d \neq 0$ (which can easily be ensured by relocating the origin of the coordinate system), this can be rewritten as

$$n_x x_i + n_y y_i + n_z z_i + 1 = 0 \tag{5.23}$$

with $\vec{n} = (n_x, n_y, n_z)^\mathsf{T}$ being the normal vector for the plane. Using this equation, the objective is to minimize S in

$$S = \sum_{i=1}^{|R_{max}|} (n_x x_i + n_y y_i + n_z z_i + 1)^2 \tag{5.24}$$

To find the minimum S, the sum in equation 5.24 needs to be partially differentiated with respect to n_x, n_y and n_z and set to zero afterwards, resulting in

$$\vec{n}_{best} = A^{-1} \cdot b \tag{5.25}$$

with

$$A = \begin{pmatrix} \sum x_i^2 & \sum x_i y_i & \sum x_i z_i \\ \sum x_i y_i & \sum y_i^2 & \sum y_i z_i \\ \sum x_i z_i & \sum y_i z_i & \sum z_i^2 \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} -\sum x_i \\ -\sum y_i \\ -\sum z_i \end{pmatrix}. \tag{5.26}$$

This equation can be solved using standard linear algebra techniques.

After determining the plane normal, the center of gravity c_{best} for the plane can be calculated as follows:

$$c_{best} = -\frac{1}{|R_{max}|} \cdot \vec{b} \tag{5.27}$$

A more detailed discussion of this algorithm can be found in [Weingarten 04] or [Wettach 07].

Planarity Check

After fitting the least-squares optimal plane to M_s, it becomes possible to check whether a planar approximation was indeed reasonable for this point set. This check is based on the ratio $r = |R_{max}|/|M_s|$ between the number of RANSAC inliers $|R_{max}|$ and the cardinality of M_s. If the ratio falls below a level r_{min} (typically set to 0.6), it is assumed that the seclet's point set actually describes a distinctively *non-planar* surface (such as a tree top) and a linear representation would be overly simplistic and inaccurate. Consequently, the sloped plane is discarded and the seclet is approximated using an elevation plane only. This prevents the subsequent traversability estimation algorithm from placing false confidence into the inappropriate slope information.

Combination with Predecessor / Successor Seclet

With increasing distances from the camera position, the number of reconstructed sparse points in each seclet tends to decrease. A major cause for this effect is the constant seclet length of 1 m, which leads to a reduced image footprint of seclets at larger distances due to perspective shortening. This effect is especially pronounced in flat terrain because of the very low viewing angle.

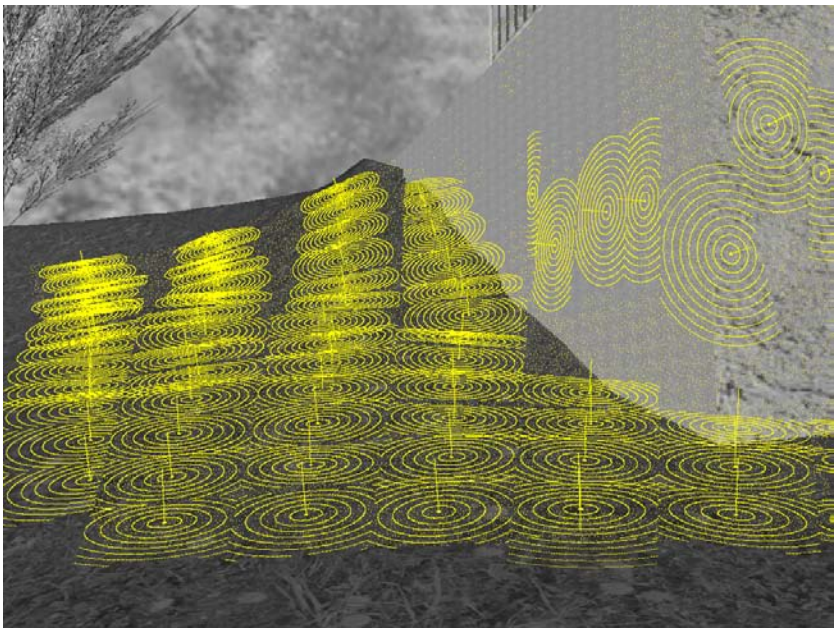
In order to reduce this problem and increase the total number of seclets that can be modeled with a sloped plane, a 'rescue attempt' is performed for seclets that do not fulfill the $|M_s| < \kappa_s$ constraint, but still contain *some* points (checked with $|M_s| \geq \kappa$ against the threshold κ introduced in the next section). This rescue attempt is implemented as follows: First, the point set M_s is tentatively combined with the point set M_s' of the *predecessor* seclet, lying in the same sector, but 1 m closer to the map origin. If the combined set holds more than κ_s points, the described plane fitting algorithm is executed using this set. Then, the inlier ratio is calculated for the resulting plane using the seclet's *own* points only, effectively computing the fraction of points in M_s with a distance of at most d_{max} to the plane. This procedure is then repeated with the current seclet's *successor*, again lying in the same sector, but 1 m further away from the map origin. Finally, the plane with the higher inlier ratio is used to model the current seclet, *if* that inlier ratio exceeds r_{min}. If this condition is not met or the combined sets also contained too few points for plane construction, the rescue attempt has failed. In this case, it is tried to model the seclet with an elevation plane as discussed in the next section.

Example

Figure 5.25 contains an example for the results obtainable by the sloped plane approximation method.

(a) Overhead view

(b) Robot perspective



(c) Fitted planar approximation

Figure 5.25: Simulated scene with fitted sloped planes

Figures 5.25a and 5.25b show an overhead view and the robot's perspective of an example scenario generated with the SimVis3D simulation framework. In this scene, the robot is placed near the foundations of a bridge and senses both the steep slope rising upwards besides the bridge and the vertical bridge support. Figure 5.25c displays the result of plane fitting using the source image from figure 5.25b. The extracted planes are visualized using a series of concentric circles which lie within the respective plane and are centered at its center of gravity (COG). The circles are clipped against the corresponding seclets borders and backprojected into the image to allow easy visual matching with the analyzed scene.

The result shows that the piecewise planar approximation captures both the geometrical shape of the sloped terrain and the vertical planar man-made structure well. As will be presented in section 5.3, the quality of this result is comparable to the performance achieved in real-world situations.

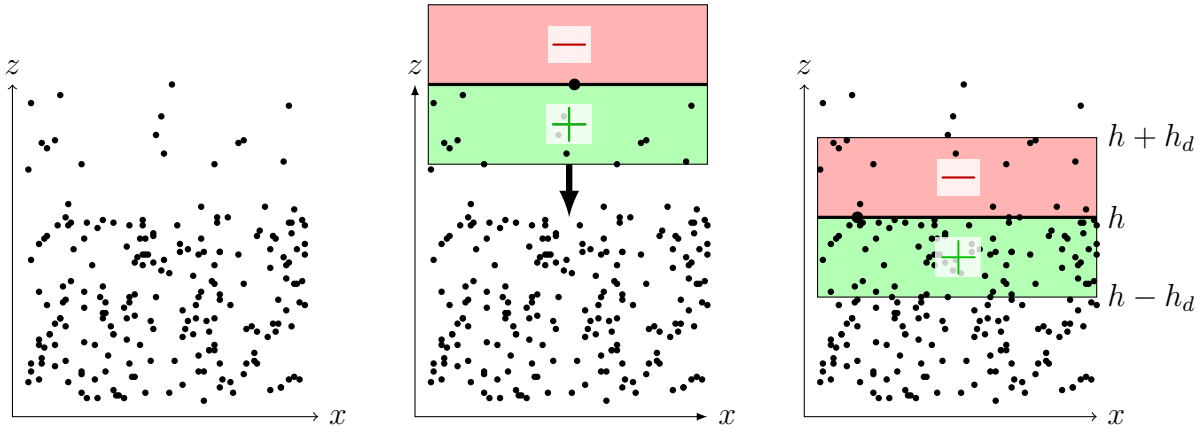
5.2.3.2 Seclet Approximation with an Elevation Plane

If a seclet was not modeled using a sloped plane due to either the $|M_s| > \kappa_s$ or the $r \geq r_{min}$ constraints and also could not be rescued by adding points from the neighboring seclets, the terrain modeling algorithm falls back to using an elevation plane containing only a height value and no slope information. Because the exact location of a point is not as relevant for this model as for a sloped plane, elevation plane construction can be performed using the combined point set $M = M_s \cup M_d$. This set encompasses both the accurately localized points from the sparse stereo algorithm *and* the less accurate data reconstructed using the dense disparity algorithm. Of course, the elevation plane model

also requires a minimum number of points to work reliably. Therefore, if fewer points than κ with $\kappa < \kappa_s$ are available in M, the seclet is completely excluded from terrain modeling.

For all other seclets, the elevation plane's height h must be determined in a way that is again robust against the presence of outliers. Therefore, it is not sufficient to set h equal to the maximum of all point z coordinates in M. Instead, a technique has been developed which determines the most pronounced 'density change' in the seclet's point set. The rationale behind this approach is the assumption that a stable surface is characterized by a high point count near the surface level and a much lower point number above it.

The detection of such a density change is achieved using a **difference operator** with a selectable height h_d, typically chosen in the range of $20 - 40$ cm. To determine the optimal operator placement given a seclet's point cloud M, the operator is placed at the height z of each point $\vec{p} \in M$ in turn. For each position, the number of points lying closely *below* the difference operator level is counted: $n_{pos} = \|\{(x_i, y_i, z_i)^\mathsf{T} \in M \mid z - h_d \leq z_i \leq z\}\|$. From n_{pos}, the number of points closely *above* the operator level is subtracted: $n = n_{pos} - \|\{(x_i, y_i, z_i)^\mathsf{T} \in M \mid z < z_i \leq z + h_d\}\|$, resulting in the final score n for this operator placement z. After iterating the operator position over all points in M, the height that resulted in the highest score is chosen as the height h of the elevation plane.



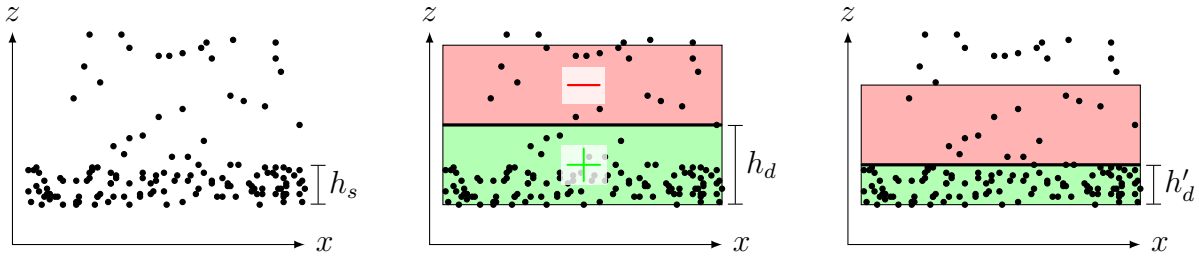
(a) Noisy point cloud (b) Begin of height search (c) Optimal operator height

Figure 5.26: Determining the optimal elevation plane height in a noisy point cloud

Figure 5.26 shows an example of the process. A very noisy point cloud with a high point density at the lower part and a low point density above is shown in figure 5.26a. Such a point cloud can be seen as a typical data set for a seclet containing an obstacle (like a hedge) with a flat surface observed from a frontal direction. Now, in order to find the right surface level, the difference operator is iteratively moved over all point heights and the score n is calculated for each position. Figure 5.26b shows the begin of the search, with the operator anchored at the highest point in the cloud. This position (which is identical to the naive maximum height strategy) results in the score $n = 14$. However, the maximum score is indeed found at the intuitively optimal placement shown in figure 5.26c. As desired, the difference operator approach effectively discards sporadic outliers that have high z-values without an underlying foundation of additional points.

'Settling' the Elevation Plane on Thin Surfaces

Although the developed difference operator is able to detect the most prominent density increase in a noisy point cloud with high reliability, the position with the highest score can be offset a bit ($0 - h_d$ cm) above the optimal level for point clouds with a special point distribution. This effect occurs when the point cloud models a *thin surface*, e.g. a surface whose points are distributed within a height range h_s less than the operator width h_d (see figure 5.27a for an example). In such a case, the operator is not pulled below the lower rim of the thin surface, since there are no (or very few) further points in that area that could contribute positively to the operator score. Consequently, the operator obtains the highest possible score at the position depicted in figure 5.27b, with the positive score region ending directly at the lower edge of the thin surface and the operator level proturing $h_d - h_s$ *above* the real surface.



(a) Noisy thin surface (b) Current operator placement (c) Settled operator

Figure 5.27: Settling the elevation plane onto thin surfaces

In order to correct this, the height estimation algorithm for the elevation plane is augmented with a second step. Initially, the difference operator is placed at its preliminary optimal position h as described before. Now, the operator is 'settled' on the real surface by reducing the height of the positive support region size iteratively in small steps and moving the operator downward accordingly. In each step, the operator score is recomputed. However, the reduced positive region size is compensated by extrapolating the point density *within* the remaining area (with height h'_d) onto the original size h_d by computing $n_{pos} = (h_d/h'_d)n_{pos}$. In effect, this allows the operator to increase the contained point *density* (and thus the score n) in the positive region step by step while shrinking it down to the size h_s of the thin surface. Finally, the best score is obtained at the real surface level as shown in figure 5.27c. With this extension, the elevation plane height is placed at the correct level even in cases where the point cloud models a compact, thin surface.

5.2.3.3 Conclusion

To conclude the presentation of the developed terrain modeling algorithm, the control flow of the entire method is shown in figure 5.28.

This flowchart summarizes the different approximations and checks that are done in order to generate a planar model for a seclet, given two point clouds from stereo reconstruction. There are three possible outcomes for the model: A sloped plane, an elevation plane or no plane at all. The decision which one of these models is used depends upon the accuracy

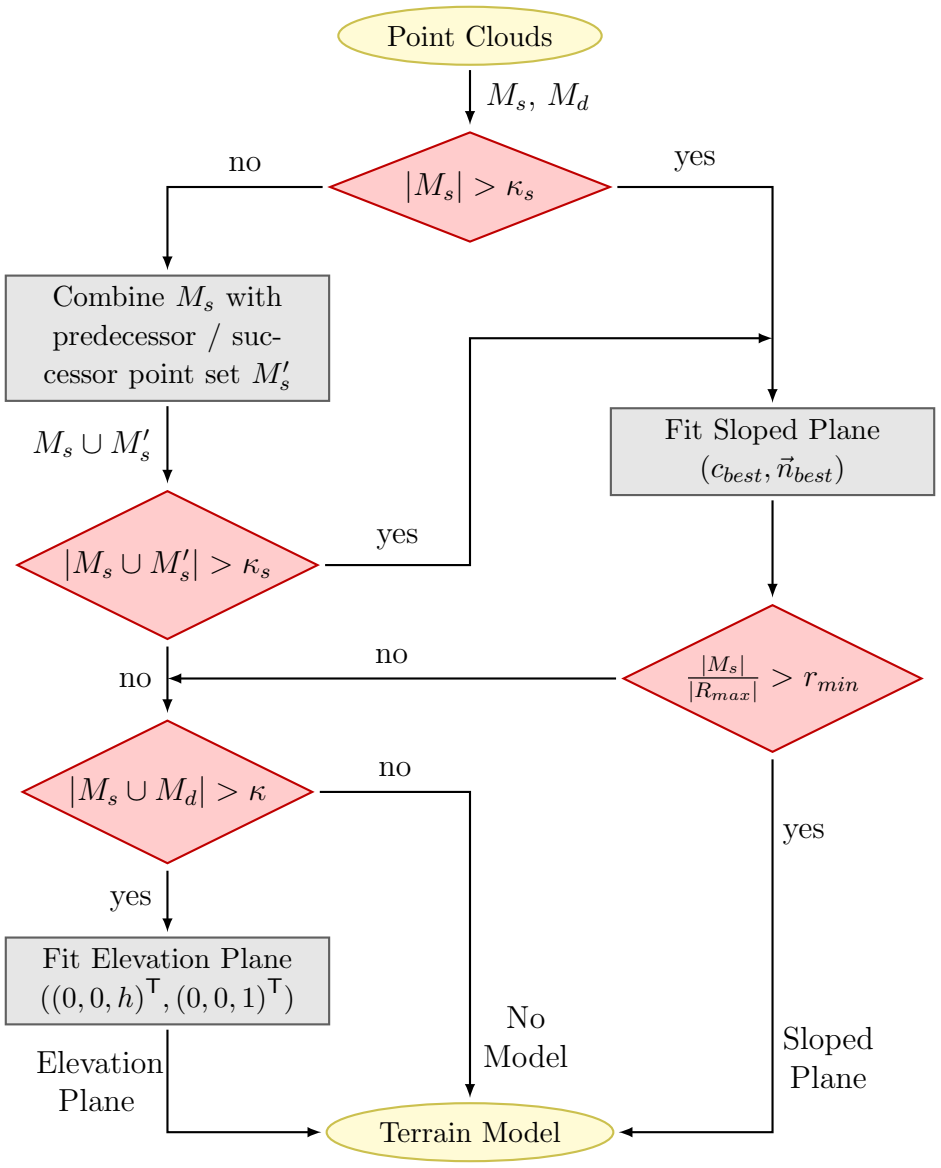


Figure 5.28: Flow of terrain modeling algorithm

of the point cloud within the seclet – a large number of accurate points along a mostly planar surface results in a sloped plane, fewer points (that may be arbitrarily distributed) yield an elevation plane, and very few points result in no model for this seclet.

5.2.4 Traversability Analysis

The result of the terrain modeling step is a seclet map approximating the geometrical shape of the terrain with seclets containing either sloped planes, elevation planes or no plane information at all. Both sloped and height planes are characterized in the same way by a normal vector $\vec{n}$ and a center of gravity (cog) c. For elevation planes, the center of gravity is set to the estimated height: $c = (0, 0, h)^\mathsf{T}$, and the plane normal is the negated gravity vector: $\vec{n} = (0, 0, 1)^\mathsf{T}$, indicating a perfectly flat surface.

To decide whether the robot can actually traverse a seclet given these planar approximations, some assumptions need to be introduced. First, all derived planes are considered

load-bearing. Second, the vehicle's mechanical capabilities are modeled by the maximum slope α_{max} and the maximum step h_{max} it can successfully traverse. Third, as done before in section 4.1.3.3, it is assumed that the robot's possible driving options can be represented as straight lines from the center of the map towards the center of a seclet lying in the desired direction. Thus, every map sector S_m contains a single path to consider, and the seclet s_m^i has s_m^{i-1} as direct predecessor and s_m^{i+1} as direct successor seclet along this path.

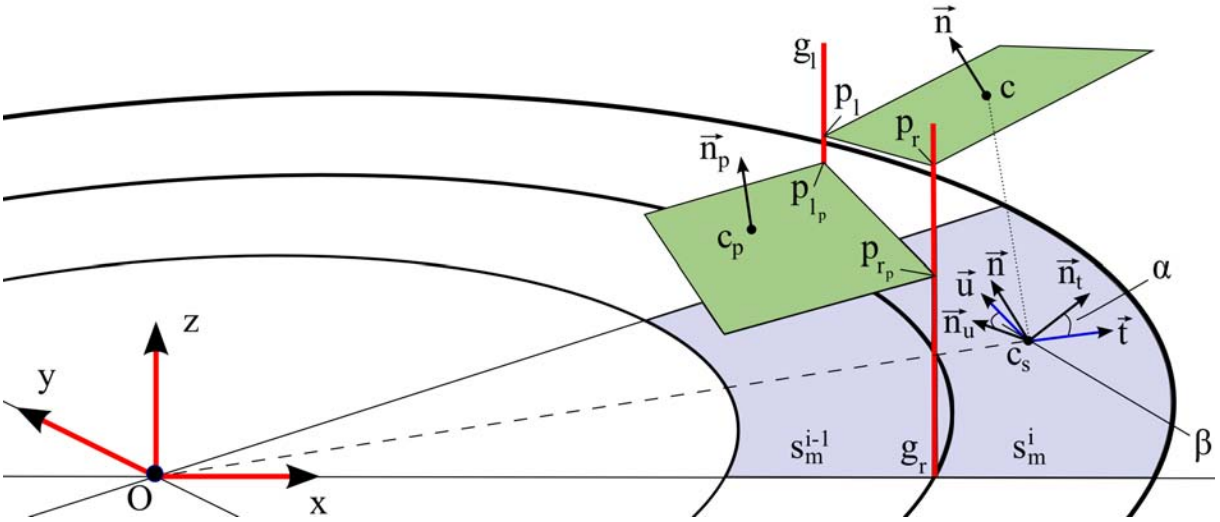


Figure 5.29: Parameters for traversability analysis

If both s_m^i and s_m^{i-1} contain valid planes described by $(\vec{n}, c)$ and $(\vec{n_p}, c_p)$, it is possible to compute **transitional steps** between the two seclets (see figure 5.29). This is accomplished by inspecting the z distance of the planes at the *rim* of the common seclet border. At both of the common seclet corners, a vertical straight line g_l and g_r is intersected with the two planes. The distances d_1, d_2 between the resulting intersection point pairs $\left(p_{l_p}, p_l\right)$ and $\left(p_{r_p}, p_r\right)$ provide two step heights.

Furthermore, the **relative orientation** ϕ of the planes with respect to each other can be determined easily by calculating the inverse cosine of the dot product of the two plane normals: $\phi = \cos^{-1}\left(\vec{n} \cdot \vec{n_p}\right)$.

The **absolute orientation** of the plane $(\vec{n}, c)$ with respect to the gravity vector $(0, 0, -1)^\mathsf{T}$ and the assumed driving direction can be computed as follows: First, two helper vectors $\vec{t}$ and $\vec{u}$ (with $\vec{t} \perp \vec{u}$) are constructed as shown in figure 5.29. $\vec{t}$ is the normalized vector running in the path direction of the current sector, e.g. heading from the map origin O to the center c_s of s_m^i in the x/y plane:

$$\vec{t} = |\overline{Oc_s}| = \begin{pmatrix} c_x \\ c_y \\ 0 \end{pmatrix} \tag{5.28}$$

$\vec{u}$ is orthogonal to both $\vec{t}$ and $(0, 0, 1)^\mathsf{T}$ and can thus be determined by

$$\vec{u} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \times \vec{t} = \begin{pmatrix} -c_y \\ c_x \\ 0 \end{pmatrix} \tag{5.29}$$

Now, the plane normal $\vec{n}$ is decomposed into two orthogonal vectors $\vec{n_t}$ and $\vec{n_u}$ lying in the planes spanned by $\vec{n}$ and the two helper vectors:

$$\vec{n_t} = \vec{u} \times \vec{n} \tag{5.30}$$

$$\vec{n_u} = \vec{n} \times \vec{t} \tag{5.31}$$

Finally, the absolute roll and pitch angles α and β of the sloped plane with respect to both the gravity vector and the assumed driving direction can be computed by solving for α and β in equation

$$\begin{pmatrix} \vec{n_t} & \vec{n_u} & \vec{n} \end{pmatrix} = R_{rpy}(\alpha, \beta, \gamma) \cdot \begin{pmatrix} \vec{t} & \vec{u} & 0 \\ & & 0 \\ & & 1 \end{pmatrix}, \tag{5.32}$$

with R_{rpy} defined as the concatenation of three rotation matrices that rotate the coordinate system with angles α, β and γ around the three primary coordinates axis according to the roll, pitch, yaw conventions. Given R_{rpy}, the three source angles can be determined analytically, as shown in [Craig 89], pp.47.

By construction, the yaw-angle γ is 0 in this case. α is the absolute slope of the approximated plane in driving direction, β is equal to the absolute slope orthogonal to it.

Necessary Conditions for Traversable Seclets

Based on the derived parameters, the following four conditions can be defined to decide if a given seclet s_m^i is traversable:

$$|\alpha| \leq \alpha_{max} \tag{5.33}$$

$$|\beta| \leq \alpha_{max} \tag{5.34}$$

$$|\vec{n_p} \cdot \vec{n}| \leq \cos(\alpha_{max}) \tag{5.35}$$

$$\max(|d_1|, |d_2|) \leq h_{max} \tag{5.36}$$

Conditions 5.33 and 5.34 ensure that the maximum absolute inclination in direction of the path (α) and orthogonal to it (β) remains below the maximal slope climbable by the robot. In case there is no usable plane available in the predecessor seclet s_m^{i-1}, only these conditions are checked. Otherwise, condition 5.35 is used to limit the allowed orientation *difference* (which may either cause high-centering of the vehicle or contact of the bumpers with the ground) between the two seclets. Condition 5.36 is used to detect steps and torsion between two adjacent seclets. As long as the determined step heights are less or equal h_{max}, the seclet junction is considered traversable.

Risk and Effort Estimation

In order to transform the traversability information into risk and effort cost modificators that can be inserted into a local traversability map, a simple heuristic has been developed based on the estimated plane parameters. It is straightforward to replace this heuristic with a more sophisticated model that represents the costs of different slopes and steps more accurately, but this would require a lot of experiments in a well controlled test environment containing defined obstacles and terrain slopes. Formulated somewhat bluntly, the expected precision gain has not been judged as worth the effort required for this.

Instead, the cost heuristics has been formulated as follows:

1. If the seclet is found untraversable according to conditions 5.33 – 5.36, both risk and effort cost modificators are set to the maximum values: $(r, w) = (1, 1)$.
2. Otherwise, the seclet's risk value is determined by the *absolute transitional step height* between the seclet and its predecessor, set into relation with the maximum possible step h_{max}. Thus, r is determined by

$$r = \frac{\max(|d_1|, |d_2|)}{h_{max}} \tag{5.37}$$

   Since the existence of a step can only be detected reliably when slope information is available, the risk is only calculated according to equation 5.37 if both planes are sloped planes. If one is an elevation plane, it is optimistically assumed that its slope favors the transition between the two seclets. Therefore, no risk penalty is enforced in this case.
3. A traversable seclet's effort value is determined by the *absolute plane orientation* in the path direction of the current seclet, with respect to the maximum possible slope α_{max}. Thus, w is determined by

$$w = \frac{\max(|\alpha|, 0)}{\alpha_{max}} \tag{5.38}$$

As can be seen, negative slopes do not provide an effort bonus with this heuristics. This is done because the quantitatively correct estimation of *negative* slopes is barely possible based on stereo images. Negative obstacles are almost invisible when seen from a low height (such as the robot camera head's mounting position). Therefore, it has been decided to treat negative slopes neutrally as planar surfaces to avoid reasoning on potentially unreliable data.

The confidence values (θ_r, θ_w) for each seclet that holds either a sloped plane or an elevation plane are set to $(0.8, 0.8)$. This value represents both the fact that the long range terrain traversability estimation is less reliable than the local obstacle model (whose cost information is assigned reliability values of $(1, 1)$) and the typical error characteristics found in a variety of experiments in different terrains.

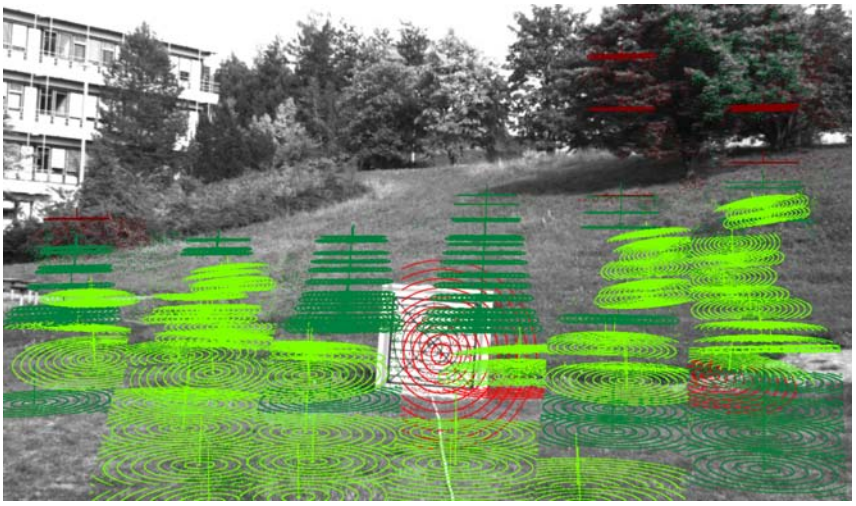
Integration

As already hinted at before, the derived cost modifiers are included into the cost prediction mechanism presented in chapter 4 by filling them into a *separate* local traversability map similar to the one defined in section 4.1.3.1. This map is subsequently attached to the topological node for which the shape-based terrain model has been constructed. It complements the already existing local map containing the cost information from the local obstacle memory. If the cost estimation mechanism needs to retrieve the stored cost modifiers for a specific seclet, the reliability scores of the two corresponding seclets in both local maps are compared. Then, the pair of cost modifiers with the higher reliability score is returned for cost prediction.

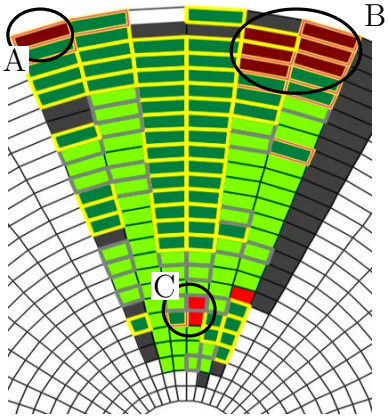
In effect, this lets the robot use the local obstacle memory for cost estimation in the local vicinity of the topological node and allows it to switch seamlessly to the shape-based cost estimation technique at longer ranges.

Example

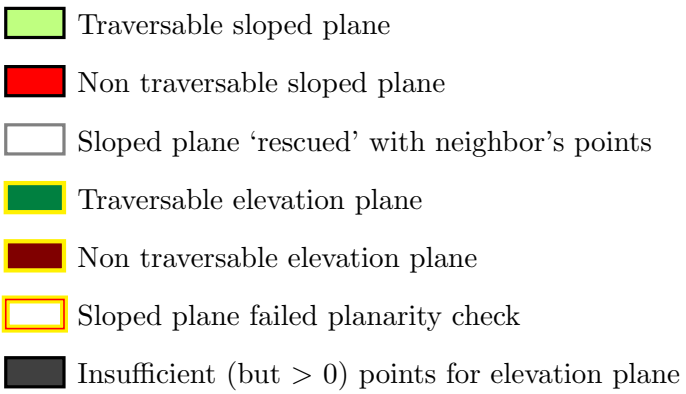
Figure 5.30 shows the results of performing the described traversability analysis on this chapter's running example image from figure 5.18.



(a) Scenario with backprojected sloped and elevation planes



(b) Seclet traversability



(c) Seclet color code key

Figure 5.30: Example scene after traversability analysis

In figure 5.30b, the estimated seclet traversability is shown. For this and the following experiments, the parameter set listed in table 5.3 was used. The three circled groups of seclets that have been found untraversable correspond to the bushes in the center left of the scene (A) at a distance of ≈ 30 m, the large tree on the right (B) with comparable distance and the calibration object in the image foreground (C), which was placed at a distance of 10 meters from the robot. Both the distances and the bad traversability of these objects have been determined correctly by the proposed method. However, one seclet has been found untraversable by mistake, as the sand pit on the far right is actually shallow enough to be traversed. However, the bad view angle caused the sloped plane to be placed in such a way that the transitional steps are exaggerated and estimated to lie above h_{max}.

This scenario is a good example of the benefit provided by the combination of building planar approximations both with and without slopes. While the terrain to the left and the

Param.	Value	Parameter Meaning
κ_s	45	number of points required in M_s for sloped plane approximation
κ	25	number of points required in M for elevation plane approximation
d_{max}	80 mm	maximum allowed distance of inlier point from sloped plane
r_{min}	0.6	required percentage of inlier points in M_s for sloped planes
h_d	40 cm	height of the difference operator for elevation plane modeling
α_{max}	40°	maximum traversable slope
h_{max}	35 cm	maximum traversable step

Table 5.3: Parameters used in the experiments

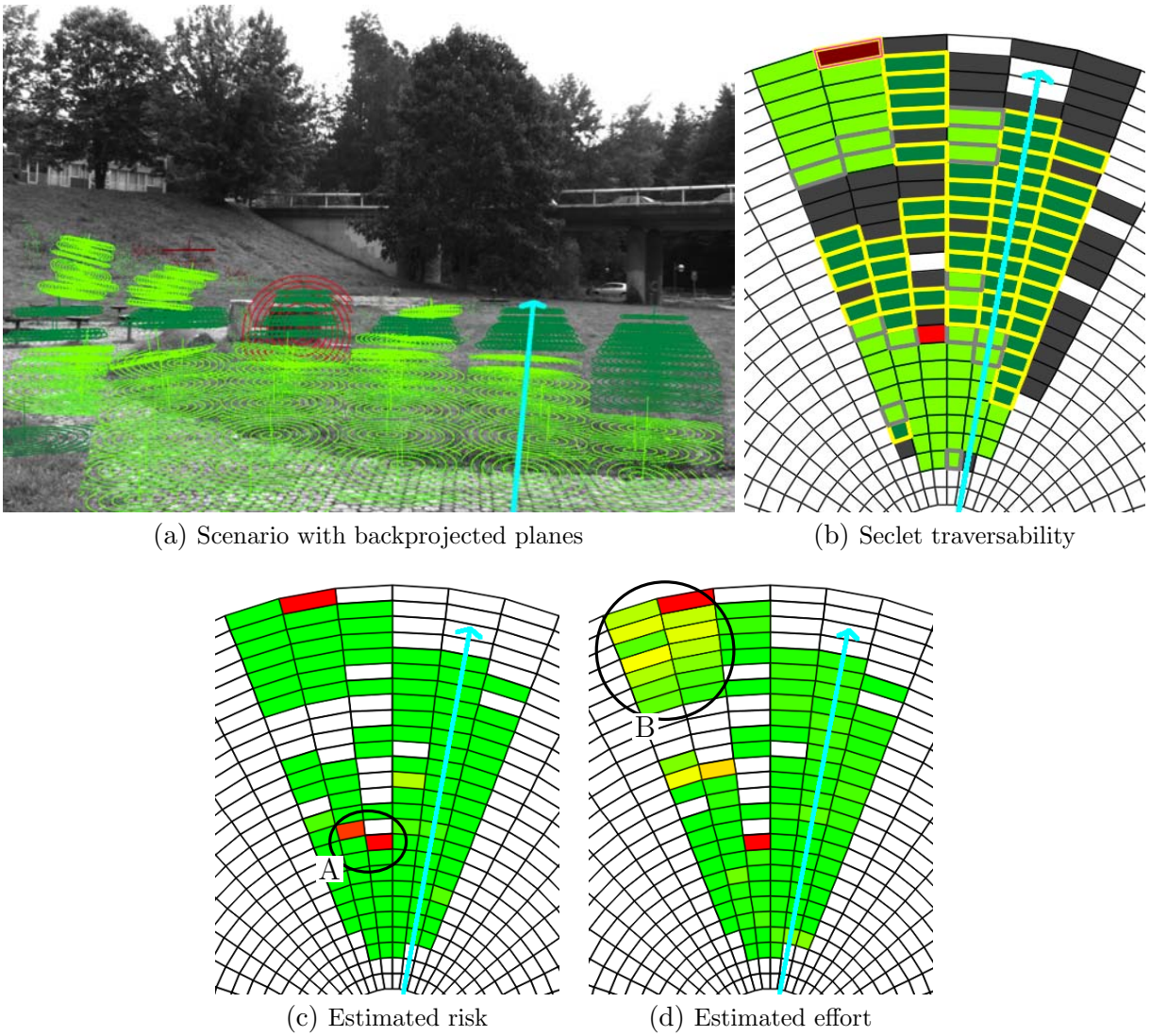
right of the calibration plate contains enough points to be modeled accurately with sloped planes, the object casts a 'point shadow' onto the seclets directly behind it (compare with figure 5.24), which precludes accurate estimation of the terrain slope. Consequently, the approximation falls back to pure elevation based modeling in this area (marked with dim green and red colored seclets), which at least suffices to model the overall rise of the ground accurately. If this fallback mechanism had not been developed, no information of the terrain behind the calibration plate would have been available at all and this well traversable area would have been missed. The same argumentation holds for distances above 24 meters, which also exhibit such a low point count that they can only be reliably modeled without considering slope information.

5.3 Experiments and Results

In the following, experimental results are presented for several different outdoor scenarios. For each result, four images are shown. First, the left input image is presented, annotated with the found planar approximations. Second, the corresponding traversability map is shown, using the color key given in figure 5.30c. Finally, the computed risk and effort cost modifiers are depicted. These modifiers are the sole information that is retained after the terrain analysis has concluded and represents the final outcome of the entire method described in this chapter. The color key used for these maps is identical to the colors used for the already existing local traversability map: green seclets indicate a cost modifier of 0, yellow stands for 0.5 and red seclets have the maximal cost of 1.

Figure 5.31 exhibits two peculiarities, which have been marked with circles A and B. Mark A is placed around the seclets occupied by a boulder and a service box, which have been correctly judged as untraversable obstacles. In contrast, the seat ring to the left of the boulder is marked traversable although the seats constitute a step obstacle which cannot be passed in reality. A cause for this is the bad viewability of the seats, which has precluded a more accurate modeling with sloped planes that would have been vertically oriented and thus found untraversable in accordance with the true circumstances. Mark B highlights the upward slope in the background of the scene. Agreeing with the estimated slope, the predicted effort increases significantly.

To give an overall impression of the effect on path planning caused by the analysis results, a cyan arrow indicates the path alternative with the lowest cost given the cost modifiers shown in figures 5.31c and 5.31d.

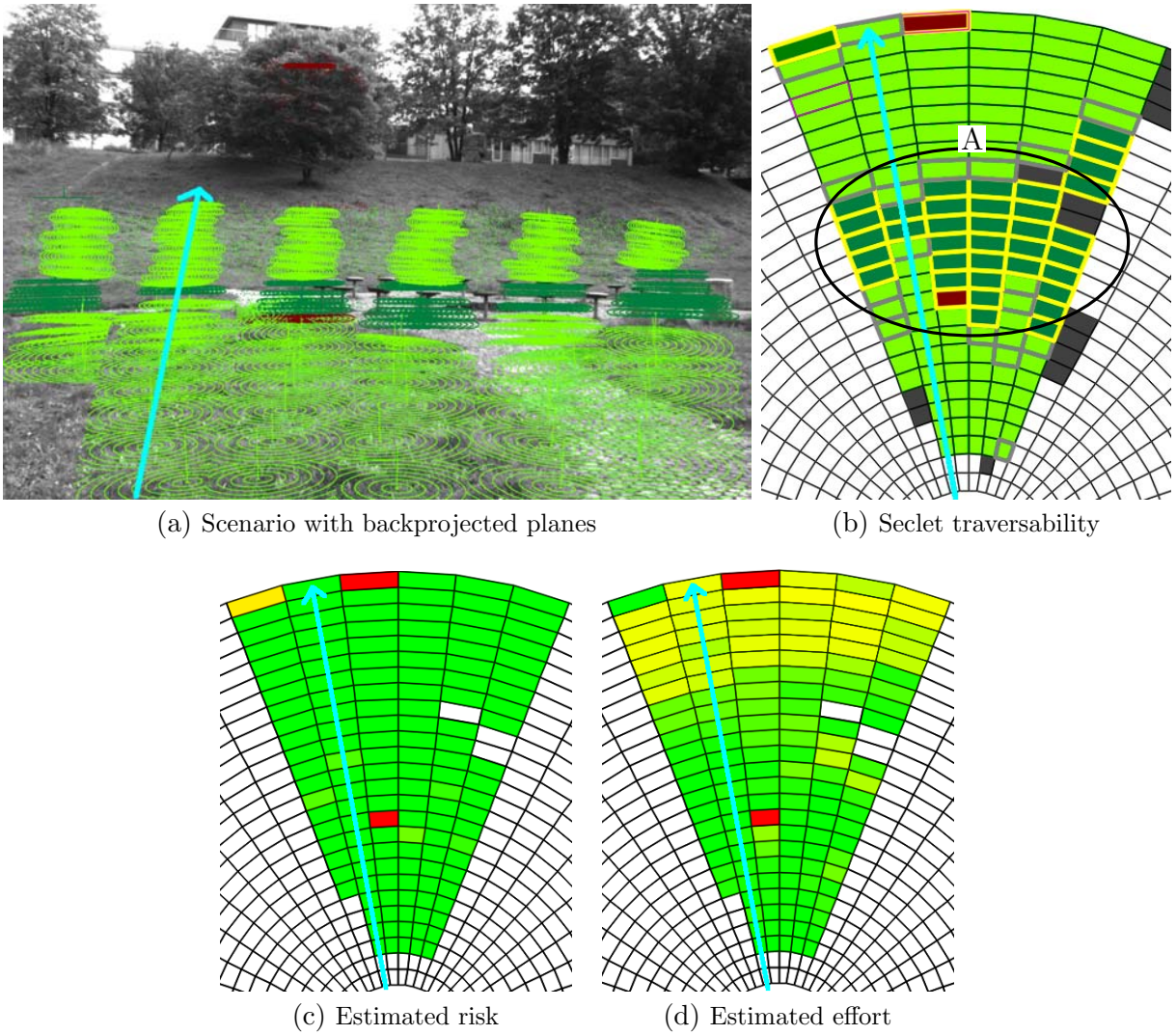


(a) Scenario with backprojected planes (b) Seclet traversability

(c) Estimated risk (d) Estimated effort

Figure 5.31: Example scene after traversability analysis

The second image shows the ring of seats more fully and also contains a larger part of the rising slope in the background. As in the previous image, the sloped plane approximation captures the overall terrain layout well and the estimated effort cost factor models the rising terrain surface appropriately. However, the area around the ring of seats (mark A) is only represented using elevation planes, due to the same reasons as discussed for the last figure. Because the steps between elevation planes are judged optimistically (as long as their size remains below the traversability limit h_{max}), they do not impose a risk penalty and the ring of seats is not recorded as a cost relevant obstacle. Nevertheless, the cheapest path does not cross the seats, due to the effect of the step heights on the estimated effort.



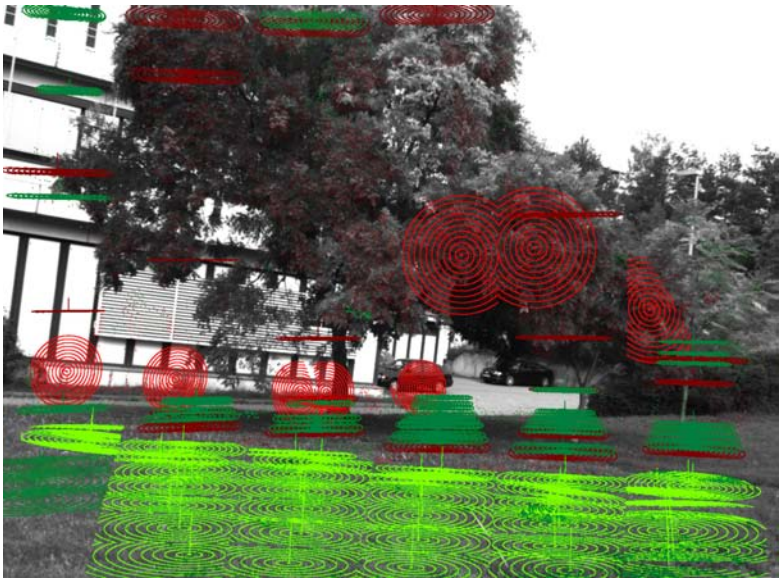
(a) Scenario with backprojected planes (b) Seclet traversability

(c) Estimated risk (d) Estimated effort

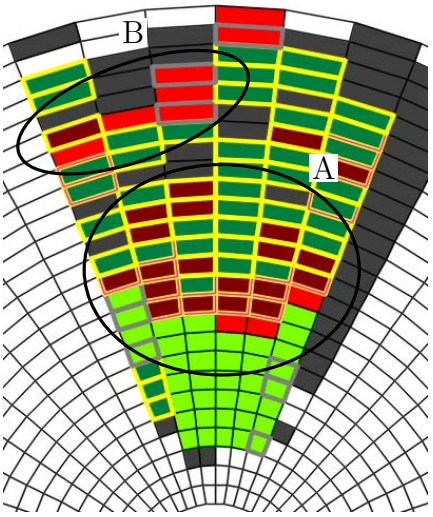
Figure 5.32: Example scene after traversability analysis

Figure 5.33 shows a more obstructed scenario. The shape-based traversability analysis marks both the large tree (mark A) and the building wall (mark B) as untraversable obstacles. Map seclets that are not covered by one of these two obstructions are correctly identified as flat and traversable. This scenario highlights a shortcoming of the proposed terrain modeling method in connection with overhanging obstacles. Although the tree top overarches the path and is correctly identified as not passable, the clearing below the tree is (at least partially) sufficiently high to let the robot pass underneath. In order to allow the correct modeling of such overhanging obstacles, a further processing step must be implemented to segment the seclet's point cloud at the robot's height.

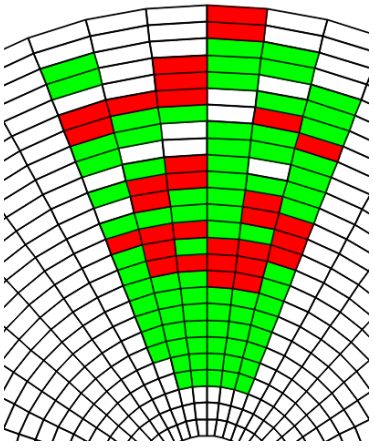
Figures 5.34, 5.35 and 5.36 depict further results that have been obtained at several locations around the campus of the University of Kaiserslautern.



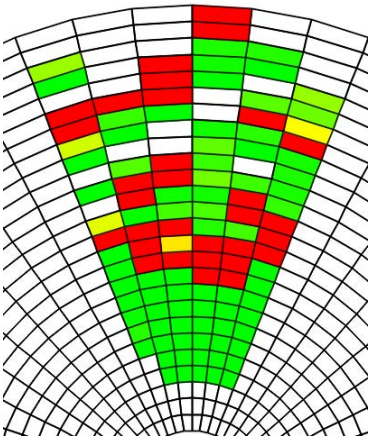
(a) Scenario with backprojected planes



(b) Seclet traversability



(c) Estimated risk



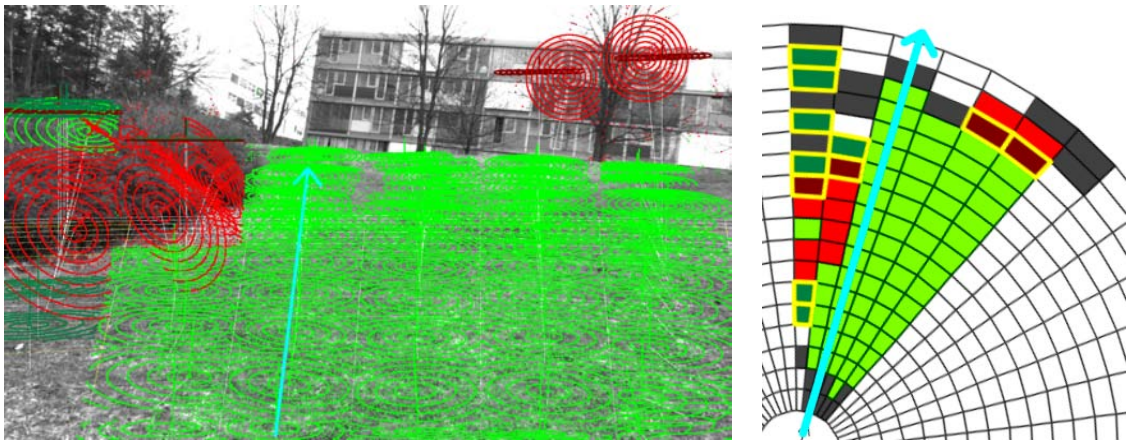
(d) Estimated effort

Figure 5.33: Example scene after traversability analysis

5.4 Conclusion

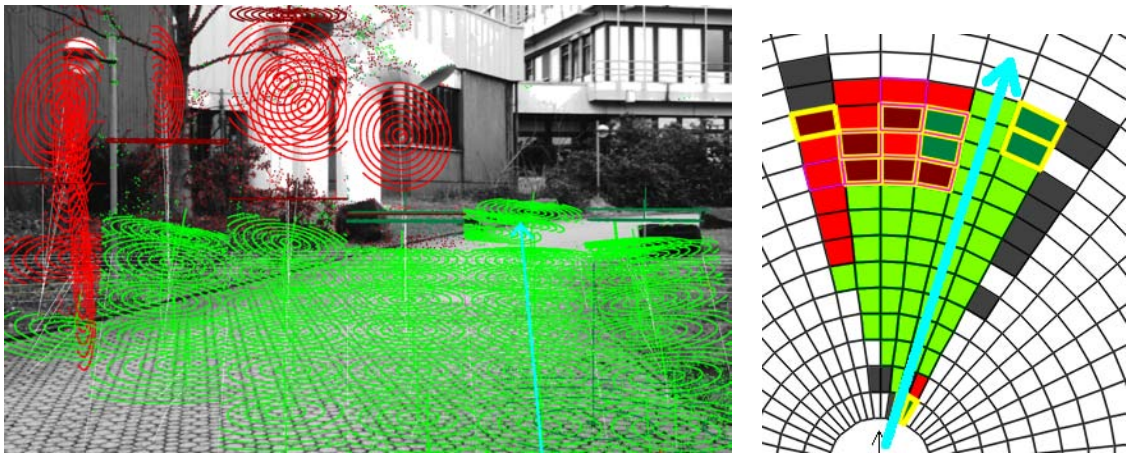
The shape-based traversability estimation method presented in this chapter has been developed to augment the cost prediction capabilities of the topological navigation system at longer ranges. For this purpose, a terrain model has been proposed which can approximate the surrounding terrain either using planes containing slope information or purely height-based elevation planes. The obtained experimental results show that the novel combination of these two established models indeed allows to extend both range and accuracy of the obtained terrain model compared to just using a single variant. Overall, the method has been proven to be applicable to a variety of different outdoor environments and generate sensible terrain traversability cost estimates within a range of up to 30 meters. The extracted information can be used to optimize path planning and avoid large scale obstacles or significant terrain slopes.

Due to the design and parameter decisions made during the development of the algorithm, the traversability estimation tends to be rather ‘optimistic’. One example of optimistic



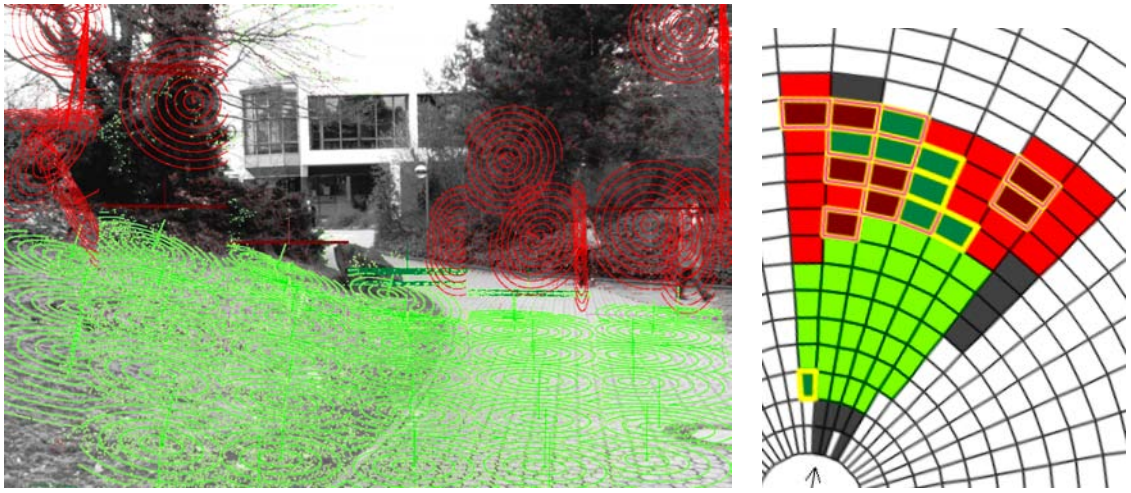
(a) Scenario (b) Traversability map

Figure 5.34: A hill and a hedge



(a) Scenario (b) Traversability map

Figure 5.35: A curved road



(a) Scenario (b) Traversability map

Figure 5.36: A second curved road

design is the fact that transitional steps are not counted as a potential problem if elevation planes are involved. While this assumption causes small obstacles such as the ring of seats to be erroneously judged as traversable due to its relatively low height and small image footprint, such optimistic decisions can be corrected later once the robot approaches the relevant area using the pilot's local obstacle avoidance strategies. A more pessimistical tuning would preclude any closer examination of uncertain traversability in the first place and thus reduce the robot's number of potentially good exploration choices.

Although the presented method does already provide significant advantages in hilly terrain containing natural obstacles such as trees or hedges, the application in more densely vegetated environments requires the extension of the proposed method in order to treat overhanging objects more appropriately. This extension can be implemented by segmenting the obtained point cloud of a given seclet according to the height of the estimated surface level in the neighboring seclets plus the minimum safety distance required for the robot. However, as a sound development would require an additional intense testing phase, this extension has been postponed in the scope of this thesis and is deemed future work.

6. Appearance-Based Terrain Traversability Estimation

In the last chapter, a method was presented to estimate terrain traversability using geometric aspects like surface slope, planar orientation and height variation. However, this approach does only consider the three-dimensional outline of the visible terrain and treats non-rigid vegetation such as tall grass or small bushes exactly like tall stones or rigid fences with a similar shape. As a consequence, the robot navigation system discards actually traversable paths through such flexible obstructions.

To improve the performance of the robot especially in vegetated terrain, both the piloting layer and the navigator need to be able to handle soft obstacles, which might appear solid to shape-based analysis methods but are actually traversable. For the pilot, a force feedback 'tactile creeping' strategy has been proposed (see [Schäfer 08]). With this strategy, a potentially soft obstacle is approached slowly and pushed with the front bumper. If the bumper is not depressed slightly during this process, the obstruction is treated as a flexible part of vegetation which may be traversed. For the navigator, such a local strategy is infeasible. Instead, a terrain analysis method with the capability to distinguish between flexible vegetation and really impassable obstacles *from a large distance* is required.

This chapter presents a traversability estimation method which has been developed to address this need. It estimates the traversability of the environment without relying on its potentially misleading geometrical shape. Instead, the approach is based on **visual terrain appearance** as data source. In order to cope with the high variability of outdoor terrain, multiple complementary visual features are extracted and used for image segmentation and terrain classification. As with the shape-based approach, the resulting traversability information is stored into a local traversability map and thus becomes available to the cost prediction algorithms in a transparent and consistent fashion. Additionally, because static rules for terrain classification based on visual appearance do not adapt well to previously untrained situations, a self-supervised online training scheme is proposed in order to allow the robot to update its classification model and enable it to work even in novel environments without explicit classifier training. The approach was formulated in cooperation with B. Seidler [Seidler 08], F. Faust [Faust 08] and G. Zolynski

[Zolynski 07]. The obtained results have been summarized and published in [Braun 08c] and [Zolynski 08].

The remaining chapter is structured as follows: Building upon a short survey of related work in section 6.1, section 6.2.1 provides details on the three complementary visual features texture, contrast and color. They have been selected as suitable means to extract characteristical appearance information from a source image. This section also describes the extraction process itself. The developed traversability estimation method then proceeds to segment the image data into homogeneous regions based on the extracted feature signatures, which is described in section 6.2.2. Afterwards, the assignment of traversability scores to the segmented image regions using a k-Nearest-Neighbor classifier is described (section 6.2.3) and the online learning scheme is presented in section 6.4. Experimental data, a final discussion of the obtained results and an outlook to future perspectives conclude this chapter.

6.1 Related Work

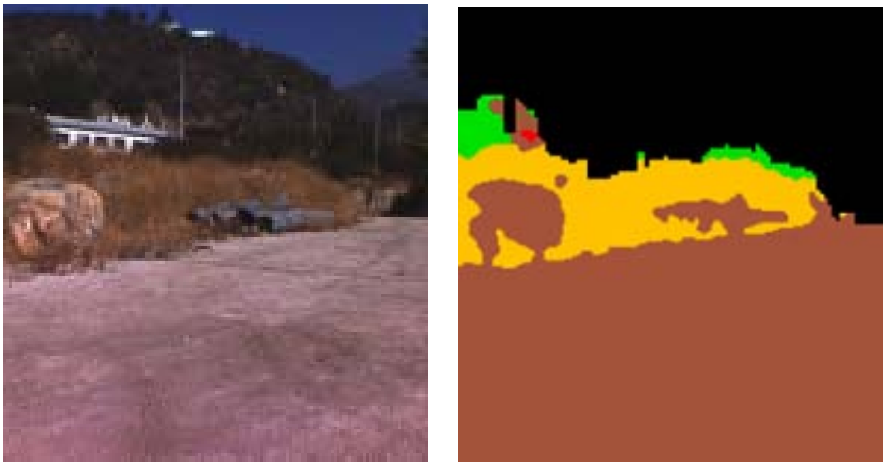
Two main classes of appearance-based terrain classification methods can be distinguished. One class relies on statically trained databases that link terrain appearance and traversability scores. Methods of this class can be employed without further delay in the environment that they have been developed and trained for, but they face problems in situations which differ from the original training phase due to e.g. seasonal changes, different lighting or weather changes. The other class of traversability analysis methods includes a continuous, self-supervised learning component able to adapt the classification database constantly during robot operation. Since this allows a more flexible and autonomous robot operation, most of the recently proposed appearance-based terrain traversability estimation methods fall into this second category.

In the following, a short overview of the most prominent approaches from both of the two major classes is presented.

6.1.1 Statically Trained Approaches

P. Belutta et. al. [Bellutta 00] [Manduchi 05] combine LADAR, radar, and color as well as infrared cameras to navigate autonomously in vegetated off-road terrain with the unmanned ground vehicle DEMO III. Geometric information from stereo vision is used to recover the 3D scene structure in order to detect obstacles. Both elevation and obstacle maps are created through triangulation. The terrain surface itself is analyzed using color information. A fast bayesian classification algorithm is trained on a priori labeled images and used to characterize the detected obstacles into different classes based on the observed color distribution (figure 6.1). Then, traversability characteristics are inferred from each point's assigned class. Although the experiments show the principal usability of the presented approach, they also highlight changing illumination conditions as a central problem of the solely color based terrain classification approach, especially when shadows are present in the observed scene.

The initial approach of P. Belutta et. al. was improved later by A. Talukder et. al. [Talukder 02]. Here, the terrain classification performance is optimized by adding texture analysis to the color classification scheme. A multi-scale, multi-orientation Gabor filter



(a) Input image (b) Classified image

Figure 6.1: Color based terrain classification in [Bellutta 00]
The color code used for (b): brown: soil/rock; yellow: dry vegetation; green: green vegetation; red: outlier

bank is utilized for the extraction of textural features, which are subsequently classified using a gaussian mixture model and maximum likelihood classification. For the color-based material classification, the same bayesian color classifier is used as in the original approach. Both classifiers are trained with sample images that are labeled with the three terrain surface classes shown in figure 6.1b: 'green vegetation', 'dry vegetation' and 'soil/rock'. Experimental results show that the fusion of both color and texture classification leads to a more robust obstacle recognition and traversability estimation than each single method alone. The approach was shown to be useful for navigation and path planning, especially in off-road terrain. However, the presented experiments cover only arid and sparsely vegetated environments.

In a different research project, terrain traversability assessment was done with three scales of resolution [Seraji 03]. The three scales are local, regional, and global traversability indexes. For local traversability a set of linguistic rules to locate obstacles and surface-softness with on-board sensors is used, while regional traversability is estimated rule-based from video images heeding roughness and slope; for global traversability, a terrain topographic map is created. Each traversability index is represented by fuzzy sets, corresponding to the safety for traversal of this region.

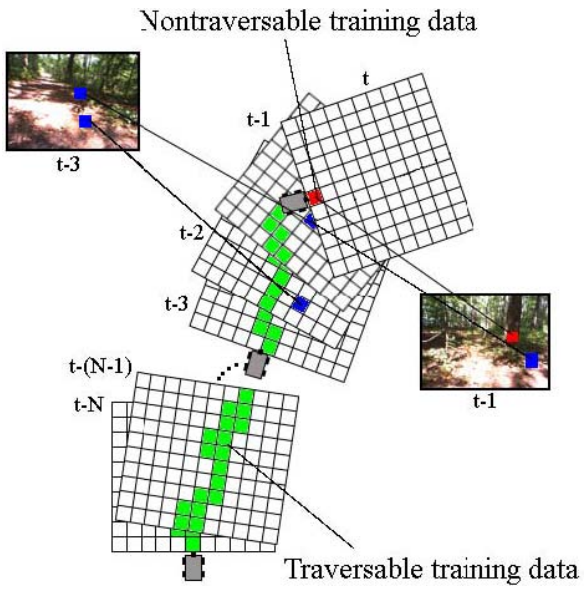
6.1.2 Learning Techniques

There are two important reasons to employ machine learning techniques for traversability estimation tasks. For one, changing environmental conditions can lead to poor long term performance of terrain classification algorithms that are based solely on static rules. In order to **adapt to new situations**, some kind of learning strategy is required instead.

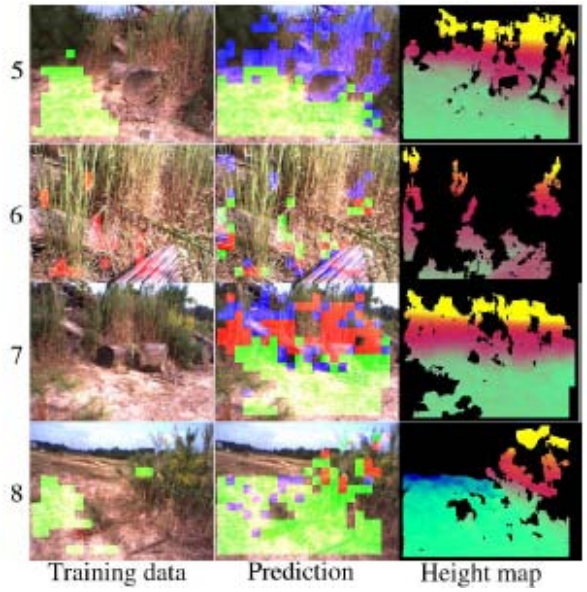
Another application of learning with regards to traversability estimation is sometimes referred to as **Near-to-Far Learning** [Hadsell 07b]. Here, the goal is to train one component of a robot's sensor suite which has a wide range but (initially) little traversability information (like a stereo camera) with labels provided by another component that has

only a limited scope but more reliable traversability information (e.g. a bumper system). In order to achieve this, the traversability data of the short range sensor must be corresponded with feature information obtained by the long range sensor. For this, a local map or memory is often used as temporary storage.

D. Kim et. al. [Kim 06a] motivate their near-to-far learning approach by noting that traversability is a complex function which depends on both vehicle properties and environmental characteristics. The exact relation is difficult to characterize a priori by simple static rules. Instead, Kim et al. propose to learn the accurate prediction of terrain traversability properties from stereo images through a self-supervised, online learning method. They employ a set of overlapping, egocentered local maps constructed at different times and positions. In order to avoid registration errors due to GPS jumps or wheel slippage, only the most recent local maps are considered, older maps are discarded.



(a)



(b) Left: Original image, middle: Color-encoded traversability based on learning, right: Color-encoded traversability based on height maps

Figure 6.2: Overlapping local maps and traversability estimation (Source: [Kim 06b])

Every time a local map is generated, feature vectors encoding the textural appearance of rectangular patches taken from the current stereo image are calculated using Laws Masks. The vectors are back-projected onto the grid cells of the map and used as a cell label. When the robot is traversing a previously labeled cell, the success or failure (detected by bumper hits) of this traversal and the feature vector label is used as training data for the classification module which assists the path planning unit.

During the evaluation by Kim et. al. , this approach turned out to be superior over height based traversability estimation, especially when vegetation like tall grass is involved.

A probabilistic online learning framework for autonomous off-road robot navigation is presented in [Erkan 07]. This approach predicts traversability in unknown environments based on vision sensors only. Self-supervised near-to-far learning is applied to convey traversability information from short range stereo-vision sensors to long range visual data.

For feature extraction, a multi-layer convolution neural network (CNN) is trained offline. Based on the derived features and the traversability information from short range analysis, an online learning module learns terrain traversability for visual input spanning longer ranges. This system is capable to adapt to any new, unseen terrain without human intervention. Tests yield an overall classification accuracy of 85 %. However, the approach assumes a single ground plane, which can restrict applicability of this method in uneven or hilly terrain.

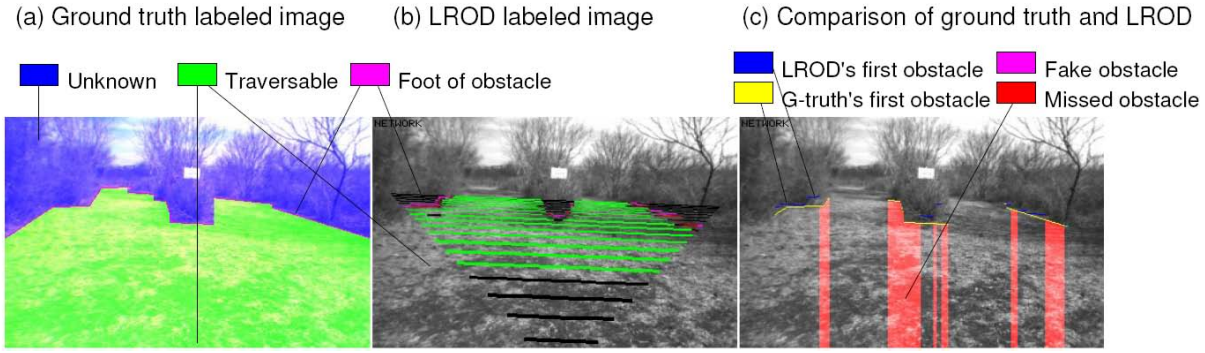


Figure 6.3: Terrain traversability learning in [Erkan 07]

A self-supervised learning algorithm for road detection in desert-like areas is shown in [Dahlkamp 06]. In contrast to the example before, several LIDAR scanners and a monocular camera are utilized and the LIDAR data is the leading source of information. At first, rigidly mounted laser scanners at the front provide data used to create a 2D binary drivability map which is based on height differences. In this map, the largest area marked as drivable is located and projected into the visual plane of the color camera.

Figure 6.4: Stanley, the vehicle used by [Dahlkamp 06]

This can be easily done since the position of the scanner and the camera with respect to the vehicle is known. After removing unwanted information such as shadows and the sky, the backprojected area in the visual plane is used to train a mixture of gaussian color model of traversable surfaces. Afterwards, the remaining image pixels are classified based on the learned model and drivable areas are detected. The major advantage of this approach is that it can adapt to different appearances of the road, as long as the LIDAR system is capable of detecting obstacles in the new terrain. A classification example for this approach can be found in figure 6.5.

[Howard 06] addresses the problem of discriminating traversable from non-traversable vegetation on rough ground in stereo images. E.g. sand dunes can have a suitable geometrical shape for traversal, but they include the risk of downhill slippage or sinkage. Their idea

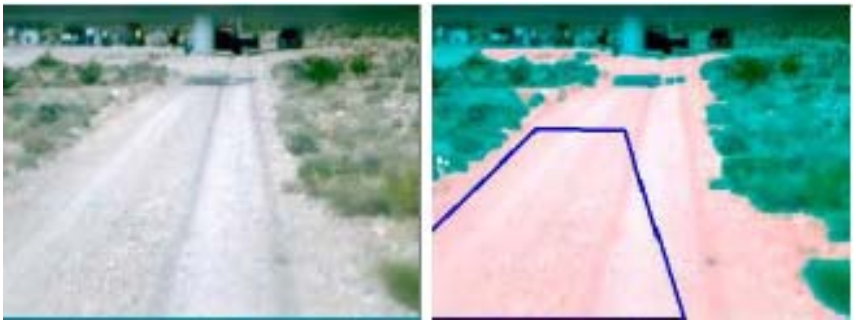


Figure 6.5: Near-to-far learning of a color based road model
Left the original image. The training area (blue lines) and the drivable area (light red) is marked in the right image (Source: [Dahlkamp 06])

is to learn the association of sensor data such as provided by odometry, gyrometers, accelerometers, inertial systems or bumper contacts with the appearance of the scene. They term this method learning from proprioception (LfP). Similar to [Dahlkamp 06], the advantage of this approach is that the traversability decision is based on learning, so no model of the physical behavior of the robot is necessary. But the generation of training data, especially for unpassable situations, has turned out to be resource expensive since intrafficability has to be experienced by the actual vehicle, resulting in emergency bumper stops or other situations an operator has to intervene. Also localization and data registration problems were found to have an impact on learning at one place and using this experience at another one.

Figure 6.6: Learning from proprioception (LfP)
The figure shows a sample image, a projection onto the local map and learned traversability predictions (Source: [Howard 06])

B. Sofman et al. [Sofman 06] use overhead imagery data to improve autonomous robot navigation. Such high quality images are widely available nowadays, due to the use of satellites and remote sensing techniques. Both color (in HSV color space) and textural features (extracted by Gabor filters) are used. The traversal costs over large areas are predicted from overhead data by a self-supervised learning method utilizing a Bayesian probabilistic framework. This leads to improved path planning with significantly shorter paths. However, the drawback of this approach is the required overhead imagery data which is not always available, especially when navigating autonomously in cross-country outdoor environments. Entirely autonomous driving would require to store overhead imagery data of each possible location a priori or a persistent connection to a base station.

In [Sun 06], a mobile robot learns to drive similar to a human operator based on sample data collected during a human-controlled example run. The presented approach learns

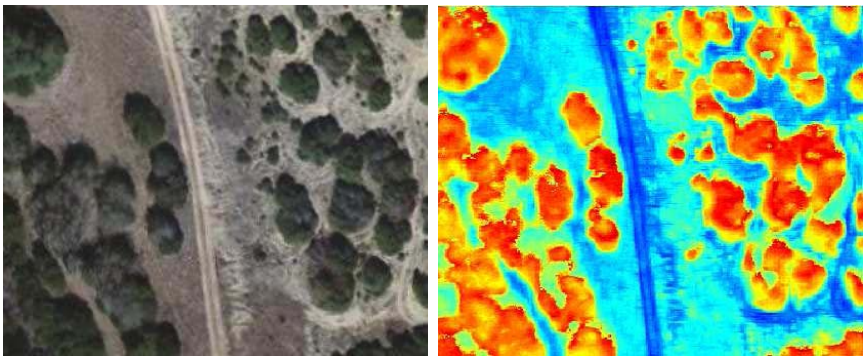


Figure 6.7: Classification of overhead imagery [Sofman 06]
Red colors indicate impassable terrain, blue well traversable areas.

on two levels. On the perception level, color and textural features are extracted and associated with traversability costs: low costs are assigned to those colors which were passed by the human operator, other colors are more costly. Subsequently, a global map with estimates of preferability and standard obstacle information is created from that and can be used for path planning. On a behavioral level, mappings from visual features to control actions are learned and used to tune the weighting of existing behavioral primitives (such as 'follow path', 'turn right' etc.) based on the currently observed features. Real-world tests show that the vehicle copies the human operator's behavior in terms of safety (e. g. distance to obstacles).

However, gathering training data with a human operator is time consuming and can even be dangerous (e. g. in traffic as for *ALVINN*). Furthermore, this approach does not adapt well to changing conditions without additional training.

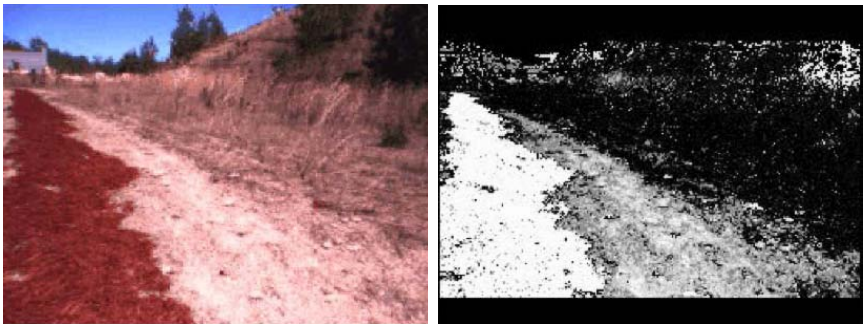


Figure 6.8: Learned image classification [Sun 06]
Brighter colorized regions are preferred.

M. Shneier et al. [Shneier 06] [Shneier 08] use range sensors (stereo vision), color camera, and the vehicle's navigation system to predict terrain traversability for path planning. The classification scheme is learned entirely without supervision, only from geometry and appearance of the scene (known from stereo imagery). The terrain traversability is then learned by building models of terrain regions using features including texture, color, and traversability properties. The range measurements from stereo vision are used to assign traversability measures to all regions. Again it is assumed that regions that look alike have similar traversability properties. Thus, terrain appearance is associated with traversability values. When all regions are classified, the path planner determines an optimal

traversable path.

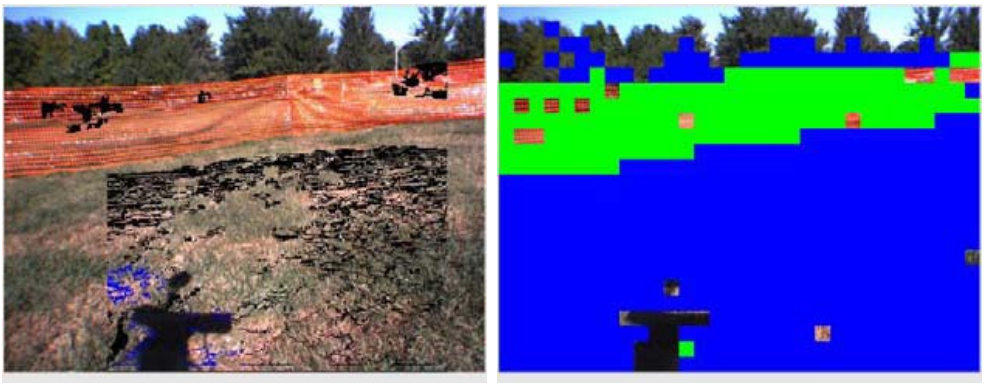


Figure 6.9:
Traversability classification based on texture and color features (Source: [Shneier 08])

The algorithms classification results were compared with ground truth created by a human observer in order to determine performance and error rate. The obstacle detection algorithm agreed with the human's classification 91 % of the time.

6.1.3 Selection of a Suitable Analysis Method

The survey of related work discussed above shows that appearance-based traversability estimation typically relies on *multiple features*, out of which color and textural features are most frequently used. Experimental results indicate that while color is a powerful cue to discriminate between different terrain types, its reliability under changing illumination conditions is critical and warrants the use of additional information. Texture, on the other hand, appears to be more stable when exposed to similar disturbances. However, the extraction of textural features can be a computationally expensive operation, especially when high-quality measures such as multi scale gabor feature banks are used. Thus, texture extraction methods need to be carefully selected in order to provide a viable compromise between run time and feature quality.

Concerning the use of statically trained or online learning methods, most recently proposed methods contain at least *some* learning component. It is apparent that the flexibility of the a priori trained methods is too low, given the high variability of outdoor terrain. Thus, many researchers propose some kind of near-to-far learning technique in order to extrapolate knowledge that is obtained from reliable close-range sensors onto the available long range equipment. Since the terrain appearance must also be evaluated by this sensor, virtually all approaches use a (stereo) color camera system for this.

Motivated by this discussion, the following decisions have been made concerning the development of the appearance-based terrain traversability method in the scope of this thesis:

1. Due to their complementary strengths and good previous results, *both* texture and color features will be used for feature extraction. In order to allow later runtime optimization, a powerful, yet computationally manageable textural feature is to be selected.

2. The terrain classification method shall be trained a priori with a set of hand labeled images. However, this database is to be extended online in order to allow adaption to new operational conditions, using a variant of near-to-far learning. More specifically, the mechanisms used for cost learning of topological edges shall be reused for this purpose: the spatial integration of behavior target ratings yields already interpreted and well accessible locations for badly traversable terrain. Combined with positive feedback from the areas which the robot has traversed, a balanced feedback loop can be implemented.

The next section presents the developed approach and its constituent parts in detail.

6.2 A New Approach for Appearance-Based Traversability Estimation

The appearance-based traversability estimation algorithm that has been developed in this thesis proceeds in the sequence of steps shown in figure 6.10.

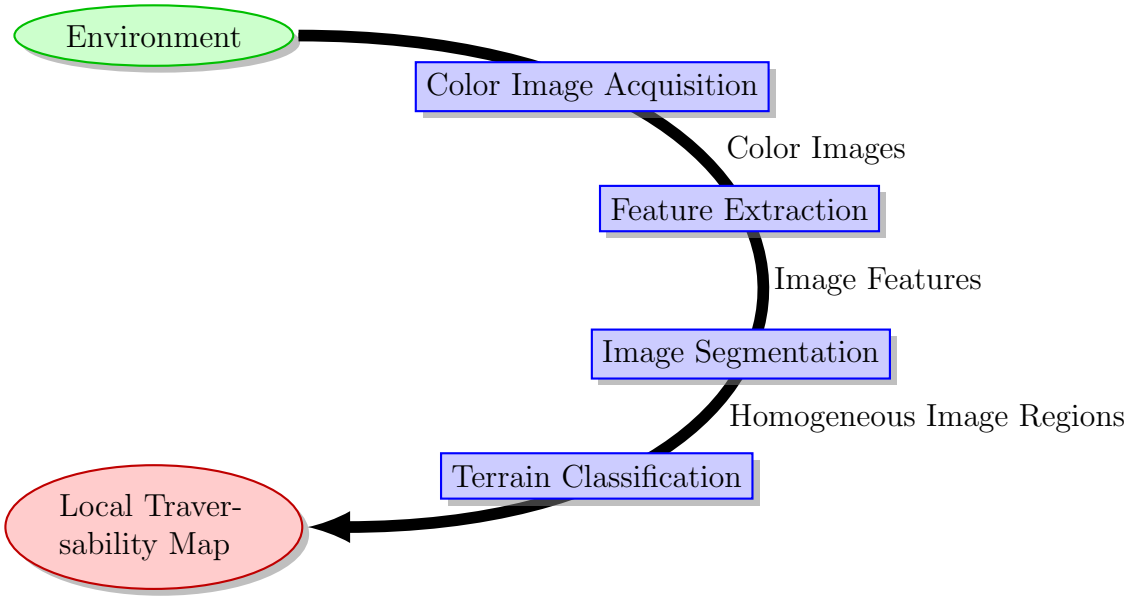


Figure 6.10: Workflow of proposed method

Similar to the shape-based traversability analysis method presented in the last chapter, the first step consists of acquiring color images of the surrounding terrain. For this, the available high-resolution stereo camera system described in section 5.2.1 is used in combination with the implemented custom exposure control. Then, texture and color features are extracted from the obtained source images. Next, the image is segmented into homogeneous regions using an unsupervised clustering method and a measure of feature similarity. This step significantly reduces the number of classifications that need to be performed later without impairing the classification accuracy. Also, the combination of smaller image patches to larger regions improves the statistical stability of the extracted image descriptors. Finally, the determined regions are classified into different traversability classes using an a priori trained database. The online learning system is not shown in figure 6.10 and will be introduced separately in section 6.4.

In the following sections, the feature extraction, image segmentation and terrain classification steps will be discussed.

6.2.1 Feature Extraction

In general, feature extraction describes the process of projecting high-dimensional data (raw image data) onto a low dimensional subspace, where extracted features are usually represented by a **feature vector**. Ideally, the chosen abstraction keeps the characteristical information which is relevant for the task at hand and removes all other, misleading or redundant data. If the method is appropriately chosen, the selected features provide a simpler and computationally cheaper representation than the original image data. This reduction is a crucial precondition for the successful application of classification methods.

Motivated by the discussion in the last section, *three* complementary operator types have been selected to extract features with high relevance for terrain classification from the captured input images. These are:

1. **Textural Structure** The first feature extractor is used to capture the textural structure of the image. Since many relevant terrain types exhibit a distinct texture (grass, bushes or trees, for example), texture can be used as a powerful clue to discriminate them. Loosely speaking, the selected approach captures the *spatial structure* of local brightness variations, e.g. distribution patterns of brighter or dimmer pixels.

2. **Brightness Variance** The textural structure operator is invariant to the magnitude of local contrast. However, this information is very useful to distinguish between uniform image areas with low brightness variations (such as roads or sky) and areas exhibiting larger variations, such as image tiles showing vegetation. In order to retain the contrast information, a second feature extractor is included which is sensitive to image brightness variance.

3. **Color Distribution** The third feature extractor considers the overall color distribution which is ignored both by the texture and the contrast operators. As image brightness is subject to large variations in outdoor environments, the most informative component of the color distribution is the hue or color tone distribution of different image regions.

Despite the differences between the three extractors concerning the type of analyzed information, each extractor is applied to the input image using the same two-step process. Figure 6.11 illustrates this procedure.

First, each feature extractor transforms the entire image into a **feature image**. Each pixel of this feature image stores the result of applying the feature extraction with that pixel as the center. Depending on the extractor, a variably sized area of the original image may influence the extracted value, but each operator generates a single scalar as a result. Then, at the choice of the image segmentation or classification steps, an arbitrarily sized region of the *feature images* is selected. From this region, the scalar values extracted by each feature extractors are collected into separate histograms, which are finally concatenated

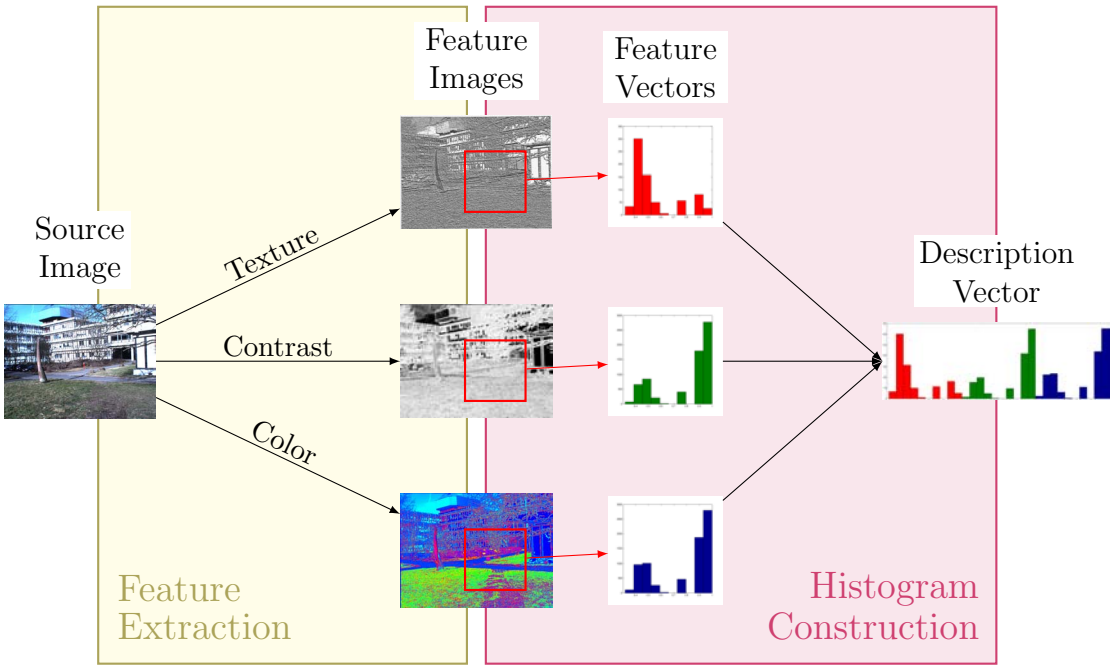


Figure 6.11: The two-step extraction process of a description vector

into a single **description vector**. This description vector characterizes the corresponding image region in a compact fashion.

The following sections present the three used feature extractors and the mathematical detail needed for their computation.

6.2.1.1 Textural Structure

Texture is a property of a surface describing its visual or tactile appearance. Visual texture consists of many simple elements created by the reflection of light from a given surface. These elements usually have some spatial relationship to each other, some kind of order which can be structured or chaotic. Also, their sizes, colors, directions, etc. can be similar or varying (both with monotonic or chaotic changes). These are characteristics of textures, but no formal definition. In fact, there is no widely-accepted formal definition yet.

Many efforts have been made to find a precise and complete definition for textures, Tuceryan and Jain [Tuceryan 98] have compiled a list of some partial definitions. In the scope of this thesis, texture is understood according to the description formulated by J.K. Hawkins:

> "The notion of texture appears to depend upon three ingredients: (i) some local 'order' is repeated over a region which is large in comparison to the orders' size, (ii) the order consists in the nonrandom arrangement of elementary parts, and (iii) the parts are roughly uniform entities having approximately the same dimensions everywhere within the textured region."

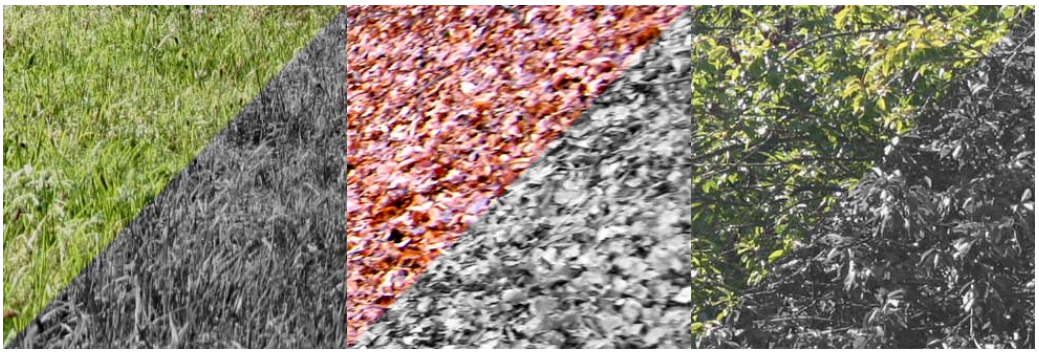


Figure 6.12: Three texture samples
From left to right, the images show grass, leaves and a tree top. To reduce the influence of lighting variances, textures are usually analyzed in grayscale.

Categories of Existing Methods

In order to remain focused on the topic of appearance-based traversability estimation, an in-depth description of existing approaches for texture analysis is omitted. Good surveys on this topic can be found in [Singh 02] or [Tuceryan 98]. Instead, a brief overview of the four main method *categories* for texture description methods is given.

Statistical methods use, as the name implies, statistical parameters to describe texture. Spatial relations of gray-values are treated as local features and the statistics are extracted from their distribution. These distributions are indeterministic under statistical texture description. Examples are co-occurrence statistics, difference histograms, gray level differences [Ojala 01], and Laws' texture energy measures. Statistical methods typically work well for many natural textures which contain some randomized noise.

Structural methods assume the existence of basic texture primitives, often called 'texels' (texture elements) or 'textons'. Texture on a macroscopic level is characterized using such texture primitives plus design rules that guide their spatial organization. Texture primitives are usually extracted by edge detection, mathematical morphology, and generalized co-occurrence matrices. Structural methods work best with man-made textures having a clear construction scheme. However, natural textures are rarely well represented by this model, which is a drawback for structural methods in outdoor applications.

A *model-based method* uses stochastical processes or other analytical models to describe texture. A parametric generative model is compiled that could have created the intensity distribution the texture consists of. Pixel-based and region-based methods exist. Examples for model-based methods are random field models, autoregressive, and moving-average.

Finally, *signal processing* or *filter methods* interpret the textured image as a two dimensional signal and submit it to a bank of filters. The filter response is used to describe the texture. Thus, the frequency content of the image is analyzed by describing the global periodicity of gray-levels in the spectrum. Spatial domain filters include Laws' masks, local linear transforms, Fourier transform, wavelet transforms, Gabor filter banks, discrete cosine transform, eigenfilters, and finite impulse response filters [Ojala 01]. A great deal of filter methods was tested and compared in [Randen 99].

Selected Approach

After considering the available techniques for textural feature extraction, the **local binary pattern** approach has been selected due to its high discriminative power and higher speed compared to other extraction methods with similar strength such as gabor filter banks [Mäenpää 03].

Local Binary Patterns (LBPs) were first mentioned in [Harwood 95] and introduced to the public by T. Ojala et al. in [Ojala 96]. Loosely speaking, the LBP value of a pixel captures the structure of local brightness variations around it. Algorithmically, the value is computed by sampling circularly around the selected pixel and setting 1-bits in the LBP value for each sample that is brighter than the center (See figure 6.13 for an example using 8 samples).

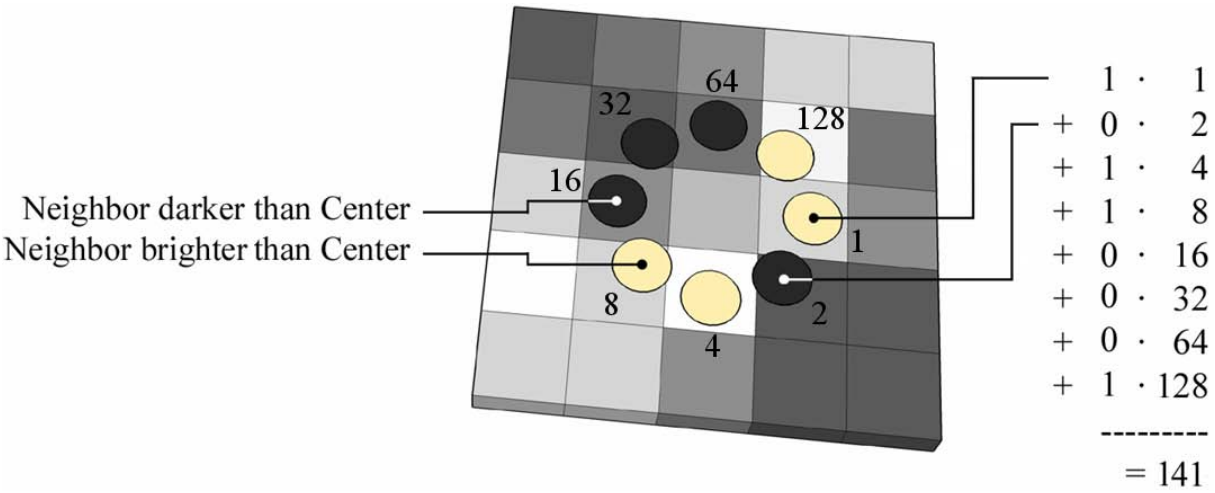


Figure 6.13: A local binary pattern with 8 samples and radius 1

Let $I(x, y)$ be defined as the intensity of the source image pixel centered at position (x, y). If the (x, y) coordinates do not address a pixel center exactly, the value of $I(x, y)$ is interpolated linearly from the intensities of the four closest pixels. Furthermore, let the intensity value of the center pixel be denoted with g_c and the intensities of the P samples around g_c (distributed circularly with radius R) be named g_i (see figure 6.14).

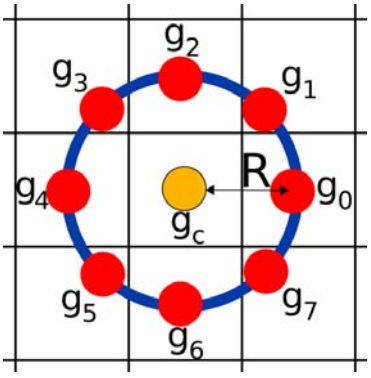


Figure 6.14: LBP computation (adapted from [Ojala 02b])

Based on these notational conventions, the sampling process can be formally described by equation 6.1.

$$g_i = I\left(x + R\cos\left(\frac{2\pi i}{P}\right), y + R\sin\left(\frac{2\pi i}{P}\right) \right) \tag{6.1}$$

With the standard signum function, defined as

$$\operatorname{sgn}(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}, \tag{6.2}$$

the return value of a LBP operator with P samples and radius R is defined by [Mäenpää 03]:

$$\mathrm{LBP}_{\mathrm{P,R}}(x, y) = \sum_{i=0}^{P-1} \operatorname{sgn}\left(g_i - I(x, y)\right) \cdot 2^i \tag{6.3}$$

The resulting scalar lies between 0 and $2^P - 1$.

As can be seen from the formulas, the LBP code of a pixel is not affected by changes in mean luminance or any monotonous scaling of image intensity, such as changes in contrast etc. [Ojala 01]. This lets local binary patterns remain relatively unaffected by illumination changes. Thus, they are well suited for outdoor applications [Mäenpää 05].

LBP codes describe the *structure* of a pixel's local neighborhood in a compact and efficient form. Thus, single LBP features can be seen as equivalent to the textons postulated by a structural texture analysis method. In fact, a local binary pattern is able to represent a variety of characteristical local structures, such as flat areas, spots, edges or edge ends (figure 6.15). Each of these patterns is assigned a unique value in the feature image.

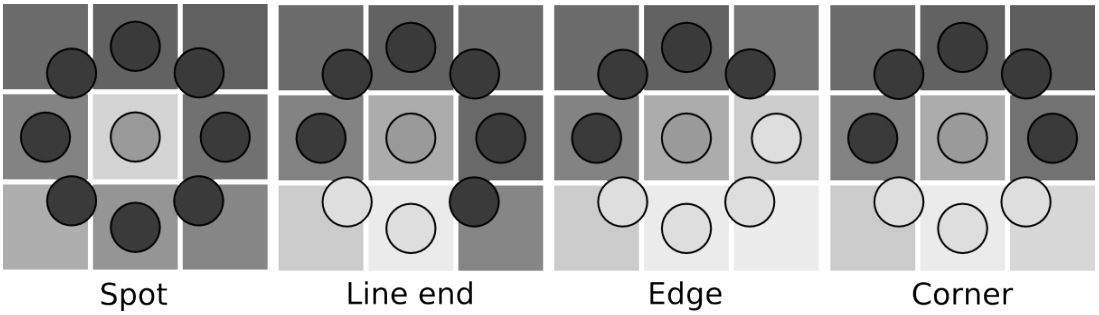


Figure 6.15: Some texture primitives representable by local binary patterns (adapted from [Mäenpää 05])

It is interesting to observe that the computation of the description vector for a given area builds a histogram of the LBP values located inside this area. In essence, this histogram is a *statistical* view of the *structural* setup of that region, embodied by the single LBP values. Thus, feature extraction based on LBP patterns can be seen as an unifying approach which exhibits characteristics of both the statistical and the structural methodologies presented earlier. In the view of the authors of the LBP technique, this fact supports the claim that local binary patterns are appropriate to characterize a wide variety of texture types and are more discriminative than approaches which fall only into one category [Mäenpää 03].

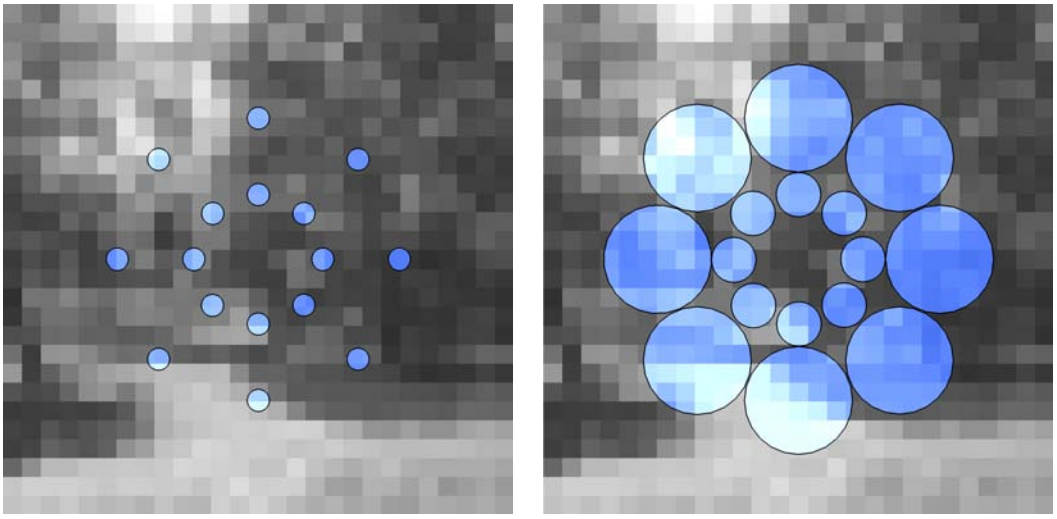
The original local binary pattern approach has undergone several modifications and extensions over time [Mäenpää 03], which offer significant increases in discrimination power. The 'multi-scale' and 'uniform' extensions presented next have proven themselves to be especially useful and are thus used by the terrain analysis algorithm.

The Multiscale LBP Extension

As pointed out by the definitions at the beginning of section 6.2.1.1, textural appearance is always linked to a specific scale of observation. When analyzed on a different scale, the texture properties of an image area may change dramatically. The texture of grass is a good example for this effect. While appearing uniform from a large distance, grass exhibits a pronounced vertical orientation if the analysis scale is reduced.

In order to improve the standard LBP and allow it to capture textural information on multiple levels of resolution, the approach has been modified by [Ojala 02b] to sample on *multiple* circles with different radii. For each circle, a separate LBP code is computed. During histogram construction, these codes are collected into separate histograms, which are then concatenated to form one large feature vector as the final result. Of course, this technique does not model pattern relations *between* sample rings. However, experiments have shown that this has little detrimental effect on texture discrimination, while it reduces the size required for the final histogram exponentially.

However, care has to be taken when sampling the source image using a radius > 1. With growing radii, the sampling points become sparsely distributed. Taking point samples at these locations does not accurately model the structure of the surrounding image area on the correct scale – the point sample can randomly hit or miss characteristic structural feature of the texture, causing instability in the obtained patterns (figure 6.16a).



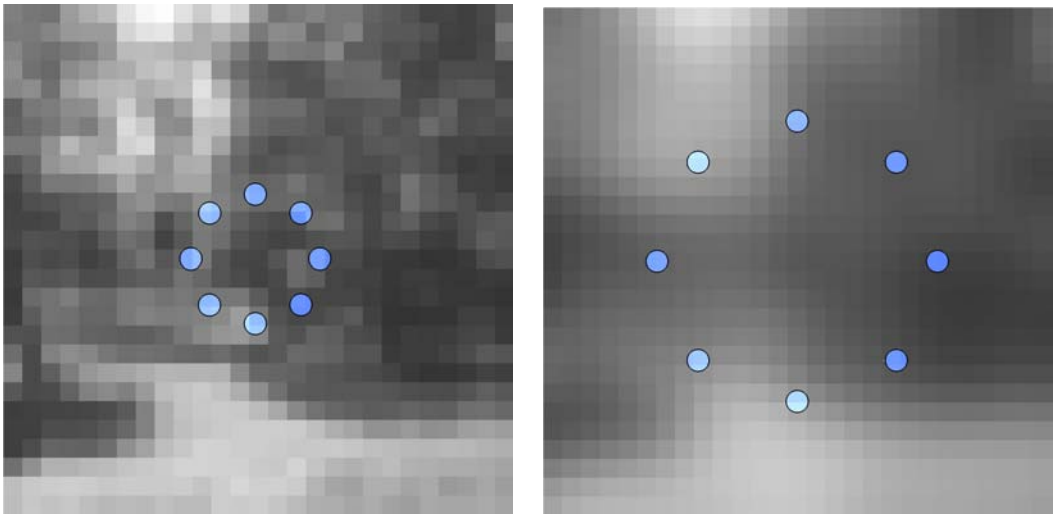
(a) Point sampling at larger radii yields unstable LBP patterns

(b) Sampling with enlarged sampling areas avoids aliasing

Figure 6.16: Larger sampling radii for multiscale analysis

To avoid this, the sampled area has to be increased over the four pixels used for linear interpolation. The idea is to expand the sampling area to circles which cover the sampled

image as tightly as possible (see figure 6.16b). While this method improves the LBP pattern's stability as desired, it increases the execution time due to the additional value summation for each sample. Fortunately, the same result can be achieved by applying the original operators on multiple versions of the input image, blurred by Gaussian filter operators with different strengths (figure 6.17).



(a) The inner ring of the multiscale LBP is sampled on a slightly blurred image

(b) The samples for the outer ring are obtained from a strongly blurred image

Figure 6.17: Blurred input images for multiscale analysis

The effective radius of the blur filters must be chosen in accordance with the LBP operator radius and the number of its samples to ensure that the sampled regions cover most of the image without overlapping each other (see figure 6.18). The blurring kernel sizes can be calculated from the innermost radius R_1 and the amount of samples per ring P_n. The other radii and the sizes of the blurring kernels depend on the choice of the first radius. The calculation of the blurring kernel size is done as follows [Mäenpää 03]:.

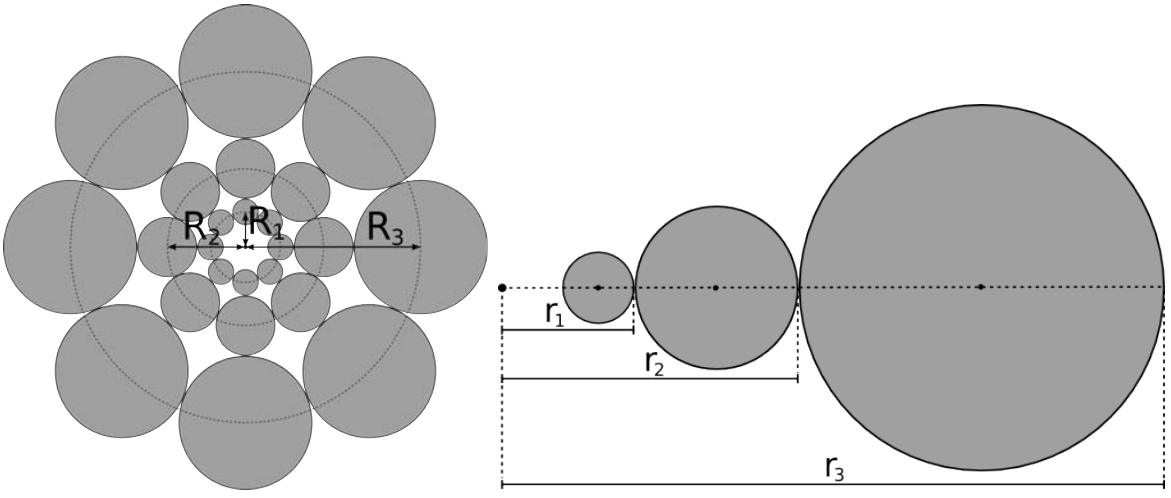


Figure 6.18: *Effective areas* for LBP with three scales: areas of influence

$$r_1 = R_1 \left(sin\left(\frac{\pi}{2P_1}\right) + 1 \right) \tag{6.4}$$

$$r_n = r_{n-1} \left(\frac{2}{1 - sin(\pi/P_n)} - 1 \right), n \in \{2, \ldots, N\} \tag{6.5}$$

$$R_n = \frac{r_n + r_{n-1}}{2} \tag{6.6}$$

The discrete sizes in pixels of the blurring kernel mask can be computed by

$$w_n = 2\lceil \frac{r_n - r_{n-1}}{2} \rceil + 1. \tag{6.7}$$

The notation for multiscale LBP operators is a straightforward extension of the nomenclature used already. The operator name $\mathrm{LBP}_{\mathrm{P_1+P_2+P_3,R_1}}$ specifies a multiscale LBP operator with three sampling rings having P_1, P_2 and P_3 samples and the innermost radius of R_1.

The Uniform Pattern LBP Extension

It has been observed by [Mäenpää 03] that certain LBP codes are extracted much more frequently than others. The corresponding intensity patterns seem to be fundamental building blocks of local image texture. Interestingly, these codes are characterized by having at most *two transitions* from one to zero or from zero to one in the circular binary code (figure 6.19). The corresponding patterns have been termed **uniform** patterns.

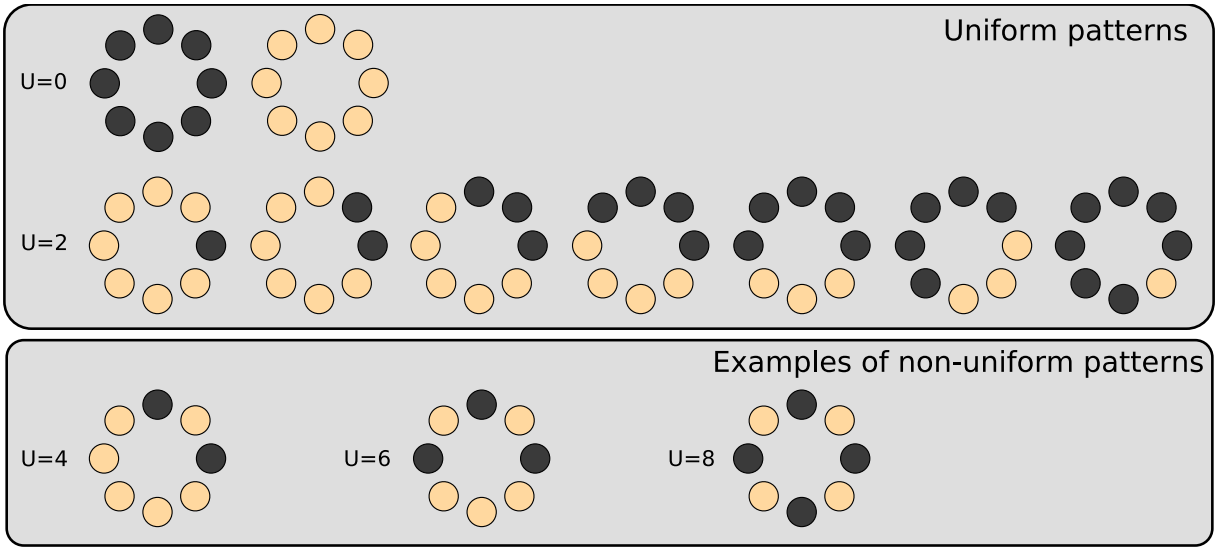


Figure 6.19: Examples for uniform and non-uniform LBP (from [Mäenpää 05])
U is the number of transitions from one to zero or vice versa in the binary patterns.

The unequal distribution of LBP patterns can be exploited to significantly reduce the number of codes that a LBP operator can produce without loosing much information. For this, a mapping is constructed which takes the 2^P possible output codes of an $\mathrm{LBP}_{\mathrm{P,R}}$ operator and maps each code that corresponds to an *uniform* pattern to an unique value $0 \ldots n$. Any other (non-uniform) code is mapped to the value $n+1$. It can be shown that

with P samples, $P(P-1)+2$ uniform patterns exist and thus $n = P(P-1)+2$. With this mapping function, the output codes of the $\text{LBP}_{8,1}$ operator shown in figure 6.13 are reduced from initially $2^8 = 256$ distinct LBP codes to only $8(8-1)+2+1 = 59$ different result values.

The benefit of this mapping is a vast reduction of the histogram size needed in the second step of the feature extraction process. On the one hand, the resulting description vector requires less storage and subsequent similarity comparisons become *a lot* faster. A second, maybe even more important advantage of using uniform LBP operators is the fact that the evaluation of small image regions (with about 16x16 pixels or less) results in only sparsely filled histograms when using the normal operator. Sparse histograms tend to be unstable when compared to each other. By reducing the bin count, this problem is reduced dramatically.

To verify that not too much information is lost when using the uniform pattern mapping on real-world images, the occurring LBP-patterns and their coverage by uniform patterns was examined on several sample images, using both a standard and a multiscale LBP variant. The results are shown in table 6.1.

Operator	Images	Recorded by	Uniform patterns
$\text{LBP}_{8,1.0}$	9 (set A)	robot stereo system	90.5 %
	52 (set B)	SLR camera	87.8 %
$\text{LBP}_{8+8+16,1.0}$	9 (set A)	robot stereo system	85.7 %
	52 (set B)	SLR camera	78.9 %

Table 6.1: Experimental determination of uniform pattern coverage

On average, the retained uniform patterns accounted for more than 80% of all extracted patterns. At the same time, the bin count was reduced by 76% for the single scale operator and an overwhelming 99.4% for the multiscale LBP. Thus, the usage of uniform LBP operators is a very appropriate means to decrease the final feature vector size. For multiscale LBPs, it can even be seen as a vital precondition in order to obtain usable feature vectors with *any* significance on reasonably sized image areas. After all, filling a $2^8 + 2^8 + 2^{16} = 66048$ bin histogram with $64^2 = 4096$ values from a 64 x 64 pixel region does not yield any useful feature vector. Using only $59 + 59 + 243 = 361$ bins in total, a much more manageable feature set is obtained.

Exact Operator Specification

Since the LBP operator and its extensions can be tuned in many different ways, some thought has to be invested into the exact operator configuration used in the terrain classification system. The final decision has been guided by several preliminary experiments as well as published results. Both results from applications of the LBP technique on benchmark databases [Ojala 02a] [Mäenpää 03] [Pietikäinen 04] or real applications like image retrieval [Takala 05], quality inspection [Mäenpää 05] or facial expression recognition [Feng 05] have been considered.

In the experiments, the multiscale LBP operator consistently yielded better results than the single scale LBP. Also, the uniform pattern extension has been validated as a highly

useful means to reduce computational complexity and stability of the multiscale technique. Due to these advantages, both extensions are selected for the final operator. Furthermore, the best results in our experiments have been obtained using *three* sampling rings with 8, 8 and 16 samples and a starting radius of 1.3. This layout has additional computational benefits, since the results can be stored compactly into a single feature image with 32 bits per pixel. Figure 6.20 shows a schematic view of the image areas that contribute to each sample taken by this $\mathrm{LBP}_{8+8+16,1.3}$ operator.

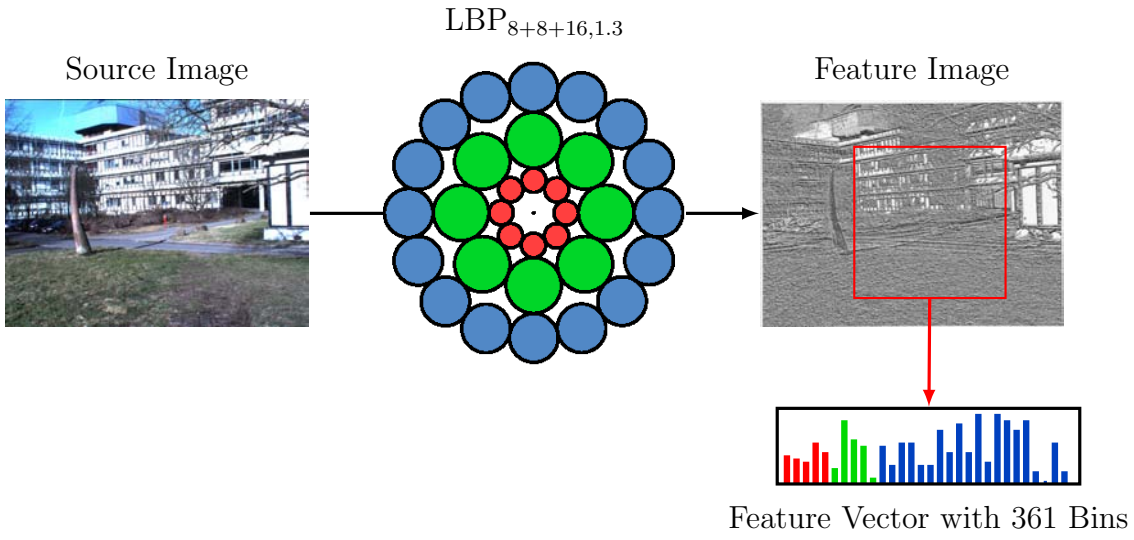


Figure 6.20: The final LBP operator used for textural feature extraction

In accordance with the uniform mapping extension, the $\mathrm{LBP}_{8+8+16,1.3}$ operator produces 59 different values for the first two sample rings with 8 samples and 243 distinct codes for the largest ring with 16 samples. After building separate histograms for each ring and concatenation of the results, a 361-dimensional feature vector is produced for each specified image region. As a final note, in order to reduce the impact of robot rotation on the extracted texture features, the estimated roll angle of the CCS compared to the WCS is used to rotate the operator's sampling positions accordingly.

6.2.1.2 Brightness Variance

Since the LBP operator only evaluates the signs of the brightness differences between two sampled areas and discards their actual magnitudes, it does not capture any information about the local contrast distribution of the image. However, this information can be very useful to distinguish between uniform image patches with low brightness variations (such as roads or sky) and areas exhibiting larger variations, such as image tiles showing vegetation.

In order to retain the contrast information, a second feature extractor is applied to the input image, which has been developed to quantify local brightness variance [Mäenpää 03]. The $\mathrm{VAR}_{\mathrm{P,R}}$ operator uses the same sampling strategy as the LBP operator introduced earlier. However, the operator result depends upon the *squared sum of brightness differences* between the average brightness of all samples and each single sample intensity.

More precisely, the $\mathrm{VAR}_{\mathrm{P,R}}$ value of a pixel is calculated from the intensity values g_i ($i = 0, \ldots, P-1$) of P samples drawn, re-using the sampling setup introduced in figure 6.14. From these samples, a variance estimate is derived:

$$VAR_{P,R} = \frac{1}{P}\sum_{i=0}^{P-1}(g_i - \mu)^2 \text{, where } \mu = \frac{1}{P}\sum_{i=0}^{P-1} g_i \,. \tag{6.8}$$

The result describes the local image contrast in a circular neighborhood around the center pixel. It is both invariant against image rotation and linear brightness changes. By applying the operator to all source image pixels, a feature image can be computed as for the textural structure operator. However, the image must store a floating-point value per pixel, since equation 6.8 returns real numbers.

Figure 6.21 shows a visualization of such a feature image.

(a) Source image (b) VAR feature image

Figure 6.21: Feature extraction using the VAR operator

The fact that the results of the variance operator are real numbers forces the introduction of an additional quantization step before histogramming of image areas can take place. During quantization, operator values that lie within a certain *value range* are mapped to discrete values $1 \ldots n$. Although a monotonous mapping is not strictly required, it is used in practice due to its intuitivity.

The number of quantization levels n is a free parameter of the VAR operator and can be chosen according to external requirements. If n is fixed, a monotonous mapping still leaves the question how to choose the size of each quantization range. Experience shows that if all ranges were made equally broad, the resulting histogram would become very unbalanced: some bins would be very full and others very empty. This would reduce the information content and usability of the produced feature vector.

In order to obtain a more balanced histogram, [Mäenpää 03] proposes to adjust the quantization ranges according to the distribution of VAR values in some typical training images. The used technique is reminiscent of the well known histogram equalization method. In essence, the ranges are calculated as follows. First, the VAR operator is used to transform all training images into feature images. Then, all VAR values are stored in a linear array and sorted according to their magnitudes. The total number of VAR values N divided by the number of desired quantization levels n yields the desired bin count for a balanced

histogram: $b = N/n$. Using the sorted array, the quantization borders are now simply determined by each b-th array entry. After training, the determined quantization ranges are saved to a file and can be reloaded upon system start.

Exact Operator Specification

The VAR operator can be extended for multiscale analysis just as the LBP operator and is adjustable using the three parameters R, P and n. Again, some preliminary experiments have been conducted to determine good settings for these.

In contrast to the textural structure operator, the analysis of multiple scales did not provide a large benefit. Also, the variation of sample numbers or quantization levels within reasonable limits had no pronounced effects on performance. However, a sample radius of ≈ 2 showed some advantage over smaller radii. Due to these observations, the $\mathrm{VAR}_{8,2.0}$ operator with $n = 256$ quantization levels has been chosen to serve as feature extractor for image brightness variance.

6.2.1.3 Color Distribution

The third feature extractor deals with the overall color distribution of image patches. Up to now, this part of information is ignored both by the texture and the contrast operators. As image brightness is subject to large variations in outdoor environments, the most informative component of the color distribution is the hue or color tone of the image tile. In order to gather this information into a single channel, the input image is converted into the HSV (Hue, Saturation, Value) color space before the actual feature extraction starts.

In this color space, the hue channel specifies the dominant spectral component of the pixel color. This value is usually represented in a circle, where each angle corresponds to a pure color. Saturation is the degree to which that color is undiluted, i. e. a measure of how much white is added to it. The value channel quantifies the overall brightness of the image pixel. Together, the three components create a cylindrical color space which is often represented as an inverted cone (see figure 6.22).

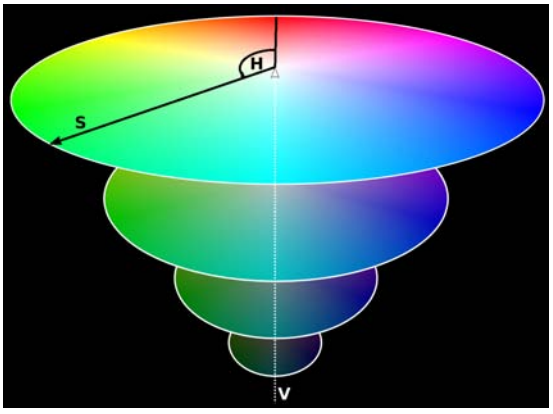
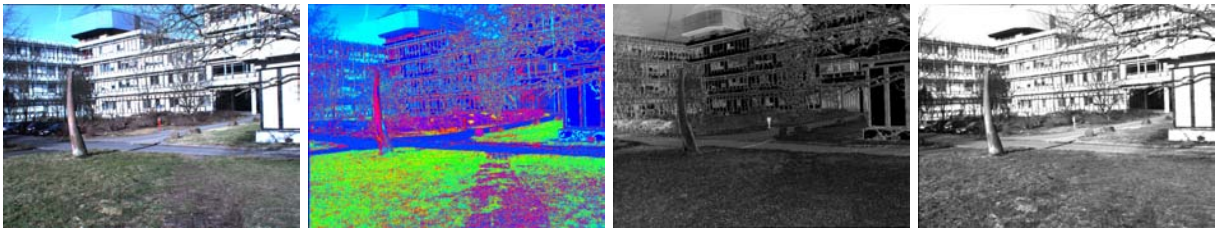


Figure 6.22: The HSV color space

The conversion from the RGB color space (with $0 \leq$ R, G, B ≤ 255) to HSV is realized with the following formulas:[1]

$$
\begin{aligned}
V &= \max(R, G, B) \\
S &= \begin{cases} (V - \min(R, G, B)) \cdot 255/V & \text{if } V \neq 0 \\ 0 & \text{if } V = 0 \end{cases} \\
H &= \begin{cases} (G - B) \cdot 60/S & \text{if } V = R \\ 180 + (B - R) \cdot 60/S & \text{if } V = G \\ 240 + (R - G) \cdot 60/S & \text{if } V = B \end{cases} \\
H &= H + 360\,, \text{ if } H < 0
\end{aligned}
$$

Figure 6.23 shows an illustration of the content stored by each of the three HSV channels.



(a) Original image (b) Hue channel (c) Saturation channel (d) Value channel

Figure 6.23: HSV color channels

After conversion, the feature image of the color feature extractor is computed. For this, three quantization parameters L_H, L_S and L_V are used. Each parameter determines the number of value ranges that will be distinguished for the H, S and V channel in the feature vector. Then, the color value (H, S, V) of each source pixel is first normalized to ranges between $[0 \ldots 1[$ and then transformed into a discrete value F by computing

$$F = \lfloor V \cdot L_V \rfloor + \lfloor S \cdot L_S \rfloor \cdot L_V + \lfloor H \cdot L_H \rfloor \cdot L_S \cdot L_V \tag{6.9}$$

. $\lfloor x \rfloor$ signifies the standard mathematic floor function.

By choosing L_S, L_V relatively low and L_H much higher, more importance can be given to hue than to the saturation or value components of a pixel color. With these settings, variations in illumination have only a small impact on the extractors result value, while even small changes in hue are represented in the feature vector. This technique can be used to tune the color operator and make it more robust especially in outdoor applications.

Exact Operator Specification

Again, the parameter selection for the color distribution operator has been guided by some preliminary experiments. At the end, the operator with $L_H = 18$, $L_S = 3$ and $L_V = 3$ has been identified as the best compromise between discriminative power and sensitivity to illumination changes. As can be seen in equation 6.9, the resultant feature vector has a dimension of $18 \cdot 3 \cdot 3 = 162$, which is comparable with the other two feature extractors.

[1] These formulas have been taken from the OpenCV API, see http://www.cs.indiana.edu/cgi-pub/oleykin/website/OpenCVHelp/.

6.2.2 Image Segmentation

The feature extraction stage presented in the last section allows to obtain a description vector for arbitrary image regions, characterizing the area's texture, contrast and color in a compact fashion. This capability could be used directly for terrain traversability estimation by training a suitable classifier and applying it to a regular grid of image patches such as shown in figure 6.24.

(a) Original image (b) Image patches to be classified

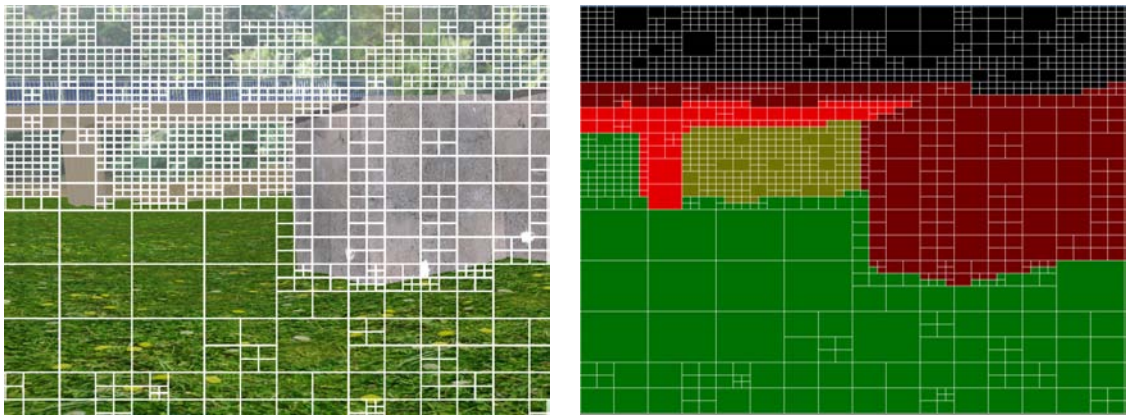
Figure 6.24: Direct terrain classification

However, such a rigid segmentation scheme would have to create rather small regions in order to adequately capture variations in terrain traversability. This in turn would lead to a large number of patches and thus many time-consuming classifications. Also, it would reduce the stability of the feature histograms used as description vector. Small appearance differences in an image patch would result in large relative changes of the corresponding description vector. As a consequence, the traversability classification step would become more difficult and error prone.

In order to circumvent these problems, a more intelligent strategy is required to divide the source image into fewer and larger regions wherever possible. Ojala et. al. showed in [Ojala 99] that a split-and-merge segmentation can be used successfully to achieve this goal in a computationally efficient way. While the original technique has been proposed for unsupervised image segmentation, it has been adopted in the context of this thesis as a preparational step that precedes image classification.

In short, the algorithm proceeds in two distinct steps, termed **split** and **merge**. Splitting recursively subdivides the source image into quadratic tiles that are homogeneous with respect to their description vector, i.e. if the tile was split up further, the description vectors of the smaller tiles would all be very similar to each other. Once splitting has terminated, many homogeneous quadratic tiles of different sizes exist (see figure 6.25a). Now, adjacent regions are iteratively merged as long as their description vectors are similar enough. As a final result, arbitrarily shaped regions which share similar description vectors are produced (see 6.25b). Compared to other segmentation techniques, split and merge increases the processing speed while maintaining small memory requirements [Horowitz 76].

Both the region splitting and merging stages rely on a measure that quantifies the similarity of two description vectors. A suitable metric for this is presented next in section 6.2.2.1.



(a) After splitting (b) After merging

Figure 6.25: Identification of homogeneous image regions using Split & Merge

After this, some details on the efficient implementation of the splitting and merging steps introduced above are given.

6.2.2.1 A Similarity Measure for Description Vectors

Since the description vector of an image region is constructed by concatenating feature histograms, histogram comparison metrics are natural candidates for a good similarity measure. [Rubner 01] published an excellent review and experimental comparison of the most prominent metrics. The reviewed candidates are well known standard metrics such as *histogram intersection* and the *Earth Movers Distance*, heuristic measures including the *Minkowski-form* and *Weighted-Mean-Variance*, divergence measures stemming from information-theory such as the *Kullback-Leibler* or *Jenson-Shannon divergences* and finally nonparametric test statistics like χ^2 (Chi Square) and the *Kolmogorov-Smirnov* distance. However, not all of these metrics are suitable for the comparison of description vectors due to the lack of any consistent relation *between* adjacent descriptor bins. Among others, this prohibits the application of the Earth Movers Distance or the Weighted-Mean-Variance metric.

The authors of the local binary pattern approach advocate the use of the Kullback-Leibler divergence as a similarity pseudo-metric for the tasks of texture classification. Given two description vectors P, Q of length n, the Kullback-Leibler Divergence $\mathrm{KL}(P, Q)$ is defined as

$$\mathrm{KL}(P, Q) = \sum_{i=1}^{n} P(i) \cdot \log \frac{P(i)}{Q(i)} \tag{6.10}$$

KL quantifies the relative entropy of a discrete sample distribution with probability function P with respect to a second discrete model distribution with probability function Q. The higher this entropy, the less similar the two distributions are.

However, the KL measure has several drawbacks. First, it is not bounded and approaches neg. infinity for $P(i) \to 0$. Second, it is not symmetrical, so that the order of P and Q are important - however, this has no reasonable interpretation in the context of image segmentation.

The *Jenson-Shannon Divergence* (JD) overcomes the symmetry disadvantage. It is based on KL but uses the arithmetic middle for each of the compared distributions:

$$\mathrm{JD}(P,Q) = \frac{1}{2}\,\mathrm{KL}(P,M) + \frac{1}{2}\,\mathrm{KL}(Q,M) \quad \text{where} \quad M = \frac{1}{2}\,(P+Q) \tag{6.11}$$

It can easily be shown that $\mathrm{JD}(P,Q) = 0$ if and only if $P = Q$. In all other cases, $\mathrm{JD}(P,Q)$ becomes larger the less similar the description vectors P and Q are.

In order to avoid the unboundedness of the KL measure, Ojala proposed in [Ojala 96] to set the bin count to 1 for any bins $P(i), Q(i)$ that contain 0 initially. This heuristics is also applicable to JD. The modified Jenson-Shannon Divergence measure is both numerically stable and symmetrical. Also, its efficiency as a metric for unsupervised image segmentation has been documented in [Rubner 01]. Due to these benefits, it has been chosen as the similarity measure for description vectors in this thesis. As a side note, an interesting other similarity measure has been researched as well. However, it is excluded in this presentation because it is not used in the final appearance based estimation method. Details concerning the other measure can be found in [Rauber 08].

Furthermore, to avoid skewing the similarity measure when description vectors stemming from regions with different sizes are compared, a **normalization** of the description vectors P and Q is performed before *any* JD measure is computed. This normalization ensures that the sum of all values in each description vector equals one.

6.2.2.2 Region Splitting

The purpose of the splitting stage is to subdivide the input image into smaller regions as long as these regions are heterogeneous, e.g. composed of inner regions with dissimilar description vectors. Once no heterogeneous regions remain, any further subdivisions would only generate regions with very similar description vectors. From the perspective of later terrain classification, these additional regions would fall into the same class and offer no new information. Thus, their creation can be safely omitted to save computational effort.

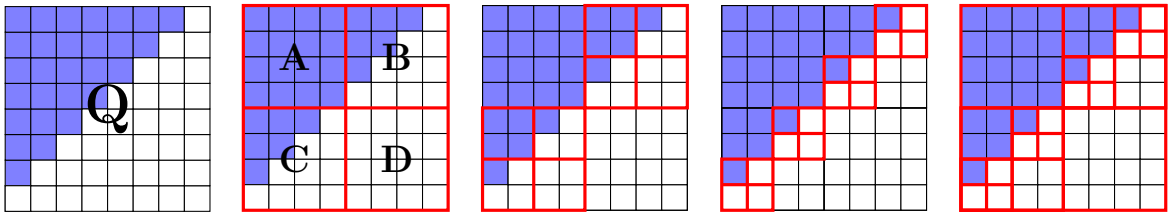
Splitting is implemented as a recursive procedure, starting with the whole image as one single region. All image regions are recursively subdivided into four quadrants with halved widths and heights *if* the JD similarity measure between any of the newly created four quads A, B, C, D exceeds a predetermined threshold. The threshold is checked against the maximal dissimilarity between all 6 possible quad pairs as shown in algorithm 11.

Algorithm 11: Recursive splitting

procedure Split(`Quad` Q)

1: Construct four `Quads` A, B, C, D from Q with halved width and height
2: Compute description vectors for A, B, C, D
3: *max_dist* = max(JD(A,B), JD(A,C), JD(A,D), JD(B,C), JD(B,D), JD(C,D))
4: **if** *max_dist* > *split_threshold* **then**
5: Split(A); Split(B); Split(C); Split(D)
6: **else**
7: Discard A,B,C,D
8: **end if**

Figure 6.26 shows a schematic drawing of the regions created by the splitting approach. Each image region is divided further until the remaining area is homogeneous (e.g. has the same color).



(a) Input image (b) Depth 1 (c) Depth 2 (d) Depth 3 (e) Final result

Figure 6.26: Regions created through recursive splitting

In the developed implementation, a *quadtree* is used to store and manage the created regions. For the splitting threshold, a conservative setting has been determined empirically. With this setting, the image is typically oversegmented - however, this can be corrected both by the subsequent merging step *and* the later classification stage. If the threshold was chosen too insensitive instead, too few regions would be constructed. This could not be corrected at a later stage.

6.2.2.3 Region Merging

Once the splitting stage has concluded, adjacent regions with high similarity are merged together until a termination criterion (see below) is met. This step breaks the quadtree structure and allows to form regions with arbitrary shapes.

In contrast to many other implementations of split and merge algorithms which perform local region growing, the merging strategy used in this thesis always fuses the two neighboring regions with the globally highest similarity [Ojala 99]. This approach is computationally more expensive than local clustering, but it removes the necessity to select seed regions from where merging starts. This is highly desirable for unsupervised and repeatable operation of the image segmentation stage.

However, the merging algorithm needs to know the JD similarity measures of *all* pairs of neighboring regions in order to select the one with the lowest description vector divergence. To compute all of these distances in each step of the merging stage would be prohibitively expensive. Therefore, a more efficient algorithm has been developed in the scope of this thesis. It uses a cache of all region pair similarity measures. Once a region pair has been merged, only the cache of regions which lie adjacent to either one of the merged couple needs to be updated [Faust 08].

The cache is implemented as a so called *neighborhood list*. Each element of this list stores pointers to two neighboring regions and the corresponding divergence measure. Initially, it is constructed from the quadtree that results from the split stage. By exploiting its regular spatial layout, it is relatively easy to determine whether two quads are neighbors using *tesseral arithmetics* [Faust 08]. If they are, the pair is inserted into the neighborhood list along with their similarity value. During the merge stage, the neighborhood list is sorted according to region similarity. Then, the pair of regions (N,M) with lowest divergence is merged into a new region R. Now, all entries in the neighborhood list which contain

pointers to either N or M need to be reevaluated. For one, the old region pointers need to be replaced with a pointer to R. For another, the similarity values must be updated.

In order to perform this similarity value update, the description vector of the region R must be determined first. Since description vectors are actually histograms of single features, the vectors of N and M can be simply added.

In order to quickly retrieve all list elements which point to N or M, the neighborhood list elements are augmented with two additional **threading pointers**, one for each of the two region pointers. The threading pointer of N leads to another list element which contains N and is guaranteed to loop around only after all list elements with N have been traversed. The same holds for the threading pointer of M. Using this threading, all list elements which need to be updated can be accessed in linear time without needing to recompute neighborhood relations between arbitrarily shaped regions.

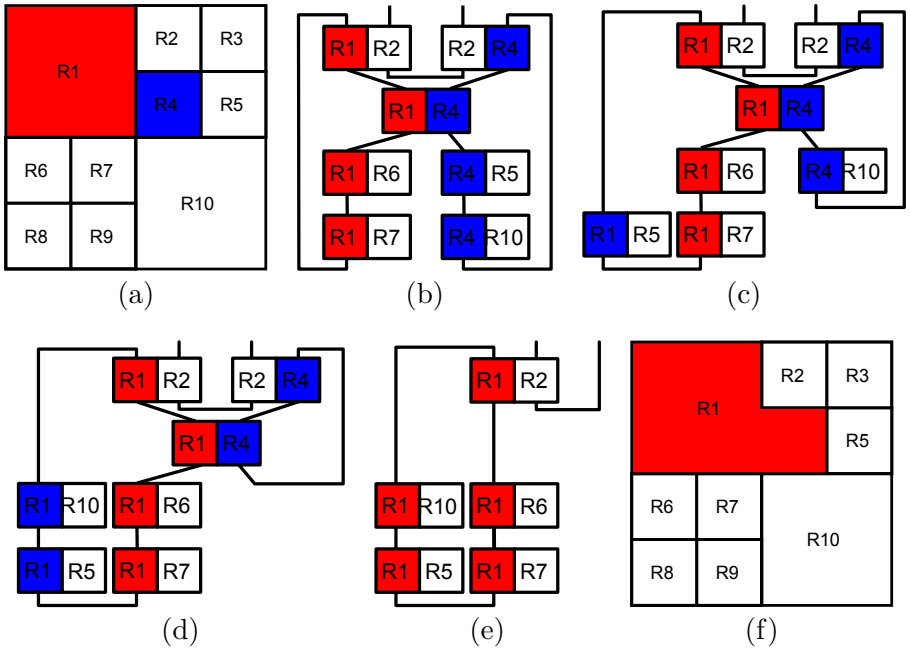


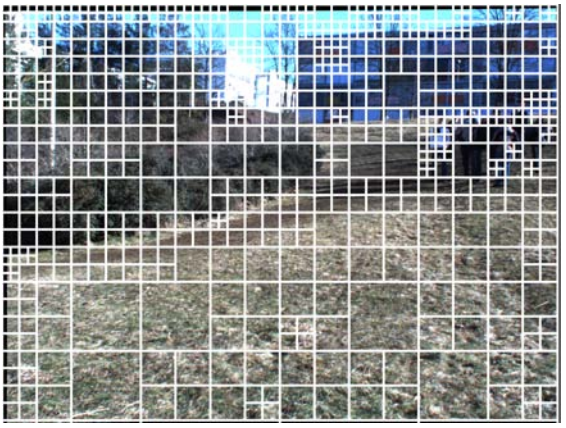
Figure 6.27: Update of the neighborhood list threading after region merging

Whenever regions are merged, the threading needs to be updated as well as the similarity measures. Figure 6.27 illustrates this updating process. Here, regions R1 and R4 are about to be merged (figure 6.27a). To update the existing threading (figure 6.27b), the elements in R4's threading list are successively moved into the list of R1 (figure 6.27c - 6.27e). During the move, all pointers to R4 are replaced by pointers to R1. If this results in a duplicated region pair such as with (R2, R4), the element containing the duplicate is removed completely from the neighborhood list and the threading (figure 6.27e). Now, the merging step can be reiterated. Merging stops if the *ratio* between the JD divergence of the current merge pair and the divergence of the last pair exceeds a second threshold. See [Ojala 99] for a justification of this technique. In essence, it allows to formulate a more general threshold that is less dependent on the actual image content.

Figure 6.28 shows the typical results obtained by the presented unsupervised split and merge image segmentation when applied to a real input image. As parameters, a splitting

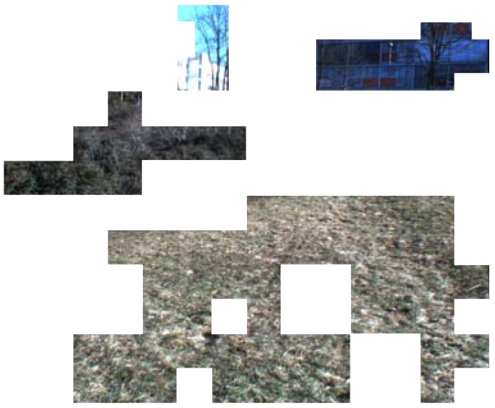
threshold of 1500, a merging threshold ratio of 1.1 and a minimal quad size of 8x8 pixels has been used. Out of the $1600 \cdot 1200/(8 \cdot 8) = 30000$ possible image patches, the splitting shown in figure 6.28b contained 721 patches. The subsequent merging stage grouped these patches into 26 regions.

(a) Input image



(b) Quadtree after split stage

(c) Regions after merge stage



(d) Four prominent, homogeneous regions

Figure 6.28: Split & Merge image segmentation in practice

As can be seen in figure 6.28c, the regions that remain after the conclusion of the merging stage are still somewhat oversegmenting the source image. However, the formed regions (four examples are shown in figure 6.28d) are relatively homogeneous in terms of appearance. Thus, the segmentation has fulfilled its job as a preparational step for terrain classification: For one, it has significantly reduced the number of region classifications (from 30000 down to 26!) that need to be performed. For another, it has produced larger, homogeneous regions with more stable description vectors.

6.2.3 Terrain Classification

The last major step that remains to be performed by the appearance-based terrain traversability estimation algorithm is to map the description vectors of the segmented image regions onto traversability scores usable for path planning. In order to reduce the complexity of this mapping task and simplify the experimental validation of the developed

approach, only *binary* traversability scores are considered. With this simplification, each image regions now only needs to be *classified* into either traversable (traversability score 0) or into impassable (traversability score 1) terrain.

Classification tasks such as this have been studied extensively and many machine learning approaches have been proposed. The following three sections present the deliberations that led to the selection of a suitable classification algorithm, its supervised training with 'ground-truth' images labeled by a human expert and the subsequent application of the resulting classifier for unknown region descriptors.

6.2.3.1 Classifier Selection

In order to select the classification algorithm that is most suitable for the task at hand, one can draw upon many existing surveys and publications that characterize the inherent benefits and drawbacks of different approaches.

For instance, Liu et. al. [Liu 05] distinguish two broad categories of classification algorithms. First, there are **statistical** classifiers, which try to infer characteristical features for the members of each class from the labeled training samples. This class can be further subdivided into parametric and nonparametric classifiers. Nonparametric classifiers such as the Parzen window method and k-nearest neighbor (k-NN) make no assumptions about the distribution of the class samples in the feature space. However, this independence comes at a price, as usually all training data needs to be stored and used to classify an unknown sample.

Parametric statistical classifiers on the other hand make certain assumptions about the distribution of features in the feature space. Examples are Bayesian learning and Mahalanobis classifiers. However, they allow to compress the learned information based on the assumed feature distribution and avoid storing the entire training set.

While statistical classifiers define a certain class based on features that are shared by all members, **discriminative** classifiers also exploit features that differentiate distinct classes. A prominent representative is the support vector machine, a binary classifier that separates classes in feature vector space by maximizing the geometric margin between them. Another group of prominent classification techniques are artificial neural networks.

The characteristics of both classifier types differ substantially. For example, adding samples typically requires complete retraining with all samples for discriminative classifiers. In contrast, some statistical classifiers such as k-NN allow to add further samples without retraining. If extensive training sets are available, discriminative methods normally exhibit higher classification accuracy. However, some statistical methods can lead to better results for small sample sizes. Discriminative methods are generally more susceptible to outliers, while at least parametric statistical classifiers are rather resistant to them.

Table 6.2 summarizes these differences very coarsely.

Given the classifier properties in table 6.2, several arguments favor the use of a nonparametric statistical classifier for the envisioned task. For one, the fact that only $10 - 30$ regions are segmented from each training image indicates that only relatively few (in the range of $100 - 500$) samples will be available for training. Also, some classifiers of this type allow to easily add more samples online using a self-supervised learning technique

Classifier Type	Statistical	Discriminative
Training time complexity	linear	square
Time needed for Classification	higher	lower
Adding samples on-the-fly	possible for some	not possible
Accuracy with large training set	not so good	better
Accuracy with small training set	better for some	worse generalization
Handling complexity	more parameters	fewer parameters
Outliers	parametric: resistant nonparametric: susceptible	susceptible

Table 6.2: Classifier characteristics (deduced from [Liu 05])

(see section 6.4). Finally, it is not handicapped by the fact that the distribution of class members in feature space is hard to guess for the given description vectors.

Due to these benefits, the very popular nonparametric, statistical **k-nearest neighbor classifier** has been selected for terrain classification. On top of the already mentioned benefits, k-NN is also robust against noisy training data which has to be expected in outdoor environments.

6.2.3.2 k-Nearest Neighbor Classification

The k-nearest neighbor classifier is based on the straightforward idea that samples with similar feature vectors belong to the same class. A sample with an unknown feature vector is thus classified based on the k known training samples with the most similar ('closest' given the used similarity metric) feature vectors. Normally, the k class labels of these samples are retrieved and a majority vote serves as classification result for the test sample (see figure 6.29). Common implementations of k-NN classifiers are based on a database that stores all training samples together with their assigned class labels. For some distance metrics, efficient data structures can speed up the retrieval of the test sample's nearest neighbors from the database.

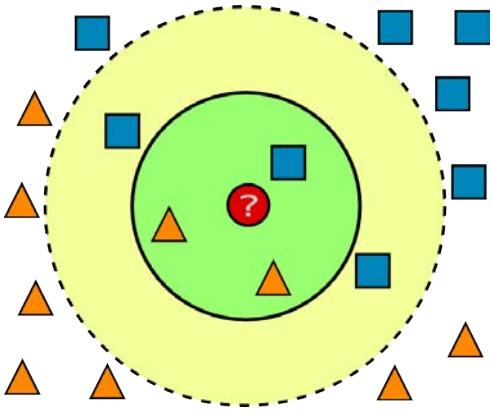


Figure 6.29: Classification with k-Nearest Neighbor
The red sample in the middle is the test sample to classify. Using majority voting, it is assigned to the triangle class for $k = 3$ and to the rectangle class for $k = 5$.

The choice of the parameter k is important. Larger values for k reduce the effect of noisy feature vectors or outliers. Smaller k lead to more adaptive estimates especially with small

training sets, but the danger of over-fitting and susceptibility to outliers is higher. The best choice is data-dependant and can be determined empirically or by techniques such as cross-validation. During preliminary testing using a set of representative input images, setting $k = 5$ has been determined as a good choice and is therefore used subsequently.

The training process of a k-NN classifier is trivial. Given a new training sample, it suffices to store the sample's feature vector together with its class label into the classifier's database. A consequence of the lack of abstraction during training is that k-NN classifiers suffer from quite high computational costs during the classification phase. For each query, the distance between the new sample and all stored samples has to be computed. However, because the number of training samples is expected to remain small, this drawback has been deemed acceptable.

Using a Classifier Ensemble instead of a Single Classifier

Several researchers have observed that classification based on multiple extracted features (such as in the present case) can be improved by combining the output of an ensemble of classifiers, each trained on just one extracted feature type, instead of training one single classifier on the concatenated feature vectors [Kittler 98]. Possible reasons for this are that classifiers which are responsible for just one type of features can adjust themselves better to the characteristics of that specific feature.

As this line of argumentation appears reasonable, the ensemble approach has been adopted for the terrain classification task at hand. Thus, 3 k-NN classifiers C_1, C_2, C_3 are used in the following, one to classify the LBP textural features, one to classify the VAR contrast features and the third one to determine the terrain class based on the color features. Details on how the outputs of the classifiers are combined are given in section 6.2.3.4.

6.2.3.3 Classifier Training

The three k-NN classifiers are initially trained using supervised training based on example images. For each image, traversable and untraversable regions are marked by a human expert in order to provide the required class labels. In practice, a colored **mask image** is drawn for each training image such as shown in figure 6.30b.

(a) Sample image (b) Mask image (c) Mask overlay

Figure 6.30: A sample image overlaid with its class labels
Red color is used to mark untraversable image regions, green marks traversable areas.

During training, all available sample images and the corresponding mask images are loaded. Each sample image is segmented using the presented split and merge algorithm.

For each of the formed regions, the color of the mask image is checked within the occupied area. If more than 75% of the region's area is marked either traversable or untraversable, the region is deemed valid and the label corresponding to the dominant color is retrieved. Otherwise, the region covers both traversable and untraversable areas, maybe due to inappropriate segmentation or insufficient discriminative power of the extracted features. In any case, such ambiguous regions do not constitute good training samples and are discarded. For all valid regions, the description vector is retrieved and split into the feature vectors created by the textural structure, contrast and color feature extractors. Each feature vector is then associated with the true class label and added to the corresponding k-NN classifier as a new training sample.

6.2.3.4 Classification

In order to identify traversable and non-traversable regions in a new image during robot operation, the feature extraction and image segmentation procedures are performed on it as described in the previous sections. Then, for each segmented image region I, the description vector $D(I)$ is retrieved and split into the three feature vectors $D_1(I)$, $D_2(I)$ and $D_3(I)$ of the textural structure, contrast and color feature extractors. After this, the three k-NN Classifiers C_i are queried with the corresponding feature vector $D_i(I)$. Each classifier produces a list of results R_i, which contains k pairs of class labels $l_1 \dots l_k \in \{traversable, untraversable\}$ and feature vectors $D_i(T_1) \dots D_i(T_k)$ of the training samples $T_1 \dots T_k$ with the highest similarity (the lowest Jenson-Shannon divergence JD) to the test vector $D_i(I)$.

Class Prediction and Confidence Estimation for each k-NN Classifier

Now, the class prediction L_i of each classifier C_i is obtained from the result list R_i using majority voting, e.g. L_i is set to the label $\{traversable, untraversable\}$ which occurs more frequently in R_i.

To estimate the quality of the class prediction, a confidence measure ρ_i is computed. The measure consists of two parts. The first component estimates the *a posteriori* probability $P(L_i|D(I))$ of the classifier C_i assigning the *correct* label L_i to the test region I, given its description vector $D(I)$. As proposed in [Arlandis 02], the probability is approximated by

$$\rho_{class} = P(L_i|D(I)) = \frac{\sum\limits_{label(T_i)=L_i} \frac{1}{JD(D_i(T_i), D_i(I))}}{\sum\limits_{i=1}^{k} \frac{1}{JD(D_i(T_i), D_i(I))}} \tag{6.12}$$

As can be seen, this measure takes both the class memberships of the nearest neighbors and their similarity to I into account. This is advantageous as it provides an informative confidence measure even in the case that only few training samples are available in the database and the value of k is small [Arlandis 02]. ρ_{class} is equal to 1 for unanimous decisions, for which all neighbors have the same class label. In case of conflicting labels, the measure can theoretically drop down to almost 0 (if 2 very similar samples are overruled by 3 very dissimilar samples), but never react the exact value.

The second part of the confidence measure addresses the problem that the estimate for ρ_{class} is always equal to 1 if the labels of all retrieved neighbors agree, regardless of their actual similarity with the test vector. This could lead to the classification of outliers with deceptively high confidence values, just because the closest neighbors happen to be of the same class. To avoid this, the probability that the test vector is adequately modeled by *any* training sample of the k-NN classifier must be included into the confidence value. A simple estimation of this is given by the denominator in equation 6.12. The direct sum of the similarity distances to the k nearest neighbors can be regarded as an indication of how well the test vector could be classified into any class at all.

In the developed algorithm, the mean value of the distances to all neighbors is used:

$$JD_{avg} = \frac{\sum_{i=1}^{k} JD(D_i(T_i), D_i(I))}{k} \tag{6.13}$$

The fact that JD_{avg} is not bound between 0 and 1 prevents it from being used as a confidence measure directly. Normalization to this range requires a mapping function and a distance threshold above which the classification is considered as too unreliable. Both can be learned or estimated statistically [Arlandis 02].

For simplicity, an empirically determined mapping function has been used in the developed approach. The mapping from average similarity to confidence is done using a quadratic function (see figure 6.31). The function drops to 0 at a threshold JD_{max}, which has been selected during experiments.

$$\rho_{dist} = \max\left(1.0 - \left(\frac{JD_{avg}}{JD_{max}}\right)^2, 0\right) \tag{6.14}$$

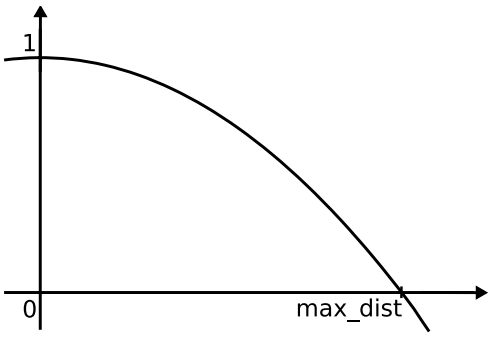


Figure 6.31: The mapping function from average similarity to confidence

The final confidence estimate ρ_i is computed as the product of the classification and distance based components:

$$\rho_i = \rho_{class}\rho_{dist} \tag{6.15}$$

Combining Class Votes for the Classifier Ensemble

Label assignment and confidence estimation for each classifier produces three class labels L_1, L_2, L_3 and three confidence scores ρ_1, ρ_2, ρ_3. These need to be combined into a single

class prediction L for the current region. Additionally, a confidence measure ρ is created for L, which will be used later as a quality criterion for filling the local traversability maps.

The fusion of outputs from multiple classifiers is not trivial and has received considerable attention, especially in the context of combining k-NN classifiers [Kuncheva 02]. From the available methods, a *weighted voting* scheme has been selected due to its straightforward applicability and good performance [Kittler 98]. For this, the confidence scores ρ_i are interpreted as voting weights and summed up according to the label L_i to form two voting scores $v(traversable)$ and $v(untraversable)$. The class label L is then set to the label that accumulated the higher score. The confidence ρ is determined by dividing the obtained voting score by the best possible voting score of 3. Again, ρ can range from 1 if all three classifiers are 100% confident and predict the same class, down to almost 0 if all three classifiers are extremely inconfident.

While theoretically simple and quite efficient in practice, the proposed fusion scheme deteriorates if the confidence estimates of the single classifiers do not agree with reality. In such a case, a single erroneous value for a ρ_i is able to dominate the entire voting – although the resulting confidence will be significantly below 1. A systematical exploration and evaluation of more sophisticated classifier fusion methods is therefore seen as a promising area for future research.

Example

Figure 6.32 shows an example of the results produced by the terrain classification.

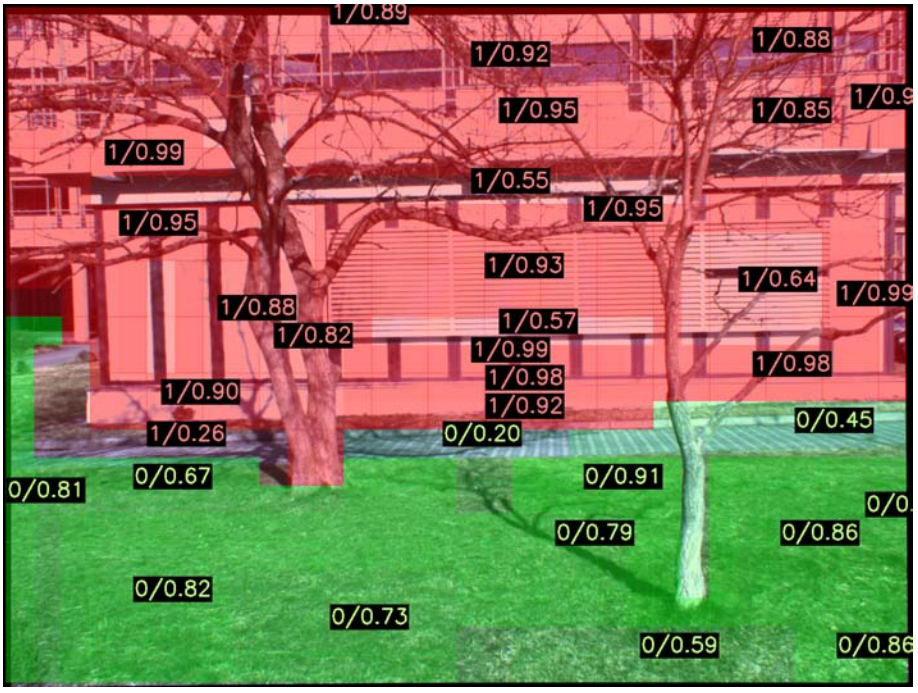


Figure 6.32: Image with assigned class labels

Each image region is labeled with the final class label L assigned to the region by the classifier ensemble (0 for traversable terrain, 1 for untraversable) and the confidence ρ associated to the classification. The region is also colored according to the final class

assignment, green for traversable, red for untraversable terrain. The opacity of the used color is determined by the confidence of the classification.

6.3 Integration

The last section explained how images are segmented and classified into traversable and non-traversable regions based on their appearance. Now, the gained information needs to be made available to the robot's navigation system so that the path planning and exploration mechanisms of the robot can benefit from it.

The least invasive way to include this new information into the already established system is to fill another local traversability map with it, similar to the approach taken for the shape-based terrain traversability estimation before (figure 6.33). With this, the edge cost prediction algorithm described in chapter 4 can draw upon a total of three different sensor based information sources: the map constructed from the local obstacle memory, the long-range map constructed by the shape-based terrain analysis and finally, the map filled by the appearance-based classification. However, neither the cost prediction nor the maximum confidence based map fusion algorithm (sect. 5.2.4) need to be modified internally to handle the new traversability map. This highlights the extensibility of the developed navigation system.

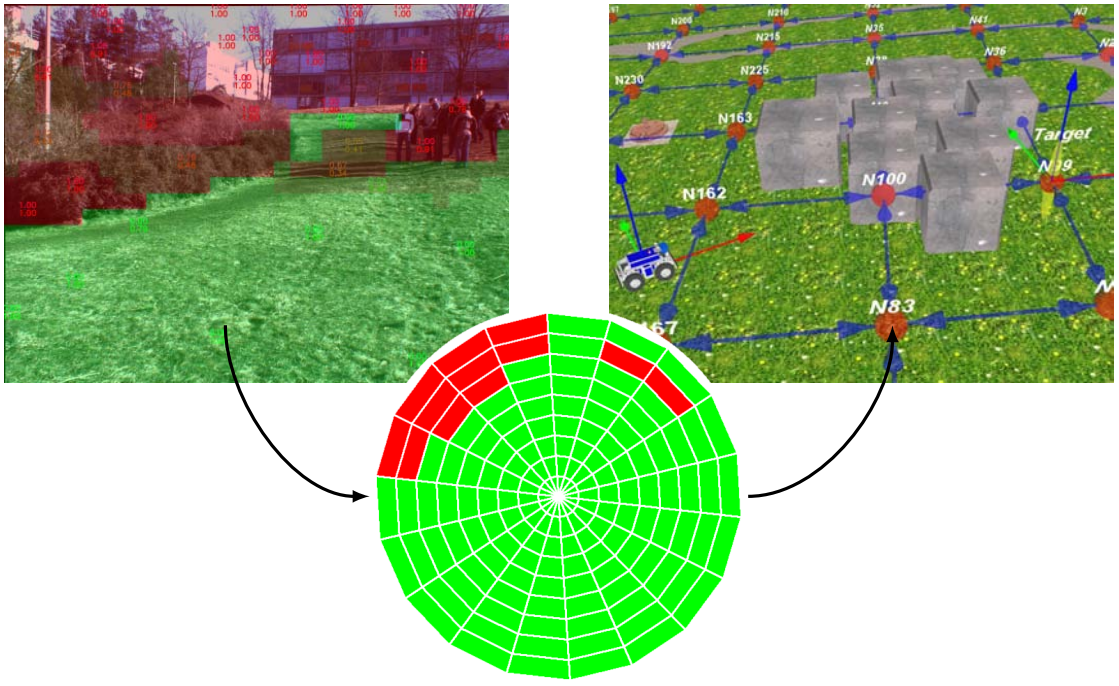


Figure 6.33: Integration of the appearance-based terrain traversability estimation
As before with the shape-based traversability estimation, the terrain information obtained by the appearance-based data interpretation is stored into a local traversability map. This map is attached to a topological node of the high-level navigation map.

In the following, it is assumed that the appearance-based traversability estimation has been triggered in the vicinity of a topological node and that this node provides a local traversability map in which the generated traversability data shall be stored.

In order to map the derived traversability information of the classified image regions onto the correct seclets of this map, three-dimensional information about the visible terrain is required. This 3D data can be extracted from the point cloud produced by the stereo reconstruction of the shape-based traversability estimator. Thus, each time an image is to be analyzed using the appearance-based approach, the stereo reconstruction step of the shape-based traversability analysis is triggered, too. Once the point cloud outlining the terrain surface is available, the points are passed to the appearance-based classifier.

Now, the obtained points are transformed from RCS over ECS into the NCS of the target node. Then, the source image of the classification is split into quadratic tiles with the minimum size used during image segmentation, producing a tile layout similar to that shown in figure 6.24b. This tiling is guaranteed to represent all the available traversability information without sampling losses.

For each image tile, a set of n valid 3D points is retrieved from the reconstructed point cloud. In order to do this efficiently, the points are temporarily indexed based on the tile they belong to (given their original 2D image coordinates) and sorted into an 2D (tile) array of point lists. During the experiments, a value of $n = 32$ yielded satisfactory results, but the choice of this parameter is not critical.

Then, the projection algorithm iterates over all points and all tiles. For each point, the corresponding seclet of the local map is determined as shown in section 4.1.3.1 resp. equation 4.8. If the confidence value θ_r of the seclet's risk cost modifier is lower than the final classifier confidence ρ of the current tile, the seclet's risk estimate and confidence is updated with $r = 0, \theta_r = \rho$ if the tile was labeled traversable, or with $(r, w) = (1, 1), (\theta_r, \theta_w) = (\rho, \rho)$ for untraversable terrain.

In effect, this procedure maps the information of the appearance-based terrain analysis onto the local traversability map for all image regions whose three dimensional position could be successfully determined. Note that the effort cost modifier of *traversable* terrain can not be estimated accurately by appearance alone. Therefore, the value of θ_w is not modified for these classifications. This missing piece of information can however be filled in very accurately by the shape-based traversability analysis described in the previous chapter.

Figure 6.34 shows the result of applying the presented mapping procedure on classified images from simulation runs. It can be observed that the prominent terrain features have been classified correctly into traversable and untraversable areas. The subsequent mapping has projected most of this information at the correct locations, although some errors occur around the borders of the stone obstacle group. These errors result from outliers in the point cloud and could be filtered easily, for example by modifying only seclets that are 'hit' by more than a minimal number of points.

6.4 Self-Supervised Traversability Learning

The performance of the terrain classification scheme presented in section 6.2.3 depends heavily on the quality of the training phase. In order to produce good results, examples for all of the terrain types that can be encountered during robot operation need to be trained. This is difficult to achieve for most applications with an useful spatial or temporal scope.

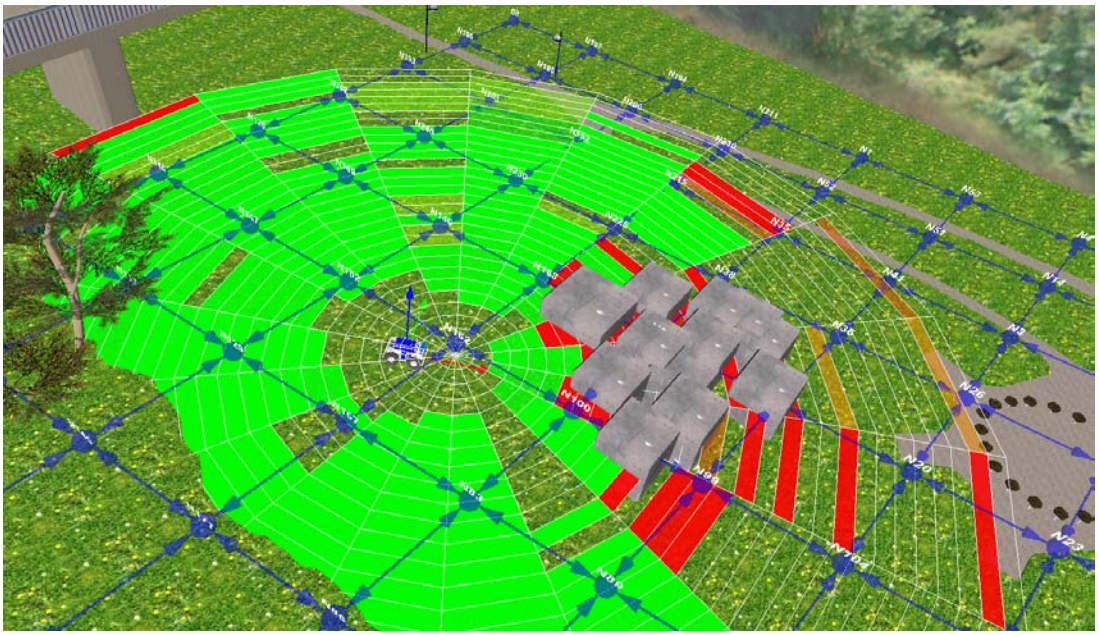


Figure 6.34: Traversability map built by appearance-based terrain analysis

In order to adapt to changing or a priori unknown terrain conditions, new terrain description vector / traversability label training pairs need to be generated during robot operation and inserted into the k-NN classifier databases. Since the robot is designed to operate autonomously, this incremental online training can not be supported by a human expert who labels the new prototypes correctly. Thus, the robot system needs to be augmented with a self-supervised learning strategy that is able to generate the needed description vector / terrain class label pairs autonomously.

In the scope of this thesis, a variant of the near-to-far learning strategy introduced in section 6.1 has been developed. In contrast to the established approaches proposed by other researchers, this learning technique does not extract traversability information from close-range tactile sensors or otherwise process raw sensor information. Instead, it observes the reactions of the behavior-based piloting layer with respect to the crossed terrain. Learning is thus based on the same source of information as the edge cost learning technique proposed in section 3.5.5, which learns and adjusts the risk and effort costs of a topological edge by observing the pilot.

Figure 6.35 shows the design of the traversability learning scheme. As can be seen, the same two sets of behaviors A, B are singled out from the behavior-based piloting subsystem as before for the edge cost learning (see figure 3.19). To recapitulate their properties, each behavior in the two sets generate the situation assessment signal $r(t)$ and exports the spatial location $\vec{p}(t)$ of the stimulus responsible for it. The Motor Control Behaviors $B_i \in B$ exhibit a non-zero activity $r^{B_i}(t)$ whenever current is fed into the actors, i.e. the robot moves. The stimulus location $\vec{p}^{B_i}(t)$ is the robot's position itself. The Obstacle Avoidance Behaviors in A steer the robot safely around obstacles and react to stimuli (obstacles) originating from a spatial location outside the robot, encoded in $\vec{p}^{A_i}(t)$. The situation assessment signal $r^{A_i}(t)$ is greater than 0 if behavior A_i sees a necessity to influence the robot trajectory in order to avoid an obstacle.

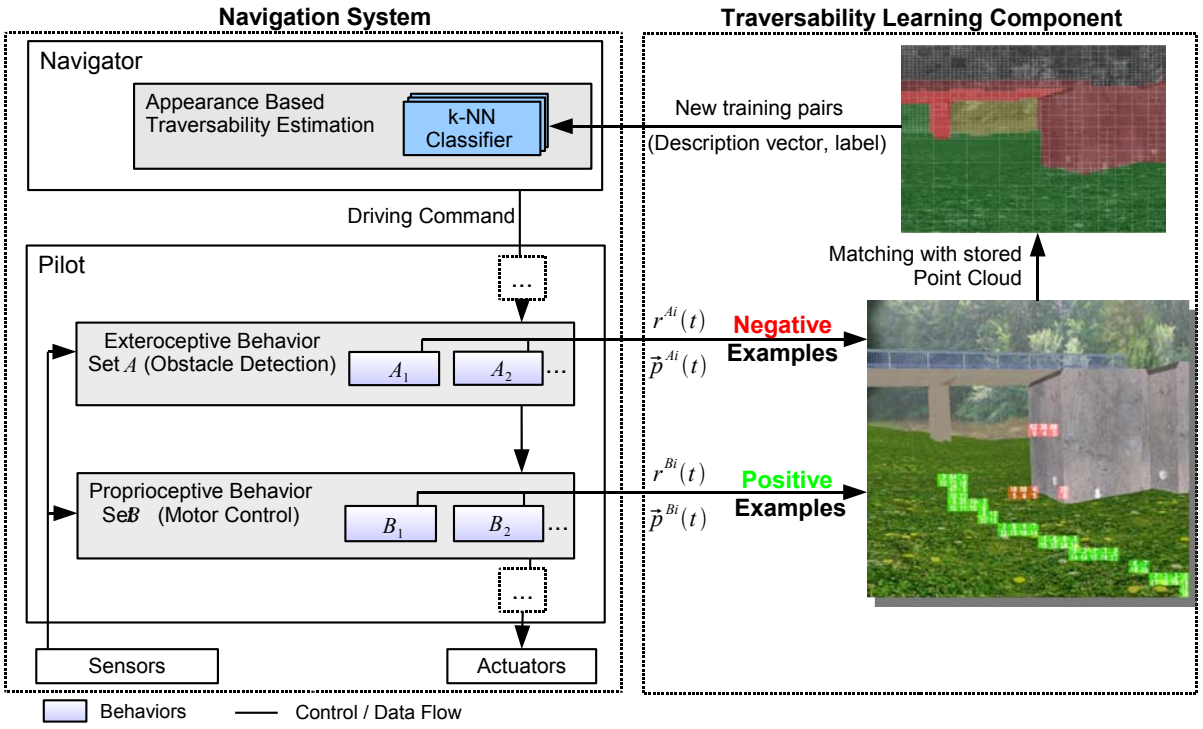


Figure 6.35: Learning scheme

The traversability learning process itself is divided into two stages: data recording and the actual generation of new training pairs.

Data Recording

Data recording is initiated whenever the robot performs an appearance-based traversability estimation for a topological node and is about to traverse a topological edge afterwards. In this situation, the point cloud generated during the creation of the local traversability map is retrieved (see section 6.3). Each point is projected into ECS and annotated with the description vector that was extracted for the corresponding image region. The annotated cloud is then saved for later use.

During robot motion, the traversability learning process observes the assessment signals $r(t)$ and stimulus locations $\vec{p}(t)$ of the pilot behaviors in sets A and B. The stimulus locations are mapped into the ECS coordinate system using the robot's pose estimate and stored as well.

Generation of Training Pairs

After the robot finishes the edge traversal movement, the stored behavior signals are considered in turn. Each time a behavior from set A exhibits a situation assessment signal $\vec{p}^{A_i}(t)$ significantly above 0, the corresponding stimulus location $\vec{p}^{A_i}(t)$ (which has been identified as an obstacle by the pilot) is matched with the nearest point of in the annotated point cloud. Then, the description vector associated with this point is labeled 'untraversable' and put into the k-NN classifier database.

The same procedure is performed for behaviors from set B. In this case however, the stimulus locations $\vec{p}^{B_i}(t)$ provide positive feedback, as the trajectory of the robot obviously crosses traversable terrain. Thus, the description vectors corresponding to set B's locations are labeled 'traversable'.

Concerning the implementation of the self-supervised learning technique, the logging mechanisms originally developed for the edge cost learning can be reused without difficulties. However, the repeated searches for the point in the annotated point cloud that lies closest to a given stimulus location are computationally taxing. In order to reduce the runtime requirements for this and strike a viable compromise between efficiency and finding correct matches, only *approximate* nearest neighbor searches are performed. These searches do only guarantee to find the closest match within a given error bound, but can be several orders of magnitudes faster that exact searches due to the use of kd-tree search structures. In this thesis, the publicly available `ANN` library[2] was used.

6.5 Experiments and Results

The algorithm for appearance-based terrain traversability estimation was subjected to a series of experiments in order to determine its accuracy and runtime performance. First, the obtained results concerning the classification accuracy will be presented. Then, the measured runtime performance will be discussed.

6.5.1 Classification Accuracy

In order to measure the accuracy of the proposed method, the k-NN classifiers were trained using a set of 16 hand-labeled training images taken during a typical use of the robot system at the testing site. Figure 6.36 shows 6 of these images. As before, regions that were labeled as untraversable by the human expert are marked with a red overlay, while traversable regions are marked green. In total, 772 valid samples were extracted from the training set and added to the k-NN classifier databases. The samples contained 11047 valid image tiles with size 32x32. 62% of these tiles were marked untraversable, 38% were traversable.

Qualitative Evaluation

To measure the overall performance of the developed visual terrain traversability estimation method, the algorithm was applied to a test set of 7 images not included in the original training set. During the tests, a minimal image region size of 32x32 pixels was used. The other parameters were kept equal to the parameter set used in section 6.2.2.3.

For each test image, a visualization image has been created to present the classification results in a lucid fashion. These images are shown in figures 6.37 and 6.38. Here, all major (>7 tiles) image regions are annotated with their classification labels and confidences. Each image region contains four pairs of scores. As before with the example shown in section 6.2.3.4, the uppermost pair shows the final class label L assigned to the region by the classifier ensemble (0 for traversable terrain, 1 for untraversable) and the confidence ρ. Again, the region is colorized to indicate the final class assignment (the color intensity

[2] http://www.cs.umd.edu/~mount/ANN/

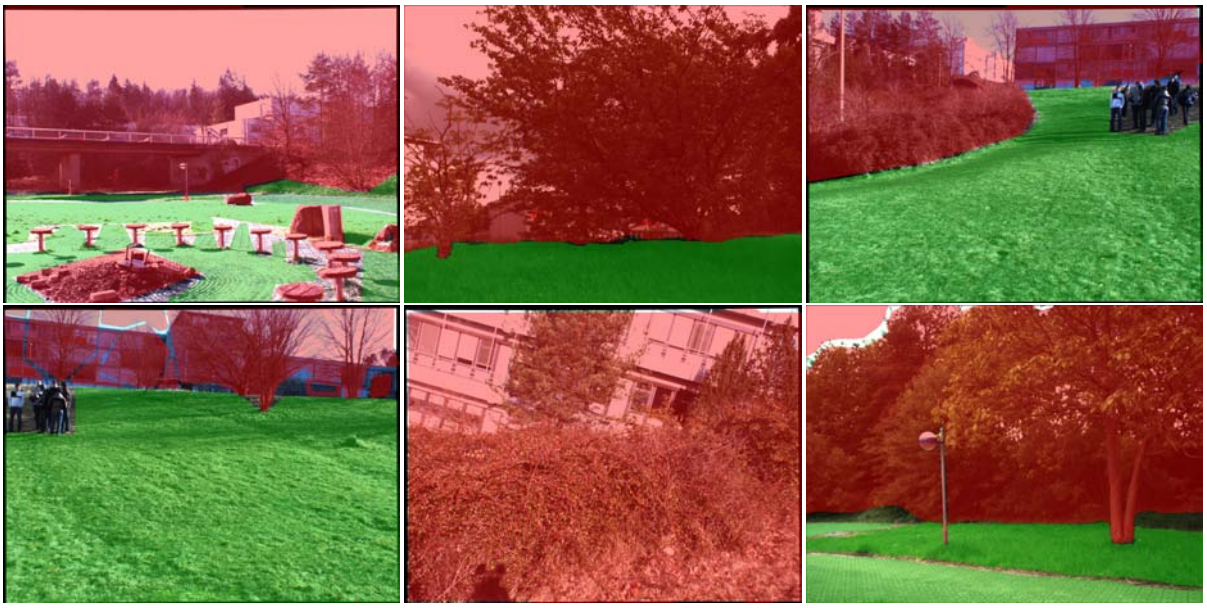


Figure 6.36: Some training images

depends upon ρ). The three pairs below the top row contain the votes and the confidences of the three separate classifiers, in the order textural structure (LBP) classifier, brightness variation (VAR) classifier and color classifier.

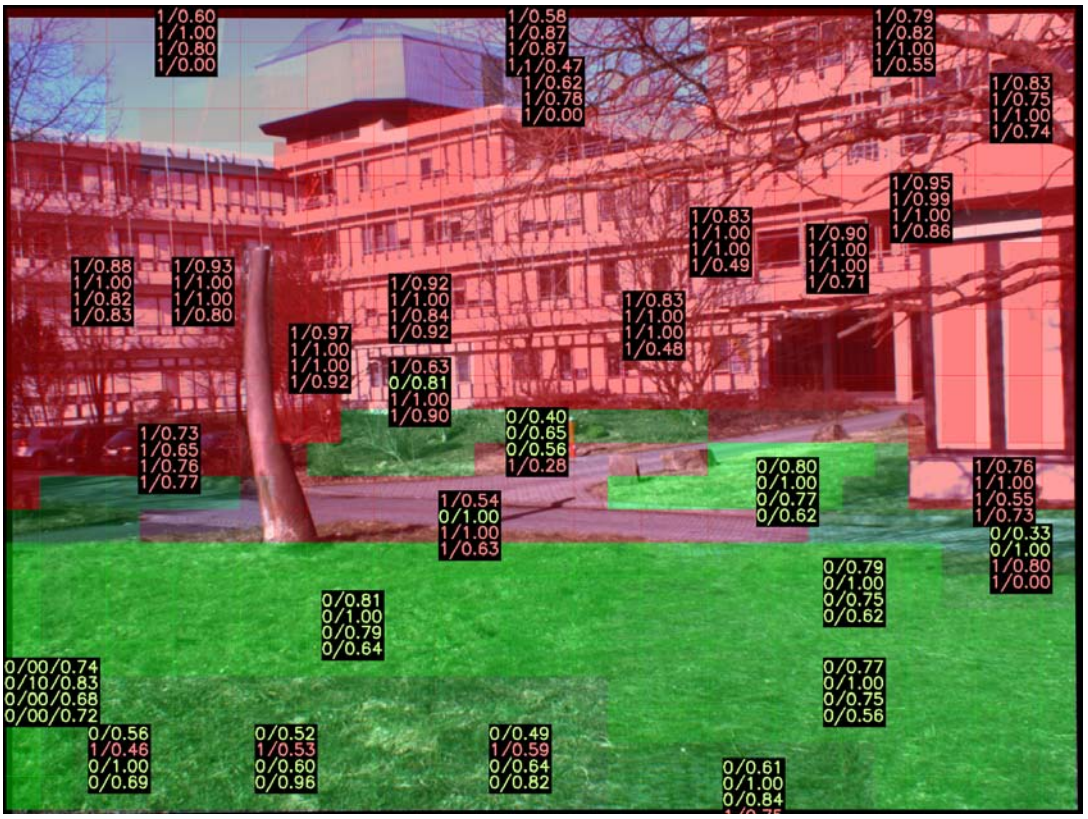
Figure 6.38 shows more classification results of the test set. In order to ensure legibility, only the final classification label L and confidence ρ is annotated for these images.

Quantitative Evaluation

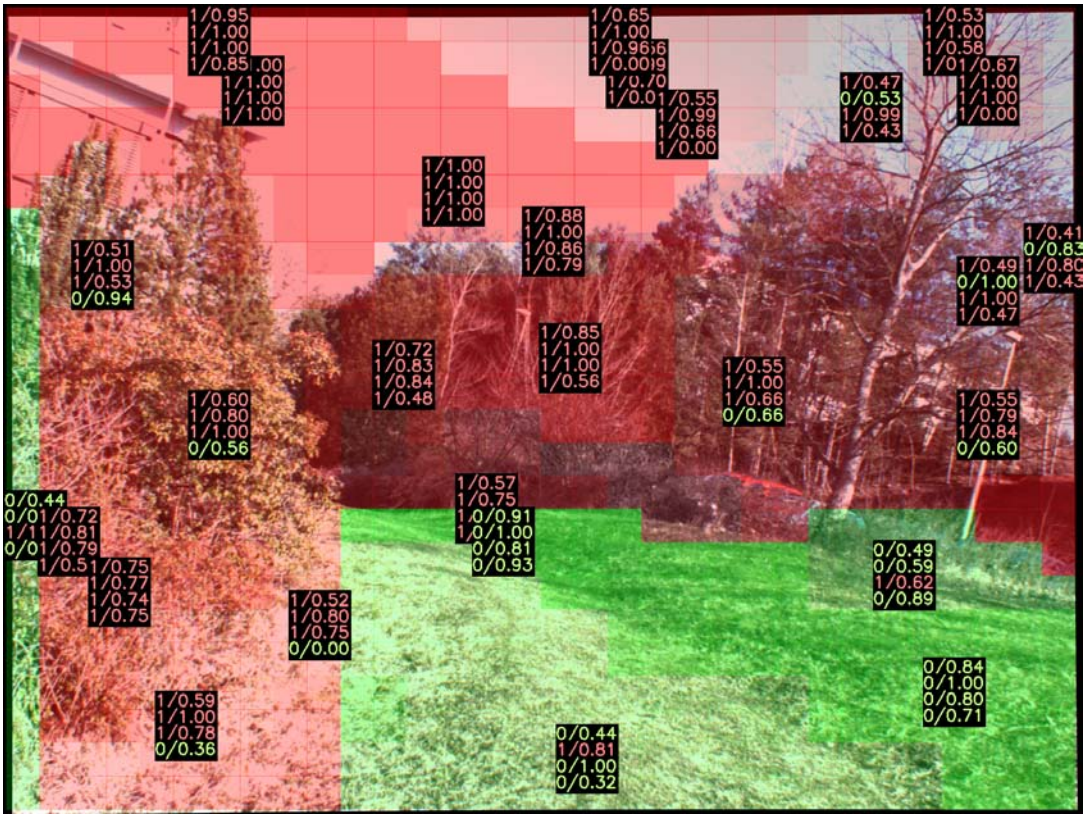
For a quantitative evaluation of the classifier accuracy, the test set was labeled by hand to generate ground truth data. This data was then matched with the classified images shown in figures 6.37 and 6.38. To compare the automatically segmented images with the ground truth images, each image was split into 32x32 pixel tiles and the comparison between assigned and correct label was done tile by tile. Tiles containing sky were excluded from the evaluation in order to avoid skewing the results by the substantial amount of sky tiles in the test set. Although these tiles are always classified correctly due to the highly distinct signature of overexposed image regions, they do not provide any useful traversability information. Including these tiles would therefore produce unrealistically good accuracy measures without practical relevance.

In total, 4619 image tiles were compared. 3308 (72%) tiles had been labeled untraversable by the human, 1311 (28%) had been labeled traversable. Table 6.3 shows the obtained overall classification accuracy for the classifier ensemble responsible for the final classification, as well as the scores that would have been obtained by each single classifier alone.

As can be seen, the combined classifier is able to distinguish traversable from untraversable terrain with an accuracy of almost 90%. Taken on it's own, this is an excellent result. Of course, the training and testing sets have been recorded in the same area and a part of the training images has been taken in bright daylight, similar to the testing set images. Thus, the environmental conditions of the test set have been well trained. Nevertheless, the



(a)



(b)

Figure 6.37: Classification results on test set

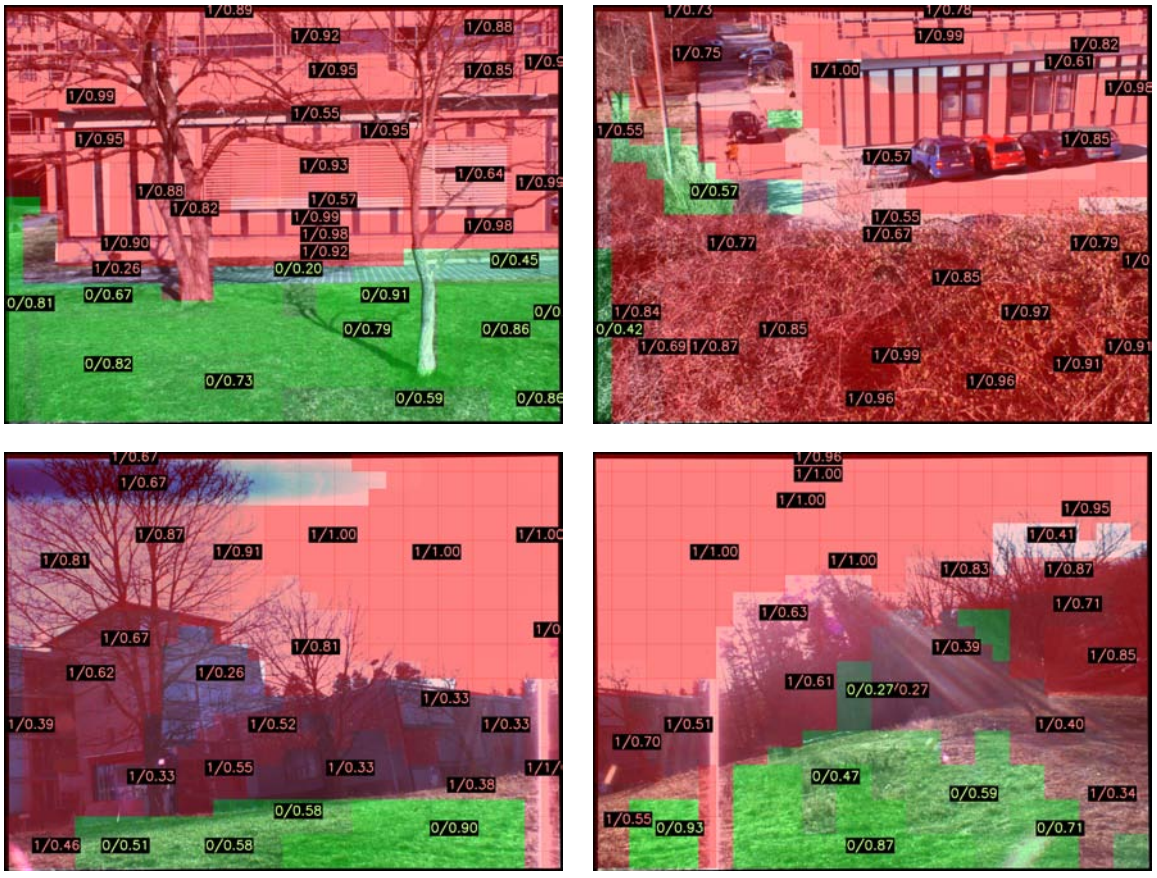


Figure 6.38: More classification results on test set

All	Texture (LBP)	Contrast (VAR)	Color (HSV)
89.3 %	81.4 %	86.5 %	88.0 %

Table 6.3: Overall classification accuracy

test set contains images with new (untrained) object types (bushes, cars, other buildings), difficult illumination conditions and rotated perspectives. With these effects in mind, the achieved accuracy is definitely noteworthy and shows how much the appearance-based traversability estimation method can contribute to the robot navigation as a whole.

Table 6.3 also shows that the classifier ensemble has the lowest overall classification error. This proves that the combination of different feature extraction operators really provides some merit. However, the benefit of using the classifier ensemble instead of the single classifiers does appears to be rather low in this experiment, as each of the separate classifiers performs almost as well as the three classifiers together. In order to further explore this curious behavior and the reasons for this and the remaining classification errors, the classifier performance is examined more thoroughly in the following.

Table 6.4 lists the percentages of correct and incorrect classifications, separated according to the true label of the image tiles. This statistics allows to look more closely at the distribution of correct and incorrect classifications with respect to the two true traversability classes.

Classifier	Correct Classification as Traversable	Correct Classification as Untraversable	Incorrect Classification as Traversable	Incorrect Classification as Untraversable
All	**79.4 %**	**93.3 %**	**6.7 %**	**20.6 %**
Texture (LBP)	77.7 %	82.8 %	17.2 %	22.3 %
Contrast (VAR)	69.4 %	93.3 %	6.6 %	30.6 %
Color (HSV)	77.7 %	92.4 %	7.6 %	22.9 %

Table 6.4: Classification confusion statistics

Given these error figures, it is apparently more difficult to classify traversable areas correctly than untraversable ones. This effect can be observed for all classifiers as well as the ensemble. One possible explanation of this could be the lower proportion of traversable training samples in the classifier databases. k-NN classifiers are well known to become biased in the presence of unbalanced training sets, and the weakness in classifying traversable terrain correctly might be a result of this property. Another possible cause could be the high similarity of streets and the sky. Both are uniformly textured, exhibit low contrast and are rather achromatic. However, their traversability labels in training are different. But since sky samples are more common than street tiles, the traversable street regions are likely to be assigned the untraversable label from the sky samples. This illustrates the importances of selecting an appropriate training set. In this case, an easy solution to this problem would be to leave the sky parts unlabeled, as sky tiles are not relevant for traversability estimation anyway.

Classifier Liability

It is also interesting to analyze which of the classifiers are involved into an ultimately wrong classification result, i.e. to evaluate the classifiers' liability for classification failures. This was done by counting the number of times that each classifier's own result was incorrect, given that the final classification was incorrect, too. Table 6.5 shows the fraction of these counts with respect to the total number of incorrect classifications. These values sum up to more than 100 % since more than one classifier can (and mostly will) be wrong if the overall classification result is incorrect.

Texture (LBP)	Contrast (VAR)	Color (HSV)
64.4 %	81.6 %	54.0 %

Table 6.5: Classifier liability

It appears that the contrast operator had the greatest detrimental effect on the overall classification. The contrast operator was at least partially involved in the production of incorrect results in 81.6% of the cases. Considering table 6.4, it appears likely that many of these cases resulted in a traversable region begin classified as untraversable. This and the fact that the street and sky textures stand out especially with respect to their contrast again points in the direction of this already identified problem. In total, the liability analysis highlights the different characteristics of the three feature extractors, which each have distinct advantages and drawbacks.

Time Series Analysis

It may come as a surprise that the classifier based on the color features alone performed almost as well as the classifier ensemble and outperformed the other, single classifiers in the test set experiment. However, the test images were relatively homogeneous in terms of illumination, making discrimination based on color information easy. One could thus hypothesize that the strengths of the other operators are more observable when stronger illumination changes or other image variations occur.

This hypothesis was tested using a time series analysis using images that have been taken over a longer period of time. Figure 6.39 shows three such images, all from the same camera position showing the same scene. The recording times of the images were 10.21 am, 14.08 pm and 17.25 pm. Obviously, illumination and contrast, and thereby colors changed significantly during that time.

Figure 6.39: Some images used for the time series experiment
From left to right, images were recorded at morning, afternoon and early evening.

Leave-one-out evaluation of the time series test set resulted in the total classification accuracies for the single classifiers listed in table 6.6.

All	Texture (LBP)	Contrast (VAR)	Color (HSV)
94.7 %	96.4 %	82.9 %	85.5 %

Table 6.6: Classification accuracy for time series experiment

In the time series experiment, the color classifier performed much worse than before. The same holds for the contrast classifier, whereas the textural structure classifier performed very well. Due to the nature of the extracted features, the contrast operator is strongly affected by the changes in illumination brightness, while the color operator is not invariant to changes in illumination color. Both do not influence the LBP operator, and since the texture pattern of the image components already have been seen during training (and have not changed much), good classification is possible based on the textural image structure.

These results support the initial hypothesis about the good performance of the color classifier on the test set. It also reveals that the real benefit of the combined classification technique is not so much an improved classification accuracy in comparison to the single classifiers, but an increased stability against environmental changes. In combination with the first experiment, the consistently good performance of the classifier ensemble shows that it is able to rely on the color and contrast features for ‘easy’ images which have

high similarity with the training set, whereas it can exploit the strengths of the textural features for classification tasks containing larger illumination changes. Using the classifier ensemble, the appearance-based terrain classification can successfully draw upon the complementary discrimination abilities of the different features.

Influence of Segmentation and Tiling

The use of unsupervised segmentation as a preparational step for terrain classification has both benefits and drawbacks. On the positive side, fewer regions need to be classified after segmentation and the description vectors of larger regions are statistically more stable. On the negative side, the segmentation is not always perfect. The remaining regions can be too large and group terrain with different traversability labels. Since one region can only be assigned one class label, some errors are bound to be induced by such an inappropriate segmentation.

Although the exact influence of the segmentation stage on the classifier performance is difficult to quantify accurately, it can be estimated by classifying the same images that have been used for classifier training. During training, the overly ambiguous regions have been removed, but such a filtering step based on ground truth data is of course not possible for real classification tasks. Thus, the errors introduced by the segmentation become visible. Table 6.7 summarizes the obtained results.

All	Texture (LBP)	Contrast (VAR)	Color (HSV)
94 %	90.2 %	86.8 %	90.1 %

Table 6.7: Overall classification accuracy on *training set*

As can be seen, the error rates are much lower (almost halved) than with the test set. However, the remaining classification error is still significant. This indicates that the unsupervised segmentation and minimum tiling size of the source images introduces a non negligible amount of error into the total classifier accuracy.

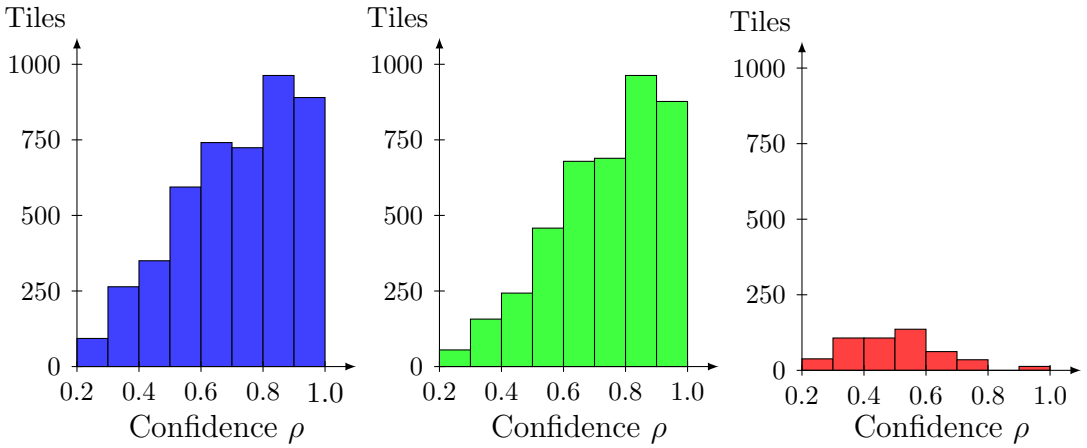
Confidence Distribution

Figure 6.40 shows the confidence distribution of the classification results. A histogram is build over all confidence values ρ that were assigned to classified tiles.

Overall, the samples are likely to be classified with a rather high confidence, over 55 % of all samples are classified with $\rho \geq 0.7$. However, the most interesting part of the confidence distributions are the differences between the confidences assigned for classifications that were correct (figure 6.40b) and those that were incorrect (figure 6.40c).

The correct classifications are made with an average confidence of 0.69 and more than 77% of the classified tiles exceed a confidence score of 0.6. In contrast to this, the incorrectly labeled samples attain a lower average score of 0.45, and less than 22% of the classified tiles attain a confidence of 0.6 or more.

Thus, the assigned confidence value does indeed contain valuable information. It can easily be utilized to filter incorrect classifications, for example by only integrating traversal cost modifiers into the final local traversability map (as shown in section 6.3) which exceed a



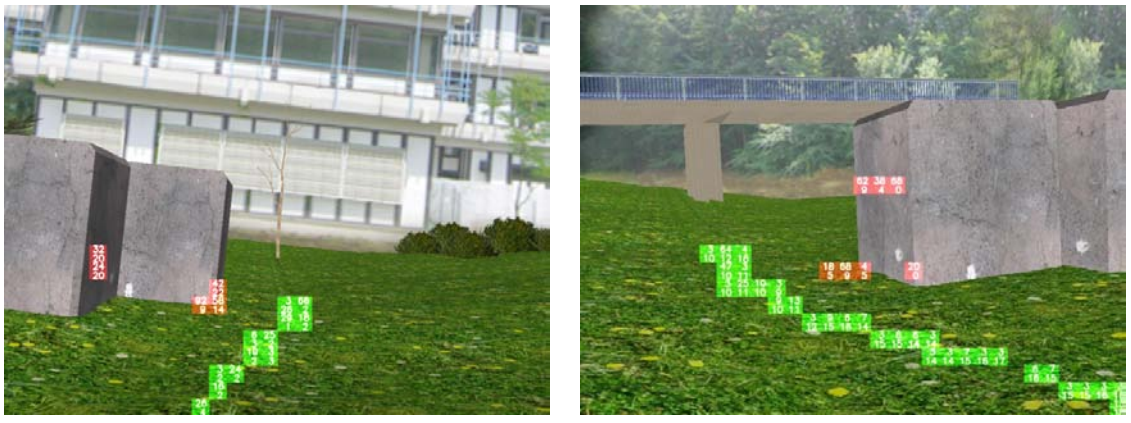
(a) All classifications (b) Correct classifications (c) Incorrect classifications

Figure 6.40: Classification confidence

given threshold for ρ. However, even without adding such a drastic filtering step, incorrect classification results with low ρ have less impact on the path cost estimation than correct ones with high ρ anyway. After all, ρ is used to set the seclet confidence value, which in turn determines the actual seclet cost through bilinear interpolation. This interpolation ensures that seclets with low confidence are assigned costs close to the default value for unknown terrain, regardless of the actual (unreliable) cost modifier.

6.5.2 Online Learning

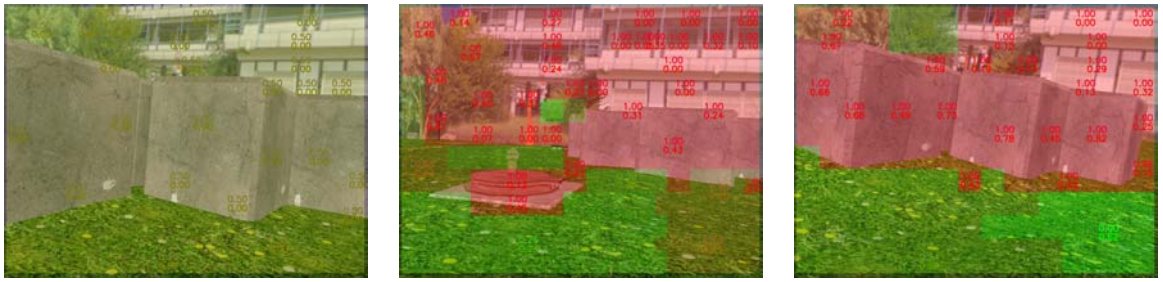
The performance of the online learning scheme has initially been evaluated in the SimVis3D simulation [Braun 07] to ensure repeatable conditions. Figure 6.41 shows the training pairs that have been successfully added to the k-NN database after executing a driving command that led the robot across a grassy surface besides some stone cubes inserted to serve as a distinct obstacle. Green squares indicate patches that have been recognized as traversable, while red indicates impassable obstacles.



(a) (b)

Figure 6.41: Training samples added after execution of a driving command

Learned terrain traversability classifications are shown in figure 6.42. For this experiment, the k-NN database has been wiped at startup, thus the first terrain classification (figure 6.42a) marks all regions with a 'neutral' traversability value of 0.5. After executing some driving commands, the classifier database starts to become populated and the classifications become more sensible (Figures 6.42b, 6.42c).



(a) No prior knowledge (b) 200 samples added (c) 400 samples added

Figure 6.42: Online learning in simulation

Finally, the learning scheme was applied to a real-world scenario using the real robot. Again, the classifier database was cleared initially. Then, the navigator was commanded to perform a series of driving commands while observing the obstacle avoidance behaviors and autonomously collecting terrain description / traversability pairs. After adding 33 samples to the database, a previously unknown image has been classified according to figure 6.43. As can be seen, the estimated traversability is approximately correct for the larger part of the image, although a significant number of regions is misclassified.

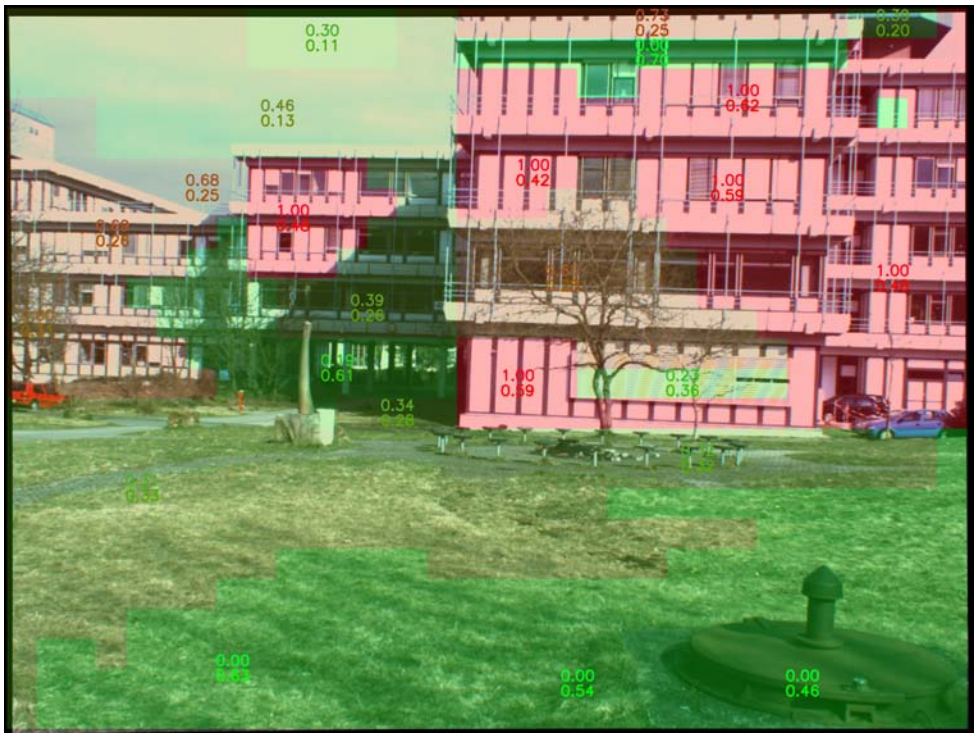


Figure 6.43: Autonomously learned terrain traversability estimation

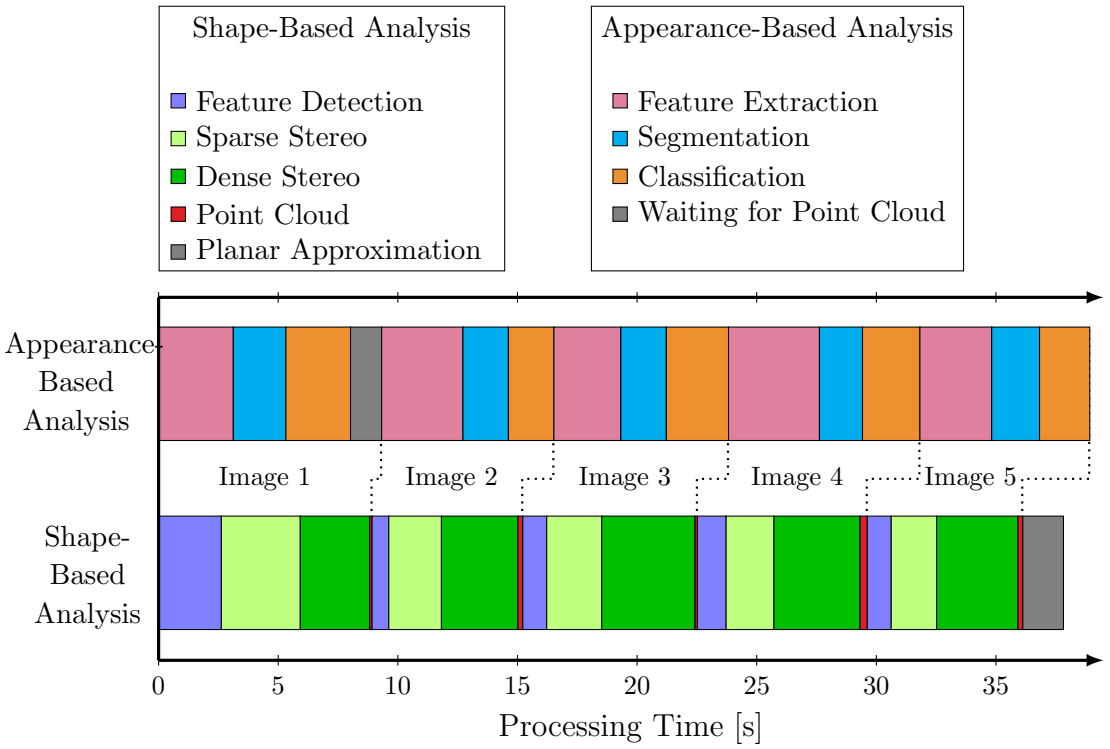


Figure 6.44: Runtimes for shape- and appearance-based terrain analysis

6.5.3 Runtime Performance

To evaluate the runtime performance of the terrain traversability system, the processing times needed by its different steps have been analyzed. This was done with a series of five images taken from simulation. As stated in section 6.3, the appearance-based evaluation requires the point cloud from shape-based analysis in order to map from the classified images into the 3D world coordinate system. Therefore, both appearance-based terrain evaluation and shape-based evaluation are normally activated simultaneously by the navigator and process the captured images from the stereo system in parallel on a dual-core CPU. Due to the needed synchronization between the two methods, the timing of both systems with respect to each other is important and has been recorded during the runtime measurements. The resulting graph is shown in figure 6.44.

It is observable that the single most time consuming part of the appearance-based algorithm is feature extraction (41 % of total processing time). The extraction of the texture features requires by far the most of this time (more than 90 %), whereas color and contrast features can be retrieved quickly. The second and third most time consuming steps are classification (27 % of the total) and image segmentation (25 % of the total).

The high time consumption of the classification step stems mainly from the nearest neighbor searches done by the k-NN classifiers. Here, for each unknown sample, the JD divergence to all d samples in the database has to be computed. The required time thus rises linearly with the size of the database. To prevent deteriorating the system performance in conjunction with online learning, one could try to ensure small database sizes for example by pruning samples that are not used for classification for a long time. Another way to speed up the k-NN classification would be to use a more sophisticated neighbor search

(e. g. using kd-trees, or hashing). However, this would require the use of a real distance metric instead of the Jensen-Shannon pseudo metric used at the moment (the kd-tree algorithms build upon the triangle inequality relation, which is not fulfilled for JD). Thus, such optimizations are deemed future work.

For the shape-based analysis, the three most time consuming parts are the generation of the dense disparity map, the matching of the sparse features, and the detection of well matchable features. As can be seen in the figure, the appearance-based operator must only wait for the point cloud after the first image; here some initializations prolong the feature detection and matching steps. After the first image, it does not need to wait any more for a new image from the image grabber, or the reconstructed point cloud from shape-based analysis.

6.6 GPU Based Runtime Optimization

The experiments revealed that the presented terrain classification approach performs well in practice and supplies useful information to the navigation system. However, a whole range of aspects can be further improved. One of the more severe problems which limit the usability of the entire sensor based terrain analysis method is the large computational cost incurred by the various processing steps. While the shape-based analysis suffers from the high computational effort of the sparse and dense stereo reconstruction methods, the appearance-based technique is slowed down most by the extraction of the image features, namely the multi-scale local binary pattern texture features.

Thus, a way to improve the runtime characteristics of the feature extraction step has been sought. In the past, specialized hardware components have been proposed as a solution for real-time texture analysis [Lahdenoja 06]. Unfortunately, these devices are often costly, not widely available and lack flexibility. In contrast to this, today's cheap and freely available customer-grade graphics cards (GPUs) have become programmable enough so that they can be used to efficiently solve such problems.

Therefore, a reformulation of the uniform, multi-scale Local Binary Pattern Operator introduced in section 6.2.1.1 has been developed which can be efficiently executed on a GPU. The new algorithm is integrated into a pipeline framework that handles the low-level data flow between different GPU program elements. This concept can be transferred easily to other GPU-based algorithms. The developed algorithm and the obtained results have been presented to the public in a joint publication with G. Zolynski [Zolynski 08].

6.6.1 Developed Approach

The GPU implementation focuses on the most time consuming, first step of the feature vector generation, the transformation of the input image into the LBP feature image (see figure 6.11).

The concrete algorithm is implemented as a sequence of GPU fragment shader programs using NVIDIA's Cg programming language. Each shader program takes an image and transforms it into an output image, which serves as basis for the next program. Since the fragment shaders are capable of processing four floating point 'channels' in parallel (usually calculating RGBA color values), the sampling points of the multi-scale LBP

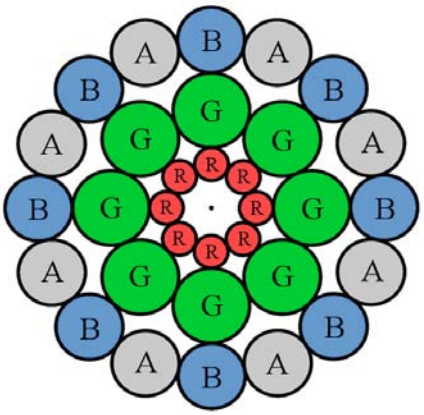


Figure 6.45: RGBA channel allocation for the multiscale LBP operator

operator $\text{LBP}_{8+8+16,1.3}$ shown in figure 6.20 are split into four LBP patterns with 8 samples, distributed across the four scales as indicated by figure 6.45.

The shader program sequence executed in order to compute the LBP image is depicted in figure 6.46.

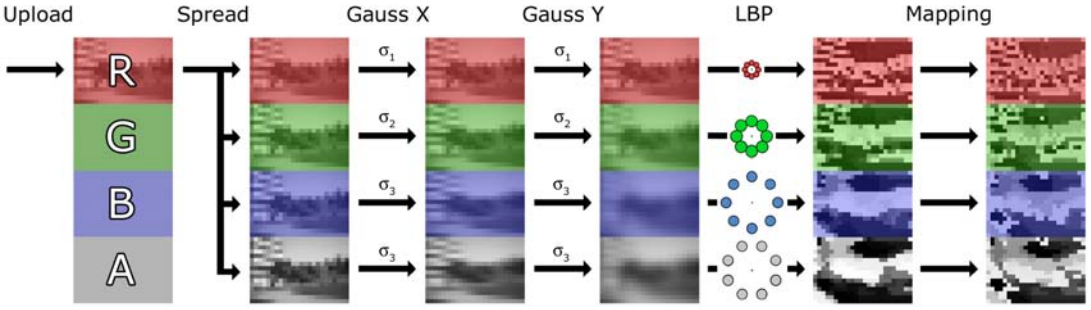


Figure 6.46: Sequence of GPU shader programs for LBP image computation

After uploading the input image, it is first spread (copied) over all color channels. Afterwards, each channel is blurred in two steps using separated Gaussian filter kernels. The kernel sigma is chosen according to the sampling radius of the corresponding LBP ring and computed according to the formulas presented in section 6.2.1.1. Note that since the two largest LBP rings (stored in the blue and alpha channels) have equal radii, their sigma values are also equal.

The core step of the algorithm is the computation of the LBP values from the blurred images. For this, a branch-free and vectorized reformulation of the standard algorithm has been developed (figure 6.47).

An important feature of this vectorized GPU algorithm is the explicitly performed 4-fold predication using the variable `predicate`. This removes the computationally expensive branching required in the naive implementation and allows the shader program to run at full speed. As a further advantage, whenever the take4Samples method needs to sample points that are not exactly aligned with actual pixels, the automatic interpolation offered by the graphics hardware can be exploited at no extra speed penalty. The final step of the processing pipeline consists of the value mapping required for the uniform LBP extension and is realized as a simple texture lookup.

6.6.2 Evaluation

Experiments were carried out using GeForce 7600 GT, 8600 GT and 8800 GTS type graphics cards measuring the speed and the accuracy of the implemented operator. As evaluation basis, 33 test images published by the Signal & Image Processing Institute (University of South California)[3] with dimensions 512x512 were used for the accuracy tests. The speed was evaluated on 43 typical input images with dimensions of 1024x1024 pixels.

Accuracy

The first experiment compared the accuracy of the proposed GPU operator with a reference implementation based upon a C++ port of the code published by the original LBP authors. Since the reference implementation is based on 32 bit integer computation and the GPU implementation uses 16-bit floating point accuracy ('half' values in Cg) to achieve faster processing speeds and enable hardware-accelerated linear interpolation, slight amounts of incorrectly computed LBP values had to be expected.

As can be seen in figure 6.48, between 0.2% and 1% of the computed values differed from the C++ implementation, depending on the input image. Also, one can observe an overall accuracy improvement between NVIDIA's GeForce 7 and GeForce 8 chipset families, as the GeForce 8 cards typically exhibit about 20-40% fewer errors than the GeForce 7 cards. Overall, the comparisons showed that on average, less than 0.5% of the computed values are inaccurate, which was seen as acceptable.

Speed

The foremost aim of using the GPU was to achieve real time capability on available and comparably cheap hardware. Being massively parallel by design, the Local Binary Pattern operator is very well suited to be implemented on the likewise parallel architecture of a GPU processor. Since every LBP pattern is independent, any amount of patterns can be computed simultaneously. The limit to the number of concurrent executions is only set

[3] http://sipi.usc.edu/database/database.cgi?volume=textures

```
int pattern = 0;
int centerVal= takeSample(center);

for (int i=0; i<numSamples; i++)
{
  int neighborValue =
          takeSample(neighbor[i]);
  if (neighborValue > centerVal)
  {
     setBit(pattern, i, 1);
  }
}
```

(a) Naive CPU Implementation

```
int4 patterns = int4(0, 0, 0, 0);
int4 centerVals = take4Samples(center);
int mult = 1;
for (int i=0; i<numSamples; i++)
{
  int4 neighborVals =
          take4Samples(neighbor[i]);
  bool4 predicate =
          (neighborVals>centerVals);
  patterns += (int4(predicate) * mult);
  mult *= 2;
}
```

(b) Vectorized GPU Code

Figure 6.47: Pseudo-code for vectorized LBP code

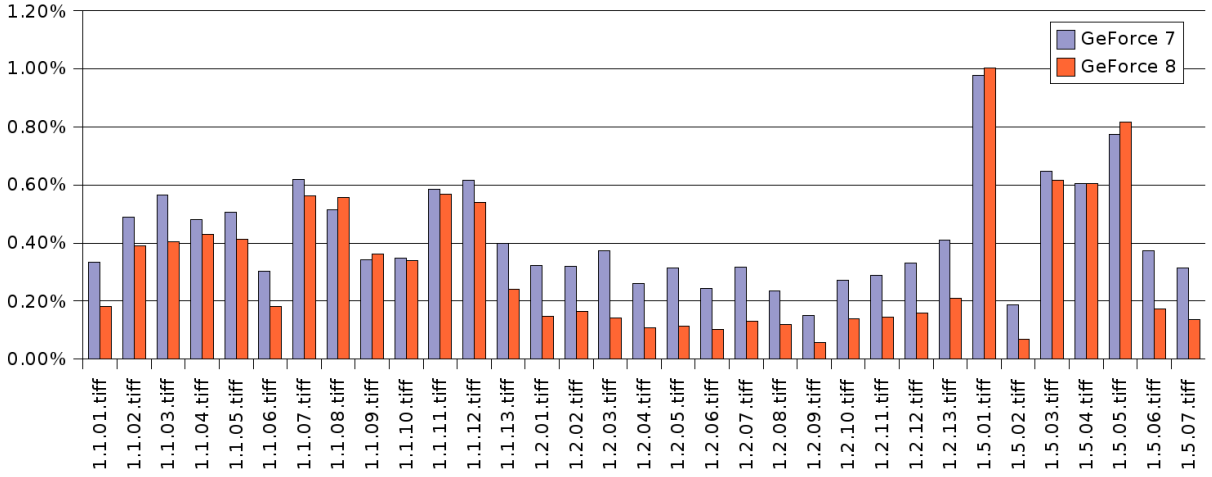


Figure 6.48: Percentage of LBP values diverging from reference
The LBP values of the GPU implementation and reference implementation were compared for the images contained in the South California SIPI Dataset

by the amount of shading pipelines within a graphics processor, which is as much as 96 for the GeForce 8800 GTS.

Run time was measured using images with dimensions of 1024x1024 pixels. For each image, the LBP feature image was created 10,000 times and the measurements were averaged.

CPU / GPU	Total Processing Time	Upload	Computation	Download
Core2Quad 2.4GHz[1]	1150 ms	-	1150 ms	-
GeForce 7600 GT	83 ms	17 ms	20 ms	46 ms
GeForce 8600 GT	72 ms	17 ms	11 ms	44 ms
GeForce 8800 GTS	65 ms	17 ms	5 ms	43 ms

Table 6.8: Speed comparison CPU / GPUs
[1] Only 1 core was used during the experiment.

Table 6.8 shows the execution speed of the three graphics adapters compared with the CPU reference implementation. Even the slowest GeForce 7600 GT adapter beats the CPU by a factor of almost 14. On the other hand, differences between cards are small, since most of the time is spent in the data up and download phase. The relative time differences are presented visually in figure 6.49.

Note that the CPU implementation was running on a single core only. Assuming that the bottleneck of CPU computation is the processor and not the memory (which appears unlikely, as the core did not reach maximum load during the tests), the CPU execution time could be lowered to a quarter, which is still about 3.5 to 4.0 times slower than the GPU implementation. However, this would block the entire CPU, preventing any other (navigation-related) computations.

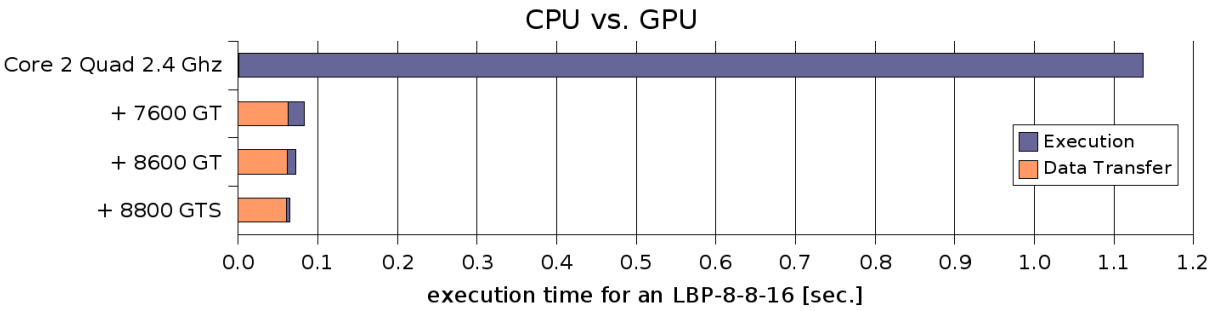


Figure 6.49: Speed comparison CPU / GPUs

6.6.2.1 Conclusion

The reformulation of the multi-scaled Local Binary Pattern Texture Analysis Operator on a GPU allows to effectively reduce the run time for feature extraction down to a level where it has no noticeable impact on the runtime of the whole appearance-based terrain classification subsystem anymore. Results show that even using older graphics cards, ≈ 10 fps can be attained even with the full resolution of 1600x1200 pixels, while the CPU remains idle except for the data transfer.

Even more importantly, it has highlighted the tremendous potential of the emerging field of GPU computation to speed up lengthy, but well parallelizable calculations. Therefore, the color and variance operators, the recursive splitting stage of image segmentation or the stereo reconstruction of the shape-based terrain classification methods are viable candidates to be ported to the GPU. If this can be done successfully, it is highly likely that the entire visual terrain traversability estimation can be run with a frequency of about 1 Hz. This would allow the use during robot motion and could provide a large benefit to the pilot's obstacle avoidance capabilities.

6.7 Conclusion

Summary

The goal of the approach presented in this chapter was to develop a system capable of estimating the traversability of terrain based on its appearance, rather than the geometrical shape. With this system, it becomes possible to distinguish vegetation types which can be traversed (such as tall grass) from intrafficable types such as bushes or trees.

In order to achieve high classification performance and robustness, a combination of three different, complementary visual features has been selected as information sources. Those include a color operator based on the HSV color space, a multi-scale local binary patterns texture feature and an operator sensitive to contrast variations. During operation, similar image regions are first identified in an unsupervised segmentation step. The clustered results are then classified into traversable or untraversable terrain types based on an ensemble of three k-NN classifiers.

The classifiers were initially trained with labeled sample images, but an online learning method has been developed in order to update the classifiers and adapt them to new terrain types. With this novel technique, traversability information is gathered through

observation of the piloting behaviors while the robot moves along a topological edge. After the robot has concluded the movement, the recorded traversability information is matched with regions of classified images and added as new samples to the classifiers' knowledge databases.

The appearance-based image classification process was integrated into the complete navigation system developed in the scope of this thesis by mapping the obtained traversability information onto a local traversability map that can be seamlessly integrated into the cost prediction and exploration algorithms of the navigator. This successfully augments the robot's long range path planning capabilities with appearance-based traversability information.

Discussion and Outlook

The conducted experiments showed that the overall classification accuracy of the develop method is quite high, even given the limited amount of training samples provided. While the color features worked best for the first test set used in the evaluation, the strengths of the textural features became visible in the time series experiment that included more pronounced illumination changes. The combination of these classifiers was shown to successfully combine the strengths of each single classifier, leading to a good and robust overall performance.

Furthermore, the online learning scheme was shown to be usable both in simulation and in real application. It allows to build a traversability database from scratch using the self-observational learning technique developed for this purpose. This addresses the lack of adaptability which is the most prominent drawback of established, statically trained traversability estimation methods.

Additionally, it was shown that the generated confidence measures are high if samples are classified correctly, and much lower if they are classified incorrectly. Thus, wrong classification results are not so harmful in general since the navigator relies less on them. Alternatively, a filtering mechanism could be established based on the confidence measure.

Although the established system works well, many improvements are still possible. For one, processing speed has been identified as a problematic issue during testing and currently prevents the terrain analysis to be employed with high frequency. However, the implementation of the time consuming textural structure extraction on a GPU and the resulting speedup (13 times faster) has marked parallel computing on graphics hardware as a very promising means to solve this issue.

Concerning the algorithm accuracy, the conducted experiments also showed that the unsupervised segmentation step is problematic and introduces significant amounts of classification errors. Pixel-wise segmentation methods should therefore be explored to reduce the impact of erroneous segmentations.

Finally, the combination of classifiers is currently done using a simple (weighted) voting technique. Many other, more sophisticated classifier fusion approaches have been presented in the literature. Further experiments should be conducted in order to determine their usability in the proposed terrain classification method. In this context, the confidence measures could possibly also be modified to represent real class probabilities, so that a rigorous mathematical treatment becomes possible.

7. Conclusion

7.1 Summary

This thesis addresses the problem of finding a global robot navigation strategy for rugged off-road terrain which is *robust* against inaccurate self-localization and *scalable* to large environments, but also *cost-efficient*, e.g. able to generate navigation paths which optimize a cost measure closely related to terrain traversability.

At the beginning of the thesis, it was argued that an appropriate cost measure which quantifies the traversability of different travel options is essential to select optimally traversable paths for global navigation. It was conjectured that many established methods for off-road navigation generate such a cost measure using a fine-grained and bulky metrical world model which can not easily be scaled up to map large, rugged environments with substantial vegetation. Other approaches employ a topological map which abstracts completely from finer metrical details. While topological techniques scale better and are less affected by bad localization, these techniques also abstract from most issues related to terrain traversability. Thus, they do not support cost-efficient path planning in off-road terrain well.

Based on these conjectures, it was hypothesized that the formulation of a new strategy which unites aspects of both metrical and topological approaches could exploit the strengths of both techniques without inheriting their weaknesses. Such a combination could allow the construction of a compact and thus scalable world model which is capable to support reasoning about traversability costs. In combination with continuous refinement of the model based on accumulated navigation experience, this could allow the definition of a scalable, yet cost-efficient navigation system.

In order to test this hypothesis, a new navigation methodology has been formulated which extends a primarily topological map with new aspects that enable cost-efficient path planning on the topological level. These novel contributions include:

- a special topological map with typed edges and nodes to signify speculative and consolidated connections and places,

- a multi-dimensional cost measure for topological edges which captures the most relevant aspects of terrain traversability,
- a mathematical method to use this cost measure for path planning with user-selectable priorities,
- algorithms to optimize the map after navigation successes and failures,
- a technique to learn consistent edge cost measures from scratch based on feedback from a local navigation layer,
- a set of methods to predict edge costs based on existing cost annotations,
- a new metrical data structure termed 'local traversability map' which contains cost modifiers to improve the cost prediction for edges leading into untraversed terrain,
- a set of methods to fill local traversability maps with cost information based on analyzing long range visual sensor data with respect to terrain shape and appearance.

Experiments conducted with the proposed navigation methodology validate the claim that the new, hybrid metrical / topological map structure indeed allows to perform cost-efficient path planning. At the same time, the navigation system does not need to construct an extensive metrical world model in order to reason about traversability costs. This makes the presented approach highly scalable and supports the initial hypothesis which encouraged a combination of purely metrical or topological navigation systems in order to combine their respective strengths.

The contributions listed above are the results of deliberations that were motivated by the set of initial questions raised in the introduction of the thesis. The following paragraphs provide more detail on these deliberations. The presentation is ordered according to the aspects that were initially identified as especially relevant for a scalable, cost-efficient navigation system.

Appropriate World and Travel Cost Representation

An extensive literature survey of map types used for in- and outdoor robot navigation evaluated the strengths and weaknesses of different map types for off-road navigation. The evaluation results supported the initial claim that established methods for off-road navigation either generate a cost measure suitable to plan well traversable paths using a fine-grained and thus badly scalable metrical world model or employ a topological map which abstracts too much from terrain traversability. The survey also indicated that a *hierarchical hybrid map* which combines a topological global map with multiple local metrical maps is a highly suitable candidate for scalable and cost-efficient global navigation.

Motivated by this result, a novel hierarchical hybrid world map for global off-road navigation was proposed and formally defined. The topological part of this map represents the global connectivity of places and allows path planning on the global scale. In contrast to many other topological maps used for indoor tasks, directed edges are used to allow the modeling of asymmetrical travel costs which arise frequently in sloped outdoor terrain. Furthermore, an additional type attribute allows to explicitly distinguish between verified, hypothetical and untraversable map elements. This eliminates mathematically unsound workarounds used by some other approaches that mark untraversable connections using e.g. unrealistically high cost values.

In order to assign travel costs to topological edges as required for cost-efficient path planning, a multi-dimensional cost measure was introduced to represent all relevant terrain aspects of outdoor terrain. The three cost factors risk, effort and familiarity were included in the measure because they model the key aspects of travel costs and can be correlated with observable robot experience. This was identified as a prerequisite to improve cost estimates through feedback learning.

To mitigate the topological map's lack of traversability cost information for terrain which is not covered by topological edges, a new metrical map structure termed 'local traversability map' was proposed. These maps constitute the metrical part of the hybrid world model and contain circularly arranged patches storing traversability cost modifiers for the terrain surrounding a topological node. It was confirmed by experiments that the capability to store cost information for these areas significantly improves the ability to predict risk costs for new topological connections. A sophisticated three-dimensional simulation environment was developed to conduct these experiments [Braun 07].

Interaction of Global and Local Navigation Layers

In order to use stored costs to perform cost-efficient path planning, a formalism was proposed to map multi-dimensional cost measures onto scalar values. This allows the application of standard graph algorithms for path planning. The introduction of a 'motivational state' which alters the relative weights of each cost factor in the mapping function offered an intuitive interface to tune path planning priorities according to specific mission requirements. The influence of different motivational states and the plausibility of the resulting paths was validated using both simulation and real-world experiments.

Based on the formulated techniques for path planning, the transformation of generated paths into driving commands for the local piloting layer was considered. In order to minimize swaying of the robot during motion, a new algorithm was proposed which specifies both target positions and suitable target orientations.

The interaction between navigator and pilot during path traversal also necessitated the specification of a reliable way to detect the arrival at a target node despite of inaccurate self-localization. The proposed 'arrival detection with postponement' method was designed to fulfill this requirement. In contrast to a standard catchment area based approach, it is more adaptive and also able to let the robot converge to the precise location of the topological node in case localization accuracy is actually better than anticipated. Furthermore, an algorithm to detect failures to reach a target position was developed. It is able to recognize both dead- and livelocks while the robot approaches a given target position.

Once a navigation success or failure is detected, the navigation system has gained information which can be used to refine its map. To make future path planning steps more efficient, algorithms to represent such events in the topological map were developed and experimentally validated. The successful arrival at a node is used to correct badly placed node positions. The fusion of nodes that move too closely together thereby ensures a maximal node density. Therefore, it can be assumed that nodes can always be identified uniquely using the available localization accuracy. Navigation failures result in untraversable connections which are excluded from further path planning.

Consistent Travel Cost Assessment

Building on the proposed cost representation and the basic capabilities to plan and traverse paths, the specification of consistent edge traversal costs was addressed. In order to estimate such costs even though the global navigation layer does not know the actual robot trajectory for a given edge in advance, a novel *a posteriori learning* scheme was proposed [Braun 08a]. This cost learning method observes the pilot's behaviors during motion and integrates the observations afterwards into multi-dimensional cost information that accurately represented the costs incurred during driving. A spatio-temporal integration step was introduced for behaviors reacting to exterior stimuli in order to ensure independence of the integration value from variable robot speeds. For proprioceptive behaviors, a purely temporal integration was found to be sufficient.

Tests showed that the pilot can react differently in each traversal and thus introduces a high degree of uncertainty in the assessed edge cost. To handle this uncertainty in the path planning stage, the mapping of the cost measures to the scalar value used for path planning was extended to incorporate the variance of the assessed travel cost measures. This allows the path planning to be tuned either 'optimistical' or 'pessimistical' and offers another degree of freedom to adjust path planning to mission requirements. Results concerning this aspect have been published in cooperation with J. Hirth [Hirth 07].

The performance of the proposed map model, cost measure and the a posteriori learning scheme was evaluated in a series of simulation and real-world experiments. The obtained results document that the proposed method is capable of refining a given map by improving node positions, detecting untraversable connections and removing them from further planning. Also, it allows to build sensible map edge costs starting from an initially unannotated topological map. By virtue of the cost learner's design, these costs are consistent with the experience of the local pilot, thus capturing all obstacle encounters that required intervention as well as all factors that influenced the robot's energy consumption. All this was achieved without adding new domain knowledge to the global navigation system, except for specifying which part of the pilot is to be observed for what cost factor.

Prediction of Travel Costs and Map Extension

While the assessment of consistent edge costs is suitable to learn and consolidate cost information for known edges which are passed frequently, the exploration of unknown terrain or hypothetical edges does not benefit from it. In order to extend the navigation system, four new extrapolation techniques were devised to transfer existing cost information to unannotated links. Three of these methods reuse information stored in the topological part of the map, but incorporate data on different levels of locality. The coarsest method builds a global cost model, the second technique constructs a local model and the third approach predicts costs based solely an edge's direct inverse twin. This allows to exploit the closest source of cost information available for a connection in question. The presented approach is able to use increasingly accurate extrapolation techniques as the available cost information accumulates. This was verified and quantitatively analyzed by an extensive large-scale simulation test.

In order to account for terrain properties which have not been included into existing edges and are therefore not captured by the extrapolation of topological map information, a fourth cost prediction algorithm was proposed which uses the local traversability maps of

the hybrid world model as information source. The system fills these maps by reusing the obstacle memory of the local pilot and extracting traversable free space and untraversable obstacles. It was shown that this method is very efficient as it does not involve additional sensor analysis steps, yet provides highly accurate cost modifiers at close ranges. By combining the learned costs in the topological edges and the modifiers stored in the local metrical map, the cost extrapolation accuracy can be greatly improved especially with respect to the 'risk' cost factor, which depends on the amount of obstacle evasions during edge traversal. This effect was quantitatively analyzed along with the performance of the other extrapolation techniques in the experimental validation.

Based on the developed cost extrapolation methods, an exploration strategy was proposed which generates new connection hypotheses from two sets of possibilities. After considering all valid hypotheses, only the candidate with the lowest predicted costs is actually implemented. This keeps the map as small as possible, in accordance to the formulated objective to retain a minimal world model.

Asides from simulation results, the devised approach's performance was put to the test during a real-world, competitive scenario posed during the ELROB 2008 trial. In this trial, the global navigation system was capable to recover from an impassable obstacle that blocked the predefined route and generate an alternative route which led the robot safely along a pathway over 1 km, until the alternative route merged again with the predefined one. This successful demonstration of global navigation skills led to the qualification of the team for the final scenario. This was accomplished by just 3 other teams out of 11 contestants. RAVON actually achieved the highest possible scores for navigation autonomy in both the qualification and the final run.

Improved Large-Scale Travel Cost Prediction using Sensor Information

The developed cost prediction method which relies on the pilot's local obstacle memory is limited to predictions within the memory's maximal range. In order to augment the cost prediction capabilities of the navigation system at greater distances, two additional methods were researched which analyze high-resolution visual sensor data and extract terrain traversability information over much longer ranges.

A *shape-based terrain analysis* approach was devised which approximates the surrounding terrain using either planes containing slope information or purely height-based elevation planes. Obtained experimental results revealed that this novel combination of two established models allows to extend both range and accuracy of the obtained terrain model compared to just using a single variant [Braun 08b]. They also showed that the method is applicable to a variety of different outdoor environments and provides useful information within a range of up to 30 meters. The extracted slope information is mapped to cost modifiers and stored in a local traversability map that can be seamlessly integrated into the cost prediction algorithm introduced before. In this way, the world model remains minimal and focused on the central topic of supporting cost-efficient navigation.

To distinguish traversable vegetation types (such as tall grass) from intrafficable types such as bushes or trees, the second sensor based method that was proposed in this thesis estimates the traversability of terrain based on its *visual appearance*. Stability against environmental changes is achieved by considering a combination of the three complementary visual features color, contrast and texture. After unsupervised clustering, regions are classified into traversable or untraversable terrain types based on a k-NN classifier ensemble.

As before, the resulting traversal costs modifiers are put into a local traversability map and attached to the corresponding topological node for the use within the cost prediction algorithm.

The appearance-based approach was subsequently augmented with an online learning method to reduce the amount of a priori world knowledge that has to be invested into training the classification database. Using the same methodology as the cost assessment algorithm, the terrain appearance learning method updates the classifier databases by observing the piloting behaviors. With this new technique, traversability information is continually gathered while the robot moves along a topological edge. The capability to learn sensible terrain classifications from scratch was demonstrated experimentally.

A second improvement was put forward to improve the computation speed of the sensor analysis methods. A GPU based method was developed to extract the visual features required for terrain classification with increased efficiency. Results concerning this aspect have been published in cooperation with G. Zolynski [Zolynski 08]. The obtained speedup of more than a factor of 13 revealed the potential of this type of optimization and revealed a possible way to make the traversability estimation executable during robot motion.

The good performance of the appearance-based image classification and the online learning scheme was validated by experiments measuring the classification accuracies for a set of typical outdoor images. Also, the fault liability, time stability and confidence distribution of the classification method were evaluated. Research results related to this have been published in [Braun 08c] and [Rauber 08].

7.2 Evaluation of the Proposed Global Navigation Methodology

The global navigation system proposed in this thesis reveals a viable new way to perform cost-efficient path planning using the proposed metrical / topological hybrid map structure. At the same time, the need to build and maintain a fine-grained, global world model which would reduce the scalability of the approach is avoided. These results support the initial hypothesis that a scalable yet cost-efficient navigation system can be designed by striking a balance between the two well established types of purely metrical or topological navigation systems.

The combination of the proposed hybrid world model, the multi-dimensional cost measure and the a posteriori learning of costs through self-observation provides a significant benefit for efficient navigation in an *approximately known environment*. In such a scenario, the navigator can autonomously build up cost estimates for the topological edges and gradually optimize path selection. Possible applications that can benefit from these capabilities include repetitive transportation tasks (e.g. on construction sites), border patrols or the continuous monitoring of environmentally relevant parameters in a large, vegetated environment.

It was shown that self-observation indeed allows to reuse domain knowledge invested in a local navigation layer and produce travel cost information at a higher level of abstraction. The presented strategy complements the 'near-to-far' learning methods which have been proposed by other researchers before to transfer sensor related knowledge from close range to long range sensors. These methods are similar in spirit to the transfer of cost

information in the domain of the world representation, but the work conducted in this thesis covers the *results* of sensor interpretation, rather than the interpretation itself. In summary, the obtained results strongly suggest that the idea of self-observation is a very general concept, which should be explored further to improve the adaptability of additional aspects of robotic systems.

Cost-efficient navigation in unknown territory is advanced by the research documented in the second part of this thesis. The methods which were proposed to predict costs of new connections through traversability analysis respectively cost extrapolation introduce several novel techniques to this field of research. The methods that estimate costs based on the existing topological map can provide highly accurate cost modifiers in well connected map regions. Likewise, the prediction technique which reuses the obstacle memory of the local pilot can estimate costs well close to topological nodes. Both methods are fast enough to be used continuously.

The two traversability analysis method which use a long range visual sensor dedicated specifically to this task require more processing time, but cover a much larger area. Their use ultimately enables more foresighted exploration decision at critical locations. Both methods complement each other. While the shape-based analysis excels at determining terrain slopes and hence is able to yield accurate predictions for the effort cost factor, the appearance-based technique is better suited for the discrimination of flexible and inflexible vegetation, which reflects more strongly in the risk cost component.

The conducted experiments support the claim that the developed approach is able to explore routes towards a goal for which no a priori map exists. The presented cost prediction techniques allow the navigation system to place new topological nodes in directions that appear well traversable. If this does not produce a feasible path to the goal position, the topological map allows to backtrack and attempt a different route. Asides from simulation results, an account for the approach's performance has been given during the highly successful ELROB 2008 trials.

Of course, the researched methods also exhibit drawbacks. For one thing, arrival detection at a topological node and the entire piloting sublayer still depends on a (coarse) knowledge of the robot's own global position. Therefore, robot operation entirely without GPS is not possible at the moment. Although normal foilage allows sufficient accuracy for the coarse localization required, denser (rain-)forests or military applications require alternative strategies.

Also, the presented cost learning scheme assumes that the robot's pilot generates trajectories which are at least somewhat comparable with each other across different traversals of the same topological edge. While this is frequently the case, there exist situations where a single decision (such as turning to the left or right in front of an obstacle) alters the resulting trajectory profoundly. The costs of such edges are difficult to estimate consistently with the current approach and the proposed cost prediction method may fail.

Furthermore, the proposed exploration and cost prediction methods can also be improved at various places. All predictions suffer from not knowing *exactly* where the robot will actually drive once commanded to move. This is the price to be paid for abstracting from the local obstacle avoidance problems on the global navigation level and letting the somewhat unpredictable pilot do the ground work. A more predictable local navigation is thus

highly desirable. Additionally, the shape-based modeling approach suffers from the inability to cope with overhanging objects and the unavoidable stereo matching unreliability. The appearance-based technique loses a substantial amount of precision due to the coarse unsupervised segmentation stage and needs to be put on a more rigorous mathematical foundation to increase the expressive power of the estimated confidence measures.

7.3 Future Perspectives

Many aspects of the presented global navigation strategy can be further improved. Some of the shortcomings that became apparent in the experimental validation can be addressed rather quickly. For one, the speed of the visual terrain analysis steps can be increased significantly by integrating already existing methods for sparse and dense stereo reconstruction using a GPU. Also, the k-NN classification stage of the appearance-based analysis can be sped up using appropriate distance measures and efficient data structures such as kd-trees. For the shape-based method, the capability to handle overhangs is vital once the system is used in denser forests. The required modifications are straightforward to implement and have already been sketched out in section 5.4. For the appearance-based method, the calculated confidence scores should be transformed into probabilities in a mathematically sound way so that the resulting cost modifiers become accessible to a more thorough analysis.

Apart from the sensor analysis methods, the cost learning and prediction methods would benefit greatly from an extensive, long-term experiment using the real robot in a real environment. Although no methodical difficulties are expected, the obtained research results would be strengthened by using real test data instead of the simulation. However, such an extensive test imposes a lot of stress on the prototypical robot and is likely to cause failures in the already very run-down electric actuators. Therefore, the testing phase needs to be preceded by an overhauling of these parts.

On a long term, the integration of the global navigator and the local pilot should be improved. For example, a mechanism to recognize navigation-relevant places and passages could be added to the local pilot which interprets the local memory build during robot operation. Once a significant feature is detected, this mechanism could then signal the navigator, which can add new topological nodes or connections at the appropriate place. This would greatly improve the up-to-now only rudimentarily developed map extension capabilities. In this context, it would also be interesting to research methods which trigger the full sensor usage at critical locations. The usage of the time consuming terrain analysis methods could then be restricted to navigation decisions which really require that much information - in other cases, the other cost extrapolation methods might be sufficient.

The detection of locations which are relevant or even critical for navigation is just one example for a much more diverse area of future research. This area encompasses the extraction of further, *semantically meaningful* data from the available sensor information. Using more sophisticated pattern recognition techniques, the robot could classify obstacles or situations using labels such as a ‘fallen tree’ or ‘a pathway in the forest’ which could be translated into more accurate cost predictions on the on side, or into specialized behaviors to handle the specific navigation requirements of that situation.

The most profound possibilities to continue work on the navigation system address the core problem of the cost extrapolation and a posteriori learning scheme, the unpredictabil-

ity of the behavior-based piloting system. Several methods to improve the repeatability of emerging trajectories could be investigated. One possibility would be to construct an intermediate layer between the navigator and the pilot and plan short trajectory pieces based on the local obstacle memory of the pilot. Along with a way to ensure trajectory continuity, this would give the piloting system more foresight and could remove unnecessary ranking or inappropriate turning maneuvers. Another interesting course of action would be to detect edges that exhibit unpredictable trajectories (based on the spread of the costs, for example) and provide additional 'hints' to the pilot, such as: 'if you encounter an obstacle in about 5 meters, try to pass it on the left side'. The generation of these hints could be effected by analysis of the set of trajectories that has been produced so far for the edge. If several trajectory clusters can be identified, this could also be represented in the map by splitting the edge and annotating it with different hints.

Asides from the closely navigation related tasks, the self-localization of the robot could be made independent of the GPS sensor system. To compensate this, a sensor based (visual) method to recognize topological places and home in towards a distinct metrical location within that place is required. A working technique could be used as a fall-back mechanism if the GPS signal is too weak or otherwise unusable.

In the far future, one could also imagine the deployment of the presented global navigation system on a more maneuverable robot (such as a walking robot, for example) instead of the current wheeled system. This could provide an enormous advantage especially for rescue missions in disaster areas, where terrain is often too rugged for the established locomotion systems. In this regard, the high adaptability of the presented global navigation methodology and the generalizable self-observation methodology are a great boon, as both features simplify the transition to such a different physical platform substantially.

A. The SimVis3D Simulation and Visualization Framework

During the work on the large-scale navigation system presented in this thesis, it became apparent that a sophisticated *simulation* of the system behavior is required to reduce development time and test new approaches under repeatable conditions. Also, a good *visualization* of the current situation was sought to improve the understanding of the robot's behavior.

Because commercially available toolkits did not provide enough support to cover the simulation and visualization of all aspects relevant to the thesis, the SimVis3D framework was developed. SimVis3D is designed as a modular framework usable both for the simulation of optical sensor-systems like cameras or laser scanners and the visualization of spatial information such as topological graphs. Besides providing basic functionality to construct and parametrize three-dimensional scenarios, the framework offers strong extensibility through a mechanism that allows to easily add new, manually coded components. Furthermore, SimVis3D can also handle input from multiple client computers. The framework is built on top of the widely used 3D rendering library Coin3D, which is API-compatible to Open Inventor. Both rely on OpenGL for the actual rendering process and use a **scene graph** to store and structure graphical elements.

The design of SimVis3D is flexible enough to allow its use in many other applications besides the purposes of this thesis. In the robotics research lab, SimVis3D is actively used for the simulation and visualization of an indoor robot system, a walking machine and a humanoid robot. The framework is described extensively in [Braun 07]. The next two sections summarize the key points of this publication.

A.1 Scene Construction

The SimVis3D framework constructs a visual scene using a XML based description file supplied by the user (fig. A.1 shows an example). Each line contains either a **part**, **element** or **sensor** command.

The **part** command adds a new scene graph stored in a separate file to the current scene. Parts define the basic geometry of individual objects such as robots, obstacles or the

```
<part file="world.iv" name="WORLD" attached_to="ROOT"  pose_offset=
      "0 0 0 0 0 0" />
<part file="robot.iv" name="ROBOT" attached_to="WORLD" pose_offset=
      "0 0 0 0 0 0" />
<element type="pose" name="R_POSE" attached_to="ROBOT" x="1" y="2"
         z="3" roll="0" pitch="0" yaw="-90" />
<sensor type="camera" name="R_CAM" attached_to="ROBOT" pose_offset=
        "3 0 5 0 0 0" />
```

Figure A.1: Example XML based scene description

static environment. New parts can be added to the existing scene at **insertion point** nodes in the scene graph. Optionally, it is possible to supply a static offset to the insertion point using the `pose_offset` attribute. Each part command creates a new insertion point (named as the part itself) in it's local coordinate frame. With this mechanism, it is very easy to nest parts and create a hierarchical scene. This allows the user to specify relations between objects, i.e. parts that are physically attached to another (in the example, the camera `R_CAM` is attached to the robot `ROBOT` which in turn is positioned relative to the `WORLD`, which is anchored at the predefined simulation origin `ROOT`).

The **element** command instantiates a *scene modification object* whose class is specified by the element name. The object is then free to extract any further configuration data from the XML command and modify the scene graph in any way it sees fit (it generally adds nodes to the scene graph at the insertion point). It is important to stress that all modifications are encapsulated inside the object; the SimVis3D framework is unaware of the element's internal workings. The only interface between them are named **parameters** exported by the element, which are floating-point scalar values that can be modified by the framework. If parameters change, the scene graph updates are carried out solely by the instantiated element. In the example in figure A.1, the third line adds an element which exports six parameters `x`, `y`, `z`, `roll`, `pitch`, `yaw` to the framework and uses their value to modify the pose of the robot relative to its insertion point, e.g. the `WORLD`.

The **sensor** command includes simulated sensors in the scene, for example a camera attached to a robot. Employing the same mechanism as the instantiation of elements, the sensor command creates a *sensor object* of the requested type at the given insertion point. Line 4 in figure A.1 creates a camera attached to the robot with a static offset relative to the robot's local coordinate frame.

This component based architecture allows users to extend SimVis3D with almost any desired interaction capability by implementing a custom element or sensor class and registering it to the object factory. The encapsulation guarantees that no changes to other components of SimVis3D are required.

A.2 Interfacing with SimVis3D

SimVis3D explicitly supports input data from different processes or even different physical machines. The data transport between computers is done via standard interprocess

communication (IPC) using e.g. shared memory or named pipes. SimVis3D was designed to be easily adaptable to a variety of these techniques by offering a very simple interface.

This interface consists of *four data arrays*, *stored compactly* in memory. The arrays contain:

1. structs of `element descriptors`
2. floating-point `parameter values`
3. strings forming a `scene change request log`
4. strings forming a `scene change log`

Figure A.2 shows the interfaces produced by the example from fig. A.1.

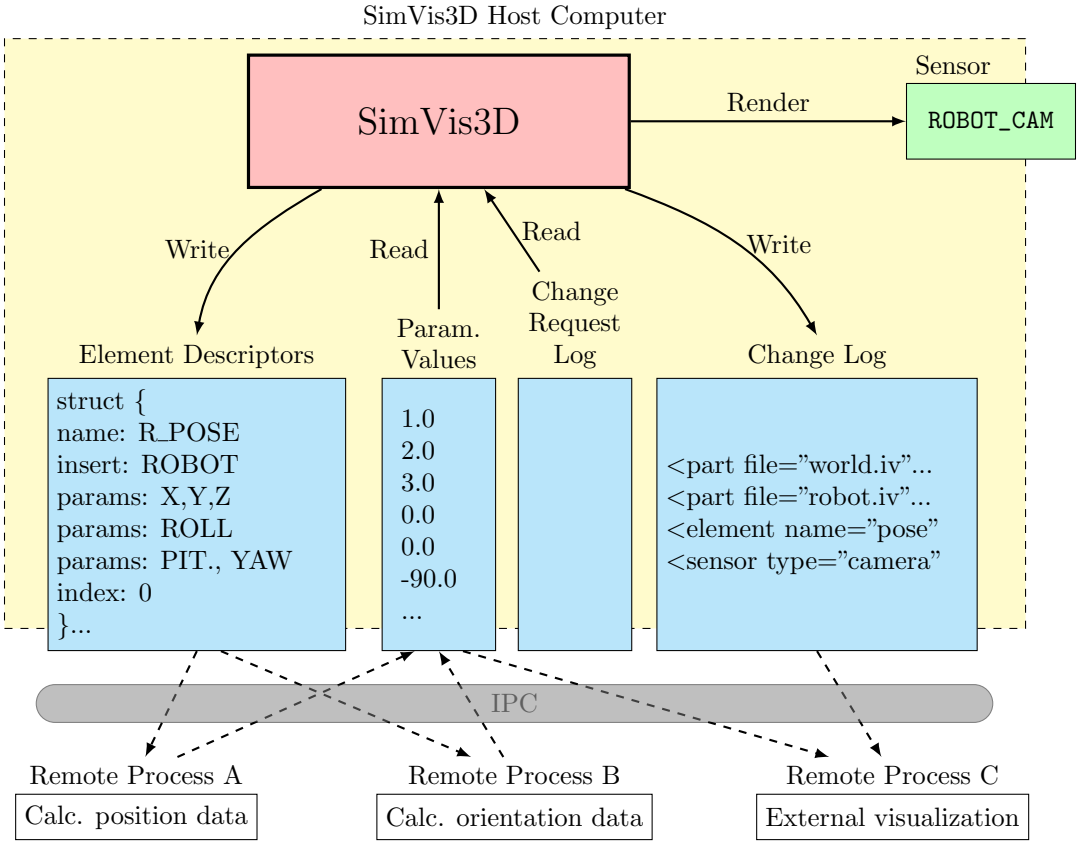


Figure A.2: The four remote access interfaces provided by SimVis3D

The `element descriptor` interface contains all information required to identify a specific element in the current scene and locate their parameter values. For each element, the interface array stores the element's name and insertion point, the number and names of its parameters and most importantly, the starting index of the parameter values in the `parameter values` interface array. This second array is a simple vector of the current values of all element parameters. With (remote) access to these two interface arrays, any process can update the parameters of every element. For example, process A in figure A.2 can calculate the robot position based on various sensors, locate the `R_POSE` element by scanning the `element descriptors` interface, extract the index of the position

parameters and write the calculated robot position into the `parameter values` array at the right place. Process B (executed on a different machine) can do the same with the orientation part of the robot pose. The reason to split the element descriptors and the actual values into two arrays is efficiency: After looking up the exact array index of a parameter value once, the client process can then manipulate the float values directly, reducing data transfer to the SimVis3D host process enormously.

While the first two arrays allow the manipulation of existing elements, the other two interfaces permit external processes to add new components or track these structural changes. In order to add a new scene component, a client can write XML commands similar to those given in figure A.1 to the `scene change request log`. The framework then executes them and confirms this by adding the XML strings to the `scene change log`. This mechanism allows remote clients to request the addition or removal of elements or parts during runtime. It also permits the construction of an identical scene by simply repeating all commands stored in the `change log` on a local scene graph. Thereby, an external client can render the same scene as is used for simulation from a different viewpoint for visualization purposes, or even exclusively render a scene that is only set up by the host machine but not rendered there. With this approach, SimVis3D supports both visual sensor simulation *and* visualization tasks seamlessly within the same basic framework. Process C in figure A.2 illustrates an example of such a use. All visualizations of the maps and navigation situations of RAVON displayed throughout the thesis are actually created using this setup.

A.3 Simulation of the RAVON Scenario

The software of the RAVON platform uses visual information from color cameras for stereo reconstruction, obstacle detection and visual odometry. In order to simulate these cameras, an outdoor scene similar to the standard testing area of the experimental platform was modeled using detailed obstacle geometries and high resolution textures. Figure A.3 shows a panoramic reconstruction of the real testing ground and a screenshot of the corresponding simulation scenario. A pair of corresponding positions is indicated by the black arrow.

(a) Real testing field (b) Corresponding simulation scene

Figure A.3: Real and simulated testing ground

To validate the adequacy of the camera simulation, images taken with the real robots stereo camera system were compared with images generated with virtual cameras in sim-

ulation. Figure A.4 shows a pair of real and simulated images along with disparity maps derived from them. The images depict the typical performance of the stereo algorithm. In

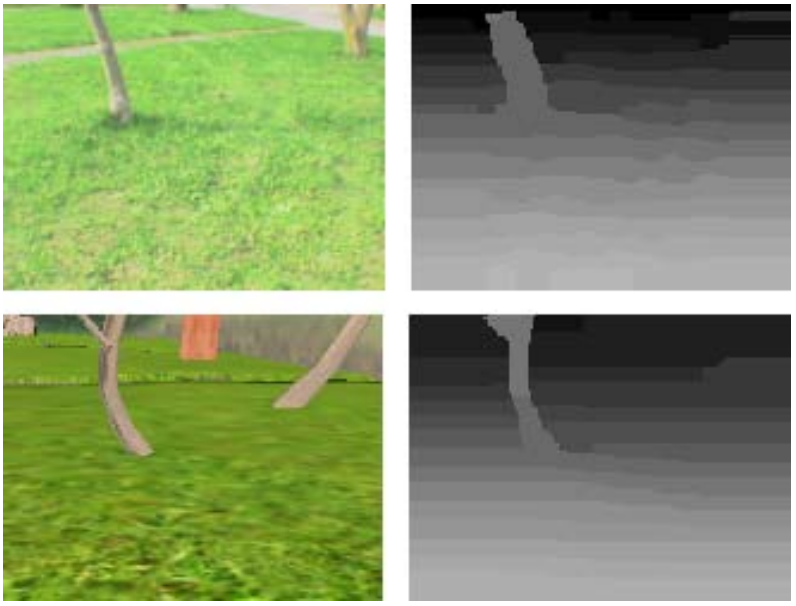


Figure A.4: Stereo reconstruction of real and simulated camera images
The disparity maps (right side) generated by stereo analysis of real (upper row) and simulated (lower row) camera images are very similar.

both cases, the stereo matching algorithm used in the RAVON project was used without any adjustments. The results demonstrate that the simulation is sufficient for stereo reconstruction and allows the image analysis algorithm to achieve realistic results. Similar experiments performed for the visual odometry computation led to the same conclusion.

Laser Scanners

The simulation of the three laser scanners on RAVON was realized by adding a `distance sensor` class to the SimVis3D framework. This class computes distance values between a given sensor center and the closest scene objects. Figure A.5 shows a visualization of the distance measures obtained by the frontal, horizontal scanner. The 3D scene depicts the overall terrain layout and highlights the direction of each distance measurement using red rays originating at the scanner position. The overlay on the lower left shows a 2D visualization of the distances as received by the robot's control software.

The sensor implementation evaluates the depth buffer used by the graphics hardware to render the scenario for a virtual camera placed at the scanner position. The produced color image is discarded, but the contents of the depth buffer is transformed into the required distance measurements. Using this technique, the GPUs high rendering speed is exploited and time consuming intersection calculations between scanner rays and the scene geometry are avoided. Thus, all three scanners on RAVON can easily be simulated at a rate of 30 Hz. Gaussian noise is added to the distance values to mimic the measurement inaccuracies of the real scanners. This simulation turned out to be relatively realistic for non-specular surfaces, which are predominant in outdoor environments. It does not suffice to model ice, water or other reflective surfaces.

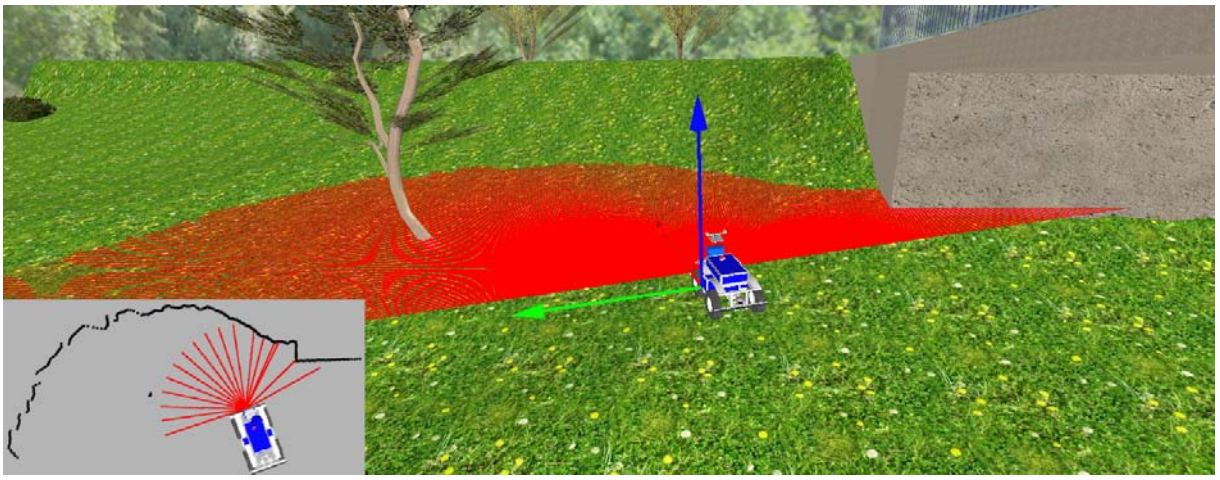


Figure A.5: Laser scanner simulation

Actuation System

The actuation system of RAVON and the interaction between the robot and the ground are simulated using an approximated physical model which *essentially* reflects the dependencies between physical forces correctly, but performs strong simplifications of the real physics involved. This was done due to the lack of a suitable physical simulation engine able to simulate wheel frictions at the time of SimVis3D's implementation.

In the physical model used in the scope of this thesis, the four wheel actuators of RAVON are simulated on the level of their DSP-based motor controllers. The real CAN connection is redirected to virtual controllers that offer the same interfaces as the real ones and implement the same motor control algorithm. However, the computed PWM output of each motor controller is transformed into a motor rotation speed (and thus encoder ticks, which are reported back to the control software) using an approximated motor model. It is based on the basic motor equation shown in eq. A.1. The equation states that the average voltage U_{pwm} 'seen' by the motor due to the applied PWM signal is equal to the back-induced voltage U_{ind} that results from the motor's current revolving speed plus the current I flowing through the motor times its internal resistance R.

$$U_{pwm} = U_{ind} + IR \tag{A.1}$$

Exploiting the facts that U_{pwm} is proportional to the PWM signal with strength P, U_{ind} is proportional to the motor speed v and I is proportional to the torque M generated by the actuator, this can be reformulated to:

$$c_1 v = c_2 P - c_3 M, \tag{A.2}$$

with proportionality constants c_1, c_2, c_3. Furthermore, neglecting the acceleration and deceleration phases of the robot (which are quite short in reality) and force components not parallel to the wheel's rolling direction, it can be assumed that the torque exerted by the motor is equal to the torque M_r resulting from the rolling resistance F_r of the wheels plus the torque M_d caused by the downhill force F_d that is introduced by the robot's weight on sloped terrain:

$$M = M_r + M_d \tag{A.3}$$

The physical laws of rolling friction and downhill force state that F_r is proportional to the cosine of the robot's roll angle α, while F_d is proportional to its sine value. However, because the robot's actuation system is very elastic and the terrain is typically very soft, the decrease of the rolling resistance with increasing roll angle of the robot was deemed negligible. Thus, F_r is assumed to be constant with respect to α. This leads to:

$$M = c_4 + c_5 \sin \alpha \tag{A.4}$$

Combining equations A.2 and A.4 and substituting products of proportionality constants with new constants c'_1, c'_2, c'_3, the relation between motor speed v and set PWM value P can be approximated using the following equation:

$$v = c'_1 P - c'_2 - c'_3 \sin \alpha \tag{A.5}$$

These three proportionality coefficients were determined heuristically so that the resulting PWM / speed pairs matched the results of three test runs of the real RAVON system on flat terrain and two ramps with slopes of approximately 10° and 15°.

Robot Pose Simulation

After each simulation step of the motor controllers, the new robot pose in the simulation scene is computed. This is done by estimating the new wheel positions using the current wheel speeds and a kinematic model. The estimated wheel positions are then corrected so that the wheels have contact with the surface mesh of the simulated terrain. Internally, this is accomplished using a distance sensor directly overhead the robot which determines the distances between estimates wheel position and the surface.

Current Sensor Simulation

The current sensors of the motor controllers are simulated in a physically plausible way using equation A.1. Rearranging the equation and introducing proportionality constants yields:

$$I = c'_4 P - c'_5 v \tag{A.6}$$

Again, the proportionality coefficients have been estimated so that the simulated currents approximate the real motor currents measured in the three test runs.

Scene Visualization

A separate instance of SimVis3D is used to visualize the robot's situation and the current navigation map. In the scope of this thesis, custom elements have been developed to show coordinate systems, topological map nodes, edges and local traversability maps. Figures A.6 and A.7 present examples of these components.

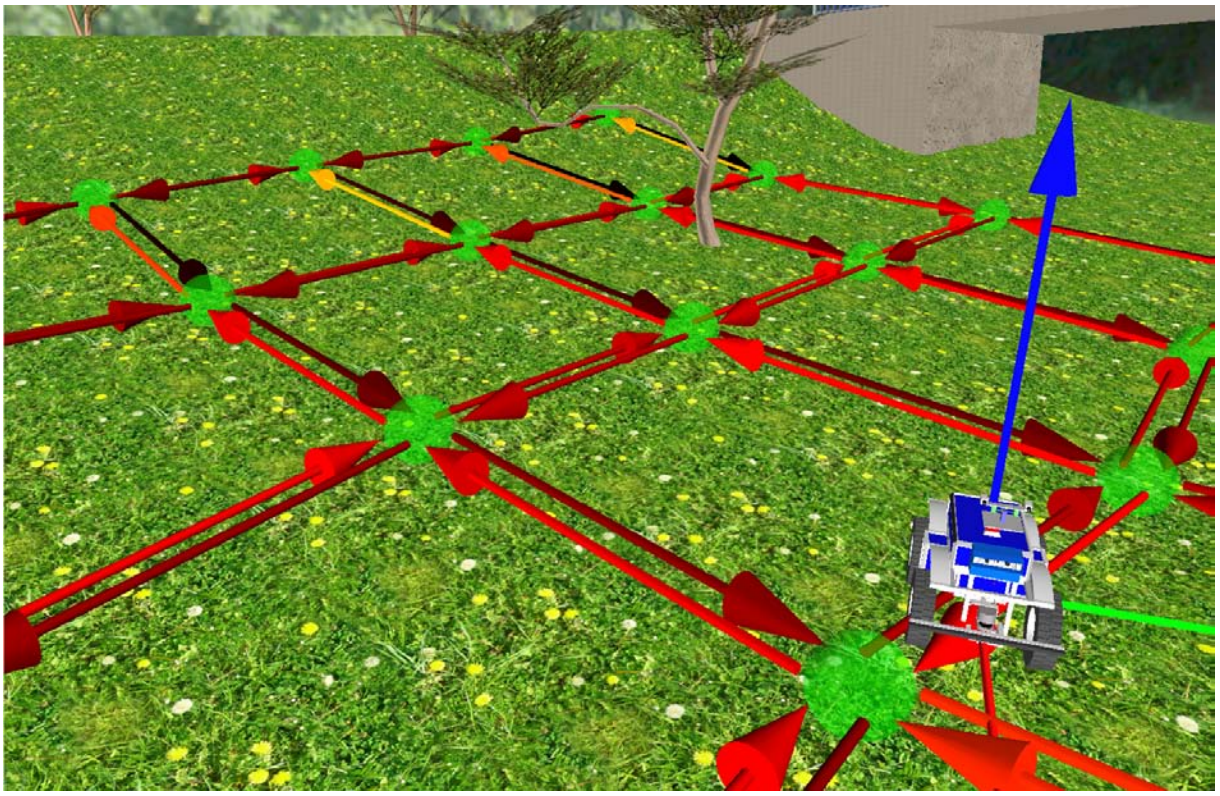


Figure A.6: Visualization of RAVON scene with overlaid topological map

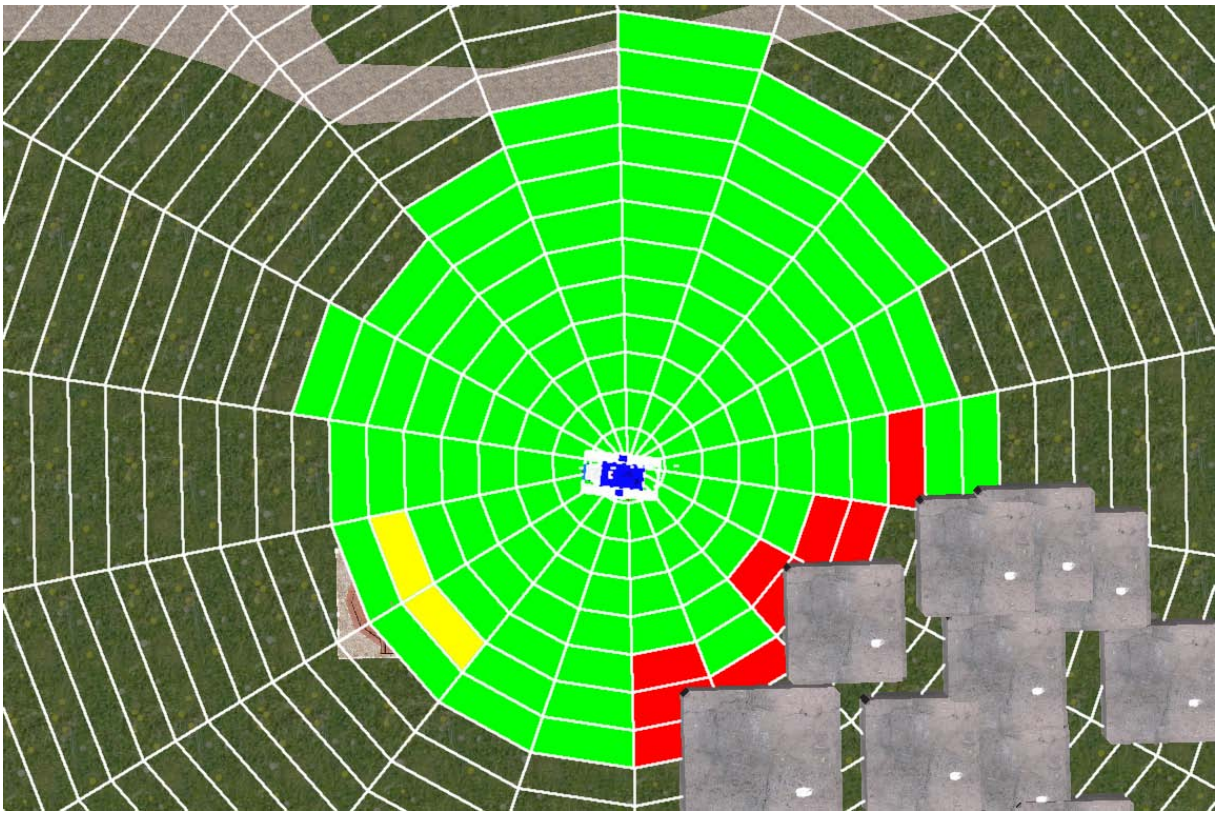


Figure A.7: Visualization of a local traversability map

Bibliography

[Andrade-Cetto 01] J. Andrade-Cetto, A. Sanfeliu, "Topological Map Learning for a Mobile Robot in Indoor Environments", in *Proceedings of the 9th Spanish Symposium on Pattern Recognition and Image Analysis*. 2001, pp. 221–226.

[Arlandis 02] J. Arlandis, J. C. Perez-Cortes, J. Cano, "Rejection Strategies and Confidence Measures for a k-NN Classifier in an OCR Task", in *16th. International Conference on Pattern Recognition ICPR-2002*. IEEE, 2002, pp. 576–579.

[Avis 90] D. Avis, H. Imai, "Locating a robot with angle measurements", *J. Symb. Comput.*, vol. 10, no. 3–4, pp. 311–326, 1990.

[Bellutta 00] P. Bellutta, R. Manduchi, L. Matthies, K. Owens, A. Rankin, "Terrain Perception for DEMO III", in *Proceedings of the IEEE Intelligent Vehicle Symposium 2000*. 2000, pp. 326–331.

[Besl 92] P. Besl, N. McKay, "A Method for Registration of 3-D Shapes", in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2. February 1992, pp. 239–258.

[Betke 94] M. Betke, K. Gurvits, "Mobile robot localization using landmarks", in *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 2. May 1994, pp. 135–142.

[Birchfield 99] S. Birchfield, C. Tomasi, "Depth Discontinuities by Pixel-to-Pixel Stereo", *International Journal of Computer Vision*, vol. 35, no. 3, pp. 269–293, 1999.

[Bitsch 08] H. Bitsch, "Outdoor Terrain Traversability Estimation based on Stereo Image Reconstruction", Diploma thesis, unpublished, Robotics Reseach Lab - TU Kaiserslautern, 2008.

[Bosse 03] M. Bosse, P. Newman, J. Leonard, M. Soika, W. Feiten, S. Teller, "An Atlas Framework for Scalable Mapping", in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2. September 2003, pp. 1899–1906.

[Braid 06] D. Braid, A. Broggi, G. Schmiedel, "The TerraMax autonomous vehicle: Field Reports", *J. Robot. Syst.*, vol. 23, no. 9, pp. 693–708, 2006.

[Braun 05] **T. Braun**, K. Szentpetery, K. Berns, "Detecting and Following Humans with a Mobile Robot", in *Proceedings of the EOS Conference On Industrial Imaging and Machine Vision*. June 2005.

[Braun 07] **T. Braun**, J. Wettach, K. Berns, "A Customizable, Multi-Host Simulation and Visualization Framework for Robot Applications", in *Recent Progress in Robotics: Viable Robotic Service to Human, Selected papers from ICAR07*, S. Lee, I. H. Suh, M. S. Kim, Eds., LNCIS Series, Springer-Verlag, 2007, vol. 370, pp. 357–369.

[Braun 08a] **T. Braun**, K. Berns, "Topological Edge Cost Estimation through Spatio-Temporal Integration of Low-level Behaviour Assessments", in *Intelligent Autonomous Systems (IAS-10)*, W. Burgard et al, Eds. IOS Press, July 2008, pp. 84–91.

[Braun 08b] **T. Braun**, H. Bitsch, K. Berns, "Visual Terrain Traversability Estimation using a Combined Slope/Elevation Model", in *Proceedings of the 2008 KI Conference.* 2008, pp. 177–184.

[Braun 08c] **T. Braun**, B. Seidler, K. Berns, "Adaptive Visual Terrain Traversability Estimation using Behavior Observation", in *Proceedings of IARP Workshop on Environmental Maintenance and Protection.* July 2008.

[Brennecke 03] C. Brennecke, O. Wulf, B. Wagner, "Using 3D Laser Range Data for SLAM in Outdoor Environments", in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* Las Vegas, October 2003, pp. 188–193.

[Brennecke 04] C. Brennecke, "Ein scanbasierter Ansatz zur exploratorischen Navigation autonomer mobiler Systeme in unstrukturierten Outdoor-Umgebungen", Phd thesis (in german), University of Hannover, 2004.

[Bronstein 81] I. N. Bronstein, K. A. Semendjajew, *Taschenbuch der Mathematik*, 21. Edition ed., G. Grosche and V. Ziegler and D. Ziegler, 1981.

[Brooks 87] R. A. Brooks, "Visual map making for a mobile robot", in *Readings in computer vision: issues, problems, principles, and paradigms*, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1987, pp. 438–443.

[Brown 66] D. C. Brown, "Decentering Distortion of Lenses", *Photometric Engineering*, vol. 32, no. 3, pp. 444–462, 1966.

[Brown 71] D. C. Brown, "Lens Distortion for Close-Range Photogrammetry", *Photometric Engineering*, vol. 37, no. 8, pp. 855–866, 1971.

[Brown 92] L. G. Brown, "A survey of image registration techniques", *ACM Comput. Surv.*, vol. 24, no. 4, pp. 325–376, 1992.

[Choset 01] H. Choset, K. Nagatani, "Topological simultaneous localization and mapping (SLAM): toward exact localization without explicit localization", *IEEE Transactions on Robotics and Automation*, vol. 17, no. 2, pp. 125 – 137, April 2001.

[Choset 96] H. Choset, "Sensor based motion planning: The hierarchical generalized voronoi graph", Dissertation, California Institute of Technology, 1996.

[Chown 95] E. Chown, S. Kaplan, D. Kortenkamp, "Prototypes, Location, and Associative Networks (PLAN): Towards a Unified Theory of Cognitive Mapping.", *Cognitive Science*, vol. 19, no. 1, pp. 1–51, 1995.

[Craig 89] J. J. Craig, *Introduction to Robotics*, 2 ed., Addison-Wesley Publishing Company, 1989. ISBN 0-201-09528-9.

[Cremean 06] L. Cremean, T. Foote, J. Gillula, G. Hines, D. Kogan, K. Kriechbaum, J. Lamb, J. Leibs, L. Lindzey, C. Rasmussen, A. Stewart, J. Burdick, R. Murray, "Alice: An information-rich autonomous vehicle for high-speed desert navigation: Field Reports", *J. Robot. Syst.*, vol. 23, no. 9, pp. 777–810, 2006.

[Dahlkamp 06] H. Dahlkamp, A. Kaehler, D. Stavens, S. Thun, G. Bradski, "Self-supervised Monocular Road Detection in Desert Terrain", in *Proceedings of Robotics: Scientce and Systems 2006 (RSS06)*. 2006.

[Dijkstra 59] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs", *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[Dissanayake 01] M. W. M. G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, M. Csorba, "A Solution to the Simultaneous Localization and Map Building (SLAM) Problem", *IEEE Transactions of Robotics and Automation*, vol. 17, no. 3, pp. 229 – 241, june 2001.

[Duckett 00] T. Duckett, U. Nehmzow, "Performance Comparison of Landmark Recognition Systems for Navigating Mobile Robots.", in *AAAI/IAAI*. AAAI Press / The MIT Press, 2000, pp. 826–831.

[Eggert 97] D. W. Eggert, A. Lorusso, R. B. Fisher, "Estimating 3-D rigid body transformations: a comparison of four major algorithms", *Machine Vision and Applications*, vol. 9, pp. 272–290, 1997.

[Eggert 98] D. W. Eggert, A. W. Fitzgibbon, R. B. Fisher, "Simultaneous Registration of Multiple Range Views for Use in Reverse Engineering of CAD Models", *Computer Vision and Image Understanding: CVIU*, vol. 69, no. 3, pp. 253–272, 1998.

[Elfes 89] A. Elfes, "Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation", Dissertation, Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, 1989.

[Elstrom 98] M. Elstrom, P. Smith, M. Abidi, "Stereo-based registration of ladar and color imagery", in *Intelligent Robots and Computer Vision XVII: Algorithms, Techniques, and Active Vision*, ser. Society of Photo-Optical Instrumentation Engineers (SPIE), D. P. Casasent, Ed., vol. 3522. october 1998, pp. 343–354.

[Erkan 07] A. Erkan, R. Hadsell, P. Sermanet, J. Ben, U. Muller, Y. LeCun, "Adaptive Long Range Vision in Unstructured Terrain", in *Proc. Intelligent Robots and Systems (IROS 07)*. November 2007, pp. 2421–2426.

[Estrada 05] C. Estrada, J. Neira, J. D. Tardos, "Hierarchical SLAM: Real-Time Accurate Mapping of Large Environments", in *IEEE Transactions of Robotics*, vol. 21, no. 4. August 2005, pp. 588–596.

[Faust 08] F. Faust, "A Versatile Approach to Fast Unsupervised Image Segmentation", Project thesis, unpublished, Robotics Research Lab - University of Kaiserslautern, July 2008.

[Feng 05] X. Feng, M. Pietikäinen, A. Hadid, "Facial expression recognition with local binary patterns and linear programming", *Pattern Recognition and Image Analysis*, vol. 15, no. 2, pp. 546–548, 2005.

[Ferguson 04] D. Ferguson, A. Stentz, "Planning with Imperfect Information", in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, vol. 2. September 2004, pp. 1926–1931.

[Fernández-Madrigal 99] J. A. Fernández-Madrigal, J. González, L. Mandow, J.-L. P. de-la Cruz, "Mobile Robot Path Planning: A Multicriteria Approach", *Engineering Applications of Artificial Intelligence*, vol. 12, pp. 543–554, 1999.

[Filliat 02] D. Filliat, J.-A. Meyer, "Global localization and topological map-learning for robot navigation", in *From animals to animats 7: Proceedings of the Seventh International Conference on Simulation of Adaptive Behavior*. Cambridge, MA, USA: MIT Press, 2002, pp. 131–140.

[Fischler 87] M. A. Fischler, R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography", in *Readings in computer vision: issues, problems, principles, and paradigms*, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1987, pp. 726–740.

[Foreman 07] M. Foreman, A. Kanamori, M. Magidor, *Handbook of Set Theory*, Springer, 2007.

[Fox 01] D. Fox, S. Thrun, W. Burgard, F. Dellaert, "Particle Filters for Mobile Robot Localization", in *Sequential Monte Carlo Methods in Practice*, A. Doucet, N. de Freitas, N. Gordon, Eds., Springer, New York, 2001, ch. 19, pp. 401–428.

[Fox 99] D. Fox, W. Burgard, S. Thrun, "Markov Localization for Mobile Robots in Dynamic Environments", *Journal of Artificial Intelligence Research*, vol. 11, pp. 391–427, 1999.

[Franz 00] M. O. Franz, H. A. Mallot, "Biomimetic robot navigation", *Robotics and Autonomous Systems*, vol. 30, pp. 133–153, 2000.

[Franz 98] M. O. Franz, B. Schölkopf, H. A. Mallot, H. H. Bülthoff, "Learning view graphs for robot navigation", *Autonomous Robots*, vol. 5, pp. 111–125, 1998.

[Gemme 05] S. Gemme, J. Bakambu, I. Rekleitis, "3D Reconstruction of Environments for Planetary Exploration", in *CRV 05: Proceedings of the 2nd Canadian conference on Computer and Robot Vision*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 594–601.

[Guivant 01] J. E. Guivant, E. M. Nebot, "Optimization of the Simultaneous Localization and Map-Building Algorithm for Real-Time Implementation", *IEEE Transactions on Robotics and Automation*, vol. 17, no. 3, pp. 242–257, june 2001.

[Hach 07] A. Hach, "Laser-Based Obstacle Detection in Off-Road Terrain", Diploma thesis, unpublished, Robotics Research Lab - University of Kaiserslautern, July 2007.

[Hadsell 07a] R. Hadsell, P. Sermanet, J. Ben, A. Erkan, J. Han, B. Flepp, U. Muller, Y. LeCun, "On-line Learning for Offroad Robots: Using Spatial Label Propagation to Learn Long-Range traversability", The Courant Institute of Mathematical Sciences, New York, Tech. Rep. CBLL-TR-2007-01-01, January 2007.

[Hadsell 07b] R. Hadsell, P. Sermanet, J. Ben, A. Erkan, J. Han, B. Flepp, U. Muller, Y. LeCun, "Online Learning for Offroad Robots: Spatial Label Propagation to Learn Long-Range Traversability", in *Robotics: Science and Systems III*, W. Burgard, O. Brock, C. Stachniss, Eds. Georgia Institute of Technology, Atlanta, Georgia, USA, June 2007.

[Hartley 03] R. Hartley, A. Zisserman, *Multiple View Geometry*, Cambridge University Press, 2003.

[Harwood 95] D. Harwood, T. Ojala, M. Pietikainen, "Texture classification by center-symmetric auto-correlation using Kullback discrimination of distributions", *Pattern Recognition Letters*, vol. 16, no. 1, pp. 1–10, 1995.

[Hebert 03] M. Hebert, N. Vandapel, "Terrain Classification Techniques from LADAR Data for Autonomous Navigation", in *Collaborative Technology Alliances conference*. May 2003.

[Hirth 07] J. Hirth, **T. Braun**, K. Berns, "Emotion Based Control Architecture for Robotics Applications", in *Proceedings of the Annual German Conference on Artificial Intelligence (KI)*. Osnabrück, Germany, September 10–13 2007, pp. 464–467.

[Horn 97] J. Horn, "Multicriterion Decision Making", in *Handbook of Evolutionary Computation*, IOP Publishing Ltd. and Oxford University Press, 1997, ch. F1.9:l - F1.9:15.

[Horowitz 76] S. L. Horowitz, T. Pavlidis, "Picture Segmentation by a Tree Traversal Algorithm", *JACM*, vol. 23, no. 2, pp. 368–388, April 1976.

[Howard 06] A. Howard, M. Turmon, L. Matthies, B. Tang, A. Angelova, "Towards Learned Traversability for Robot Navigation: From Underfoot to the Far Field", *Journal of Field Robotics*, vol. 23, no. 11–12, pp. 1005 – 1017, Jan 2006.

[Hu 97] H. Hu, M. Brady, "Dynamic global path planning with uncertainty for mobile robots in manufacturing", *IEEE Transactions on Robotics and Automation*, vol. 13, no. 5, pp. 760–767, October 1997.

[Huertas 05] A. Huertas, L. Matthies, A. Rankin, "Stereo-Based Tree Traversability Analysis for Autonomous Off-Road Navigation", in *WACV-MOTION 05: Proceedings of the Seventh IEEE Workshops on Application of Computer Vision (WACV/MOTION 05)*, vol. 1. Washington, DC, USA: IEEE Computer Society, 2005, pp. 210–217.

[IEEE 00] IEEE, "IEEE Standard for a high Performance Serial Bus-Amendment 1M", *IEEE standards*, 2000. ISBN 0-7381-1958-X.

[Jennings 97] C. Jennings, D. Murray, "Stereo vision based mapping and navigation for mobile robots", in *Proc. IEEE International Conference on Robotics and Automation.* 1997, pp. 1694–1699.

[Kagami 05] S. Kagami, Y. Takaoka, Y. Kida, K. Nishiwaki, T. Kanade, "Online dense local 3D world reconstruction from stereo image sequences", *IEEE/RSJ Int. Conference on Intelligent Robots and Systems*, pp. 3858–3863, 2005.

[Kim 06a] D. H. Kim, S. U. Jung, K. H. An, H. S. Lee, M. J. Chung, "Development of a Facial Expression Imitation System", in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* Beijing, China, October 9–15 2006, pp. 3107–3112.

[Kim 06b] D. Kim, J. Sun, S. M. Oh, J. M. Rehg, A. F. Bobick, "Traversability Classification using Unsupervised on-line Visual Learning for Outdoor Robot Navigation", in *ICRA*. 2006, pp. 518–525.

[Kim 07] D. Kim, S. M. Oh, J. M. Rehg, "Traversability Classification for UGV Navigation: A Comparison of Patch and Superpixel Representations", in *Proceedings: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 07).* 2007.

[Kittler 98] J. Kittler, M. Hatef, R. P. W. Duin, J. Matas, "On Combining Classifiers", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 3, pp. 226–239, 1998.

[Knight 01] J. Knight, A. Davison, I. Reid, "Towards constant time SLAM using postponement", in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 1. 2001, pp. 405–413.

[Kortenkamp 94] D. Kortenkamp, T. Weymouth, "Topological mapping for mobile robots using a combination of sonar and vision sensing", in *AAAI'94: Proceedings of the twelfth national conference on Artificial intelligence (vol. 2).* Menlo Park, CA, USA: American Association for Artificial Intelligence, 1994, pp. 979–984.

[Kruusmaa 03] M. Kruusmaa, "Global Level Path Planning for Mobile Robots in Dynamic Environments", *J. Intell. Robotics Syst.*, vol. 38, no. 1, pp. 55–83, 2003.

[Kuipers 00] B. Kuipers, "The spatial semantic hierarchy", *Artificial Intelligence*, vol. 119, no. 1–2, pp. 191–233, 2000.

[Kuipers 02] B. Kuipers, P. Beeson, "Bootstrap learning for place recognition", in *Eighteenth national conference on Artificial intelligence.* Menlo Park, CA, USA: American Association for Artificial Intelligence, 2002, pp. 174–180.

[Kuipers 77] B. Kuipers, "Representing Knowledge of Large-Scale Space", Dissertation, M.I.T. Artificial Intelligence Laboratory, 1977.

[Kuipers 91] B. Kuipers, Y. T. Byun, "A Robot Exploration and Mapping Strategy Based on a Semantic Hierarchiy of Spatial Representations", Department of Computer Sciences, University of Texas at Austin, Tech. Rep., 1991.

[Kuncheva 02] L. I. Kuncheva, "Switching between selection and fusion in combining classifiers: an experiment", *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 32, no. 2, pp. 146–156, 2002.

[Kweon 92] I. S. Kweon, T. Kanade, "High-Resolution Terrain Map from Multiple Sensor Data", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 278–292, February 1992.

[Lahdenoja 06] O. Lahdenoja, J. Maunu, M. Laiho, A. Paasio, "A massively parallel algorithm for local binary pattern based face recognition", in *Proceedings of the 2006 IEEE International Symposium on Circuits and Systems (ISCAS 2006)*. Island of Kos, Greece, May 21–24 2006, pp. 3730–3733.

[Lalonde 06] J.-F. Lalonde, N. Vandapel, D. Huber, M. Hebert, "Natural terrain classification using three-dimensional ladar data for ground robot mobility", *Journal of Field Robotics*, vol. 23, no. 10, pp. 839 – 861, November 2006.

[Lambrinos 00] D. Lambrinos, R. Moeller, T. Labhart, R. Pfeifer, R. Wehner, "A mobile robot employing insect strategies for navigation", in *Robotics and Autonomous Systems, special issue on Biomimetic Robots*, vol. 30. 2000, pp. 39–64.

[Lisien 03] B. Lisien, D. Morales, D. Silver, G. Kantor, I. Rekleitis, H. Choset, "Hierarchical Simultaneous Localization and Mapping", *IEEE/RSJ Int. Conference on Intelligent Robots and Systems*, vol. 1, pp. 448–453, 2003.

[Liu 05] C.-L. Liu, H. Fujisawa, "Classification and Learning for Character Recognition - Comparison of Methods and Remaining Problems", in *Neural Networks and Learning in Document Analysis and Recognition*. 2005, pp. 1–7.

[Lorusso 95] A. Lorusso, D. Eggert, R. Fisher, "A comparison of four algorithms for estimating 3-d rigid transformations", in *Proceedings of the British Machine Vision Conference*. 1995, pp. 237–246.

[Loui 83] R. P. Loui, "Optimal paths in graphs with stochastic or multidimensional weights", *Commun. ACM*, vol. 26, no. 9, pp. 670–676, 1983.

[Lu 94] F. Lu, E. Milios, "Robot Pose Estimation in Unknown Environments by Matching 2D Range Scans", in *Conference on Computer Vision and Pattern Recognition*. June 1994, pp. 935–938.

[Lucas 81] B. D. Lucas, T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision", in *IJCAI81*. 1981, pp. 674–679.

[Lumelsky 87] V. J. Lumelsky, "Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape", *Algorithmica*, vol. 2, no. 1, pp. 403–430, 1987.

[Lungarella 03] M. Lungarella, G. Metta, R. Pfeifer, G. Sandini, "Developmental robotics: a survey", *Connection Science*, vol. 15, no. 4, pp. 151–190, 2003.

[Mäenpää 03] T. Mäenpää, "The Local Binary Pattern Approach to Texture Analysis – Extensions and Applications", Dissertation, University of Oulu, 2003.

[Mäenpää 05] T. Mäenpää, M. Pietikäinen, "Texture analysis with local binary patterns", in *Handbook of Pattern Recognition and Computer Vision*, 3 ed., C. H. Chen, P. S. P. Wang, Eds., World Scientific, 2005, pp. 197–216.

[Manduchi 05] R. Manduchi, A. Castano, A. Talukder, L. Matthies, "Obstacle Detection and Terrain Classification for Autonomous Off-Road Navigation", in *Autonomous Robots*. 2005.

[Matarić 02] M. J. Matarić, "Situated Robotics", invited contribution to the Encyclopedia of Cognitive Science, Nature Publishing Group, Macmillan Reference Limited, November 2002.

[Matarić 92] M. J. Matarić, "Behavior-Based Systems: Main Properties and Implications", in *Proceedings of the IEEE International Conference on Robotics and Automation*, IEEE. Nice, France, May 1992, pp. 46–54. from the Workshop on Architectures for Intelligent Control Systems.

[Matarić 97] M. J. Matarić, "Behavior-Based Control: Examples from Navigation, Learning, and Group Behavior", *Journal of Experimental and Theoretical Artificial Intelligence*, vol. Special issue on Software Architectures for Physical Agents, 9(2-3), pp. 323–336, 1997.

[Moorehead 01] S. Moorehead, "Autonomous Surface Exploration for Mobile Robots", Dissertation, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 2001.

[Moravec 88] H. Moravec, "Sensor fusion in certainty grids for mobile robots", *AI Magazine*, vol. 9, no. 2, pp. 61–74, 1988.

[Moravec 93] H. Moravec, M. Blackwell, "Learning Sensor Models for Evidence Grids", in *CMU Robotics Institute 1991 Annual Research Review*, 1993, pp. 8–15.

[Moravec 96] H. Moravec, "Robot Spatial Perception by Stereoscopic Vision and 3D Evidence Grids", Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-96-34, September 1996.

[Nehmzow 00] U. Nehmzow, C. Owen, "Robot navigation in the real world:Experiments with Manchester's FortyTwo in unmodified, large environments.", *Robotics and Autonomous Systems*, vol. 33, no. 4, pp. 223–242, 2000.

[Nuechter 06] A. Nuechter, "Semantische dreidimensionale Karten für autonome mobile Roboter", Dissertation, University of Bonn, 2006.

[Nuechter 07] A. Nuechter, K. Lingemann, J. Hertzberg, H. Surmann, "6D SLAM 3D mapping outdoor environments: Research Articles", *J. Field Robot.*, vol. 24, no. 8–9, pp. 699–722, 2007.

[Ojala 01] T. Ojala, K. Valkealahti, E. Oja, M. Pietikainen, "Texture discrimination with multidimensional distributions of signed gray-level differences", *Pattern Recognition*, vol. 34, no. 3, pp. 727–739, 2001.

[Ojala 02a] T. Ojala, T. Mäenpää, M. Pietikainen, J. Viertola, J. Kyllonen, S. Huovinen, "Outex: New framework for empirical evaluation of texture analysis algorithms", in *International Conference on Pattern Recognition*. 2002, pp. 701–706.

[Ojala 02b] T. Ojala, M. Pietikäinen, T. Mäenpää, "Multiresolution Gray-Scale and Rotation Invariant Texture Classification with Local Binary Patterns", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 971–987, 2002.

[Ojala 96] T. Ojala, M. Pietikäinen, "A Comparative Study of Texture Measures with Classification Based on Feature Distributions", *Pattern Recognition*, vol. 29, no. 1, pp. 51–59, January 1996.

[Ojala 99] T. Ojala, M. Pietikäinen, "Unsupervised Texture Segmentation using Feature Distributions", *Pattern Recognition*, vol. 32, no. 3, pp. 477–486, 1999.

[Pfaff 05] P. Pfaff, W. Burgard, "An Efficient Extension of Elevation Maps for Outdoor Terrain Mapping", in *Proc. of the Int. Conf. on Field and Service Robotics (FSR)*. 2005.

[Pietikäinen 04] M. Pietikäinen, T. Nurmela, T. Mäenpää, M. Turtinen, "View-based recognition of real-world textures", *Pattern Recognition*, vol. 37, no. 2, pp. 313–323, February 2004.

[Pollefeys 01] M. Pollefeys, R. Moreas, F. Xu, G. Visentin, L. V. Gool, H. V. Brusse, "Calibration, Terrain Reconstruction and Path Planning for a Planetary Exploration System", *International Symposium on Artificial Intelligence, Robotics and Automation in Space*, june 2001.

[Pollefeys 02] M. Pollefeys, "Obtaining 3D Models with a Hand-Held Camera", *East-West Vision*, pp. 127–136, 2002.

[Proetzsch 07a] M. Proetzsch, K. Berns, T. Schuele, K. Schneider, "Formal Verification of Safety Behaviours of the Outdoor Robot RAVON", in *Fourth International Conference on Informatics in Control, Automation and Robotics (ICINCO), Angers, France*. INSTICC Press, May 2007, pp. 157–164.

[Proetzsch 07b] M. Proetzsch, T. Luksch, K. Berns, "The Behaviour-Based Control Architecture iB2C for Complex Robotic Systems", in *30th Annual German Conference on Artificial Intelligence (KI)*. Osnabrück, Germany, September 10–13 2007, pp. 494–497.

[Proetzsch 08] M. Proetzsch, T. Luksch, K. Berns, "Robotic Systems Design Using the Behavior-Based Control Architecture iB2C", *Journal of ROBOTICS AND AUTONOMOUS SYSTEMS*, 2008. submitted to.

[Randen 99] T. Randen, J. H. Husoy, "Filtering for Texture Classification: A Comparative Study", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 4, pp. 291–310, 1999.

[Rasmussen 02] C. Rasmussen, "Combining Laser Range, Color, and Texture Cues for Autonomous Road Following", in *Proceedings of the International Conference on Robotics and Automation*, vol. 4. may 2002, pp. 4320–4325.

[Rauber 08] T. W. Rauber, **T. Braun**, K. Berns, "Probabilistic Distance Measures of the Dirichlet and Beta distributions", *Pattern Recognition*, vol. 41, no. 2, pp. 637–645, 2008.

[Remolina 04] E. Remolina, B. Kuipers, "Towards a general theory of topological maps", *Artificial Intelligence*, vol. 152, no. 1, pp. 47–104, 2004.

[Rubner 01] Y. Rubner, J. Puzicha, C. Tomasi, J. M. Buhmann, "Empirical Evaluation of Dissimilarity Measures for Color and Texture", *Computer Vision and Image Understanding*, vol. 84, no. 1, pp. 25–43, october 2001.

[Ryu 99] B. Ryu, H. S. Yang, "Integration of reactive behaviors and enhanced topological map for robust mobile robot navigation", *IEEE Transactions on Systems, Man and Cybernetics, Part A*, vol. 29, no. 5, pp. 474–485, September 1999.

[Schäfer 05] H. Schäfer, "Security Aspects of Motion Execution in Outdoor Terrain", Diploma thesis, unpublished, Robotics Research Lab - University of Kaiserslautern, 2005.

[Schäfer 07] H. Schäfer, M. Proetzsch, "OBSTACLE DETECTION IN MOBILE OUTDOOR ROBOTS - A Short-term Memory for the Mobile Outdoor Platform RAVON", in *International Conference on Informatics in Control, Automation and Robotics (ICINCO), Angers, France*. 2007, pp. 141–148.

[Schäfer 08] H. Schäfer, A. Hach, M. Proetzsch, K. Berns, "3D Obstacle Detection and Avoidance in Vegetated Off-road Terrain", in *IEEE International Conference on Robotics and Automation (ICRA)*. Pasadena, USA, May, 19–13 2008, pp. 923–928.

[Schmitz 05] N. Schmitz, "Satellitenortung und Sensorfusion zur Lokalisierung von Fahrzeugen in unstrukturierter Umgebung", Diploma thesis, unpublished, Robotics Research Lab - University of Kaiserslautern, December 2005.

[Schmitz 06] N. Schmitz, M. Proetzsch, K. Berns, "Pose Estimation in Rough Terrain for the Outdoor Vehicle Ravon", in *37th International Symposium on Robotics (ISR)*. Munich, Germany, May 15–17 2006.

[Seidler 08] B. Seidler, "Texture based Terrain Traversability Estimation for Outdoor Robot Navigation", Diploma thesis, unpublished, Robotics Research Lab - University of Kaiserslautern, Kaiserslautern, Germany, March 2008.

[Seraji 03] H. Seraji, "Rule-based traversability indices for multi-scale terrain assessment", in *Proceedings of 2003 IEEE International Conference on Control Applications (CCA)*, vol. 1. june 2003, pp. 1469–1476.

[Shi 94] J. Shi, C. Tomasi, "Good Features to Track", in *IEEE Conference on Computer Vision and Pattern Recognition*. June 1994, pp. 593–600.

[Shneier 06] M. Shneier, W. Shackleford, T. Hong, T. Chang, "Performance Evaluation of a Terrain Traversability Learning Algorithm in the DARPA LAGR Program", *NIST Workshop on Performance Metrics for Intelligent Systems*, August 2006.

[Shneier 08] M. Shneier, T. Chang, T. Hong, W. Shackleford, R. Bostelman, J. S. Albus, "Learning traversability models for autonomous mobile vehicles", *Autonomous Robots*, vol. 24, no. 1, pp. 69–86, january 2008.

[Simmons 97] R. Simmons, R. Goodwin, K. Haigh, S. Koenig, J. OŚullivan, M. Veloso, "Xavier: Experience with a Layered Robot Architecture", in *Agents ́97*. 1997.

[Simond 06] N. Simond, "Reconstruction of the road plane with an embedded stereo-rig in urban environments", *Intelligent Vehicles Symposium, 2006 IEEE*, pp. 70–75, 2006.

[Singh 02] M. Singh, S. Singh, "Spatial Texture Analysis: A Comparative Study", in *ICPR (1)*. 2002, pp. 676–679.

[Singh 99] S. Singh, B. Digney, "Autonomous Cross-Country Navigation Using Stereo Vision", Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-99-03, January 1999.

[Smith 90] R. Smith, M. Self, P. Cheeseman, "Estimating uncertain spatial relationships in robotics", *Autonomous robot vehicles*, pp. 167–193, 1990.

[Sofman 06] B. Sofman, E. Lin, J. A. Bagnell, J. Cole, N. Vandapel, A. Stentz, "Improving Robot Navigation Through Self-Supervised Online Learning", *Journal of Field Robotics*, vol. 23, no. 12, pp. 1059–1075, December 2006.

[Soltani 02] A. R. Soltani, H. Tawfik, J. Y. Goulermas, T. Fernando, "Path planning in construction sites: performance evaluation of the Dijkstra, A*, and GA search algorithms", *Advanced Engineering Informatics*, vol. 16, no. 4, pp. 291–303, 2002.

[Steck 99] S. D. Steck, H. A. Mallot, "Different strategies of global and local landmark usage in virtual environment navigation", *Journal of Cognitive Neuroscience*, pp. 76–77, 1999.

[Sugihara 88] K. Sugihara, "Some location problems for robot navigation using a single camera", *Comput. Vision Graph. Image Process.*, vol. 42, no. 1, pp. 112–129, 1988.

[Sun 06] J. Sun, T. Mehta, D. Wooden, M. Powers, J. Regh, T. Balch, M. Egerstedt, "Learning from Examples in Unstructured, Outdoor Environments", *Journal of Field Robotics*, vol. 23, no. 11/12, pp. 1019–1036, 2006.

[Takala 05] V. Takala, T. Ahonen, M. Pietikainen, "Block-Based Methods for Image Retrieval Using Local Binary Patterns", in *Scandinavian Conference on Image Analysis*. 2005, pp. 882–891.

[Talukder 02] A. Talukder, R. Manduchi, A. Rankin, l Matthies, "Fast and Reliable Obstacle Detection and Segmentation for cross-country Navigation", in *IEEE Intelligent Vehicles Symposium*. June 2002, pp. 610–618.

[Tapus 05] A. Tapus, R. Siegwart, "Incremental Robot Mapping with Fingerprints of Places", in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. August 2005, pp. 2429– 2434.

[Tardos 02] J. D. Tardos, J. Neira, P. M. Newman, J. J. Leonard, "Robust Mapping and Localization in Indoor Environments using Sonar Data", *International Journal of Robotic Research*, vol. 21, no. 4, pp. 311–330, 2002.

[Thrun 01] S. Thrun, D. Fox, W. Burgard, F. Dellaert, "Robust Monte Carlo localization for mobile robots", *Artificial Intelligence*, vol. 128, no. 1–2, pp. 99–141, 2001.

[Thrun 02] S. Thrun, "Robotic Mapping: A Survey", CMU, Tech. Rep., Feb. 2002. CMU-CS-02-111.

[Thrun 06] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann et al, "Winning the DARPA Grand Challenge", *Journal o Field Robotics*, 2006.

[Thrun 96a] S. Thrun, "Is Learning the n-th Thing Any Easier Than Learning The First?", in *Advances Neural Information Processing Systems 8*. MIT Press, 1996, pp. 640–646.

[Thrun 96b] S. Thrun, A. Buecken, W. Burgard, D. Fox, T. Froehlinghaus, D. Hennig, T. Hofmann, M. Krell, T. Schmidt, "Map Learning and High-Speed Navigation in RHINO", University of Bonn, Tech. Rep. IAI-TR-96-3, 1996.

[Thrun 98a] S. Thrun, "Learning Metric-Topological Maps for Indoor Mobile Robot Navigation", in *Artificial Intelligence*, ser. 1, vol. 99. 1998, pp. 21–71.

[Thrun 98b] S. Thrun, W. Burgard, D. Fox, "A probabilistic approach to concurrent mapping and localization for mobile robots", in *Machine Learning*, vol. 31, no. 1. 1998, pp. 29–53.

[Tomasi 91] C. Tomasi, T. Kanade, "Detection and Tracking of Point Features", Carnegie Mellon University, Tech. Rep. CMU-CS-91-132, April 1991.

[Tomatis 03] N. Tomatis, I. R. Nourbakhsh, R. Siegwart, "Hybrid simultaneous localization and map building: a natural integration of topological and metric", *Robotics and Autonomous Systems*, vol. 44, no. 1, pp. 3–14, 2003.

[Triebel 06] R. Triebel, P. Pfaff, W. Burgard, "Multi Level Surface Maps for Outdoor Terrain Mapping and Loop Closing", *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2006.

[Tuceryan 98] M. Tuceryan, A. K. Jain, "Texture Analysis", in *Handbook of Pattern Recognition and Computer Vision*, P. S. P. W. C H Chen, L F Pau, Ed. World Scientific Publishing Co., 1998, pp. 207–248.

[Vale 05] A. Vale, "Mobile Robot Navigation in Outdoor Environments: A Topological Approach", Dissertation, Institute for Systems and Robotics, Lisboa, 2005.

[Vandapel 04] N. Vandapel, D. Huber, A. Kapuria, M. Hebert, "Natural Terrain Classification using 3-D Ladar Data", in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 5. May 2004, pp. 5117– 5122.

[Vergauwen 01] M. Vergauwen, M. Pollefeys, R. Moreas, F. Xu, G. Visentin, L. V. Gool, H. V. Brussel, "Calibration, Terrain Reconstruction and Path Planning for a Planetary Micro-Rover", in *i-SAIRAS 6 th International Symposium on Artificial Intelligence, Robotics and Automation in Space*. 2001, pp. 18–22.

[Wehner 03] R. Wehner, "Desert ant navigation: how miniature brains solve complex tasks", *JOURNAL OF COMPARATIVE PHYSIOLOGY A*, vol. 189, no. 8, pp. 579–588, 2003.

[Weingarten 03] J. Weingarten, G. Gruener, R. Siegwart, "A Fast and Robust 3D Feature Extraction Algorithm for Structured Environment Reconstruction", in *Proceedings of the 11th International Conference on Advanced Robotics (ICAR)*. July 2003.

[Weingarten 04] J. Weingarten, G. Gruener, R. Siegwart, "Probabilistic Plane Fitting in 3D and an Application to Robotic Mapping", in *Proceedings of the International Conference on Robotics and Automation (ICRA)*. 2004.

[Weng 92] J. Weng, P. Cohen, M. Herniou, "Camera calibration with distortion models and accuracy evaluation", *Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 10, pp. 965–980, October 1992.

[Wettach 07] J. Wettach, K. Berns, "3D Reconstruction for Exploration of Indoor Environments", in *Autonome Mobile Systeme 2007*, ser. Informatik aktuell, K. Berns, T. Luksch, Eds. Kaiserslautern, October 18–19 2007, pp. 57–63.

[Yairi 02] T. Yairi, M. Togami, K. Hori, "Learning Topological Maps from Sequential Observation and Action Data under Partially Observable Environment", *j-LECT-NOTES-COMP-SCI*, vol. 2417, p. 305pp., 2002.

[Yamauchi 96] B. Yamauchi, R. Beer, "Spatial learning for navigation in dynamic environments", *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 26, no. 3, pp. 496 – 505, 1996.

[Ye 04] C. Ye, J. Borenstein, "A Method for Mobile Robot Navigation on Rough Terrain.", in *IEEE International Conference on Robotics and Automation (ICRA)*. New Orleans, LA, USA: IEEE, April 2004, pp. 3863–3869.

[Yguel 06] M. Yguel, O. Aycard, C. Laugier, "Efficient GPU-based Construction of Occupancy Grids Using several Laser Range-finders", in *Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*. 2006.

[Zimmer 96] U. R. Zimmer, "Robust world-modelling and navigation in a real world", *Neurocomputing*, vol. 13, no. 2, pp. 247–260, 1996.

[Zitzler 99] E. Zitzler, L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach", *IEEE Trans. Evolutionary Computation*, vol. 3, no. 4, pp. 257–271, 1999.

[Zolynski 07] G. Zolynski, "GPULBP: Implementing the Local Binary Pattern Operator on Graphics Processing Hardware", Project thesis, unpublished, Robotics Research Lab - University of Kaiserslautern, Oct 2007.

[Zolynski 08] G. Zolynski, **T. Braun**, K. Berns, "Local Binary Pattern Based Texture Analysis in Real Time using a Graphics Processing Unit (long version)", in *Proceedings of Robotik 2008*, ser. VDI-Berichte, vol. 2012. VDI Wissensforum GmbH, June 2008. Extended CD-ROM Version, ISBN 978-3-18-092012-2.

[Zwynsvoorde 00] D. V. Zwynsvoorde, T. Simeon, R. Alami, "Incremental topological modeling using local Voronoi-like graphs", in *Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 2. 2000, pp. 897–902.