Panoramic Image Processing and Route Navigation

Vom Fachbereich Mathematik und Informatik
der Universität Bremen

zur Verleihung des akademischen Grades eines
Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigte Dissertation
von

Thomas Röfer

Gutachter:	Prof. Dr. Bernd Krieg-Brückner
	Prof. Dr. Gerhard Manteuffel
Kolloquium:	7. Januar 1998

Editorial

Thomas Röfer was a scholar in the Graduate College "Spatial Orientation and Organization of Action in Autonomous Systems", an interdisciplinary cooperation of neurobiologists, neurophysicists, computer scientists and engineers funded by DFG (German Research Organization). In this context he chose a thesis topic from Cognitive Robotics. His work lead to an intensive cooperation with psychologists in the context of the DFG Focus Program "Spatial Cognition".

Cognitive Robotics shall realize abstract models, inspired by nature, in technical implementations and thus validate them experimentally. Accordingly Thomas Röfer's thesis emphasizes mechanisms for navigation modeled after the behavior of animals (e.g. honeybees, ants, birds, rats) and humans. This approach is interesting for service robotics, because biologically motivated methods are supposed to be most effective. In contrast to conventional industrial robotics, where high precision is imperative, service robotics is concerned with robustness, adaptivity to changes of the environment, and reaction to dynamic obstacles, e.g. different lighting conditions, relocated furniture, moving people. Thus there are minimal assumptions about the environment, especially no precisely measured metrical maps; as far as possible, simple sensors are used that complement each other to increase robustness.

In his dissertation, Thomas Röfer treats the navigation with local landmarks, in particular an image processing and navigation approach that has been studied primarily in social insects and birds. He models and extends this approach, uses it on two autonomous mobile systems in simulation and experiments, and thus demonstrates its robustness and practical use. Moreover, he realizes a complex tactical navigation algorithm for routes with artificial landmarks.

Thomas Röfer has submitted a technically very demanding thesis that solves some theoretical and practical tasks in scene-based dynamic vision in a simple and elegant way; it also impresses by its interdisciplinary approach and the comprehensive treatment of its hardware and software realization. In particular, his contribution to the Bremen Autonomous Wheelchair is to be appreciated that has been successful in the DFG Focus Program "Spatial Cognition". On the basis of the work by Thomas Röfer and others we expect that the second prototype of the Bremen Autonomous Wheelchair, currently under development, will soon become a marketable product in cooperation with Meyra, the German market leader for wheelchairs.

May 1998 Prof. Dr. Bernd Krieg-Brückner

Vorwort des Herausgebers

Thomas Röfer hat sich als Stipendiat im Rahmen des DFG Graduiertenkollegs „Raumorientierung und Handlungsorganisation Autonomer Systeme“, einer interdisziplinären Kooperation von Neurobiologen und Neurophysikern einerseits sowie Informatikern und Ingenieuren andererseits, ein Thema aus dem Bereich der Kognitiven Robotik gewählt. Im weiteren Verlauf der Arbeit kam es zu intensiven Kooperationen mit Psychologen im Rahmen des DFG Schwerpunktprogramms „Raumkognition“.

Die Kognitive Robotik soll abstrakte, an der Natur ausgerichtete Modellierungen konkret in technischen Verfahren realisieren und sie so experimentell validieren. Dementsprechend stehen in der Dissertation von Thomas Röfer Mechanismen der Navigation im Vordergrund, die sich an den Verhaltensweisen von Tieren (z.B. Honigbienen, Ameisen, Vögeln, Ratten) und Menschen orientieren. Dieser Ansatz ist für die Servicerobotik besonders interessant, da vermutet wird, daß die biologisch motivierten Verfahren in diesem Bereich besonders erfolgreich sein könnten. Denn im Gegensatz zur herkömmlichen industriellen Robotik, wo höchste Präzision unabdingbar ist, kommt es in der Servicerobotik auf Robustheit, Adaptivität an räumliche Veränderungen und Reaktionsfähigkeit auf dynamische Hindernisse an, z.B. verschiedenartige Beleuchtungsbedingungen, verschobene Möbelstücke, Passanten. Biologisch motivierte Navigationsverfahren machen daher nur minimale Annahmen über die Umwelt, insbesondere benutzen sie keine präzise vermessenen, metrischen Karten; es wird mit möglichst einfachen Sensoren gearbeitet, die sich aber zur Erhöhung der Robustheit gegenseitig ergänzen sollen.

In seiner Dissertation behandelt Thomas Röfer die Navigation mit lokalen Landmarken, vor allem ein Bildverarbeitungs- und Navigationsverfahren, das besonders an sozialen Insekten und Vögeln studiert wurde. Er modelliert und erweitert dieses Verfahren, setzt es auf zwei verschiedenartigen Autonomen Mobilen Systemen in der Simulation und im Experiment ein und demonstriert so die Robustheit und praktische Einsatzfähigkeit. Ferner realisiert er ein komplexes taktisches Navigationsverfahren für Routen mit künstlichen Landmarken.

Thomas Röfer hat eine technisch äußerst anspruchsvolle Arbeit vorgelegt, die nicht nur einige theoretische Aufgaben der szenenbasierten Bildfolgenverarbeitung elegant und einfach löst, sondern vor allem durch ihren interdisziplinären Ansatz und den breiten Umfang der hardware- und software-technischen Realisierung besticht. Dabei ist insbesondere der Beitrag zum Bremer Autonomen Rollstuhl zu würdigen, mit dem das Bremer Teilprojekt im Schwerpunktprogramm „Raumkognition“ erfolgreich war. Auf der Basis der Arbeiten von Herrn Röfer und anderen ist damit zu rechnen, daß der zweite in Arbeit befindliche Prototyp des Bremer Autonomen Rollstuhls in Zusammenarbeit mit der Firma Meyra, dem deutschen Marktführer für Rollstühle, bald zu einem marktfähigen Produkt wird.

Mai 1998 — Prof. Dr. Bernd Krieg-Brückner

Summary

The area of autonomous mobile robots will play an increasing role in our daily life in the future. Especially in the field of service robotics and rehabilitation robotics, large progress is expected. In contrast to previous systems that only move in specially adapted surroundings, this new generation of robots should be used in an environment that was designed for people's needs and in which special changes for the use of such systems are either impossible or undesirable. Hence, service robots must be able to find their way in a human-oriented, natural environment, i.e., they have to navigate in it.

In this thesis, an image processing method is presented that enables a robot to orientate in unchanged, existing environments. The approach is based on the use of one-dimensional 360° (panoramic) color images that are taken by a special optical sensor. From two of these images, the *Panama* algorithm that is presented in this thesis determines the spatial relationship between the positions where the images have been taken. The approach can be compared to methods that determine the optical motion flow but it addresses a different application: it is used to determine the spatial relations between positions that may be located several meters apart.

This image processing method is embedded in a navigation technique in which the environment is represented as a network of routes along which the navigating agent can move. These routes are represented as sequences of panoramic images. This route knowledge is acquired by teaching. A teacher only presets the routes that should be learned but not the images that are required for their description. These images are independently selected by the autonomous system during the training, depending on the particular environment and the system's kinematic restrictions.

The navigation method was implemented and tested on a simulated as well as on a real system. The real mobile system was an electric wheelchair—with the application rehabilitation robotics in mind.

In addition to the navigation method, an approach for the use of the introduced image processing method for metrical self-localization is presented and its precision is analyzed.

Furthermore, a second navigation method is presented that describes routes as sequences of basic behaviors. The recognition of routemark constellations triggers the switching between these behaviors. The method detects possible navigational errors and tries to recover from them by backtracking the previous way.

Zusammenfassung

Der Bereich der autonomen mobilen Roboter wird in unserem täglichen Leben in Zukunft eine immer größere Rolle spielen. Insbesondere in den Gebieten der Servicerobotik und Rehabilitationsrobotik werden große Fortschritte erwartet. Im Gegensatz zu den bisherigen Systemen, die sich in speziell für sie angepaßten Umgebungen bewegen, soll diese neue Generation von Robotern in einer für den Menschen geschaffenen Umwelt eingesetzt werden, in der spezielle Umgestaltungen für den Einsatz solcher Systeme entweder nicht möglich oder nicht wünschenswert sind. Daher müssen Serviceroboter in der Lage sein, sich in einer für den Menschen gestalteten, natürlichen Umwelt zurechtzufinden, d.h. in ihr zu navigieren.

In dieser Dissertation wird ein Bildverarbeitungsverfahren vorgestellt, das eine solche Orientierung mit Hilfe der vorhandenen Umwelt ermöglicht. Das Verfahren basiert auf der Verwendung eindimensionaler, farbiger 360°-Panoramabilder, die mit einem speziellen optischen Sensor aufgenommen werden. Aus jeweils zwei dieser Bilder wird von dem in dieser Arbeit vorgestellten *Panama*-Algorithmus die räumliche Beziehung zwischen den Positionen ermittelt, an denen die Bilder aufgenommen wurden. Das Verfahren kann mit Methoden zur Bestimmung des optischen Bewegungsflusses verglichen werden, verfolgt aber eine andere Zielrichtung: es dient der Bestimmung räumlicher Beziehungen zwischen Positionen, die möglicherweise mehrere Meter voneinander entfernt sind.

Diese Bildverarbeitungsmethode wird in ein Navigationsverfahren eingebettet, in dem die Umwelt als ein Netzwerk von Routen repräsentiert wird, auf denen sich das navigierende System bewegen kann. Die Routen werden durch Folgen von Panoramabildern repräsentiert. Dieses Routenwissen wird durch *Teaching* erworben, wobei ein Lehrer lediglich die zu lernenden Routen vorgibt, nicht aber die zu deren Beschreibung notwendigen Bilder. Diese ermittelt das autonome System während des Trainings selbständig, abhängig von der jeweiligen Umgebung und seinen eigenen kinematischen Beschränkungen.

Das Navigationsverfahren wurde sowohl auf einem simulierten als auch auf einem realen System implementiert und getestet. Bei dem realen System handelt es sich – mit dem Anwendungsgebiet Rehabilitationsrobotik – um einen elektrisch angetriebenen Rollstuhl.

Zusätzlich zu dem Navigationsverfahren wird eine Methode zur Nutzung des vorgestellten Bildverarbeitungsverfahrens zur metrischen Selbstlokalisation vorgestellt und ihre Genauigkeit untersucht.

Des weiteren wird ein zweites Navigationsverfahren vorgestellt, das Routen als Folgen von Grundverhalten beschreibt. Das Weiterschalten zwischen den Grundverhalten wird durch das Wiedererkennen von Wegmarkenkonstellationen ausgelöst. Das Verfahren erkennt mögliche Navigationsfehler und versucht, diese durch Zurückverfolgen des bisherigen Weges zu beheben.

Contents

Acknowledgements

The work presented in this thesis was embedded in the graduate program "Raumorientierung und Handlungsorganisation autonomer Systeme" (spatial orientation and organization of action in autonomous systems) funded by the Deutsche Forschungsgemeinschaft and the Freie Hansestadt Bremen. In addition, during the last year of working on this thesis, the author was employed in the priority program "Raumkognition" (spatial cognition) also funded by the Deutsche Forschungsgemeinschaft.

I want to thank my advisor Prof. Dr. Bernd Krieg-Brückner for giving me the chance to participate in the graduate program and to work in the priority program. He also initiated the student projects "FORANN – FORmalisierung und Anwendung Neuronaler Netze" (formalization and application of neural networks) and "SAUS – Sensomotorik AUtonomer Systeme" (sensorimotorics of autonomous systems) from which I profited very much.

Many thanks to my second advisor Prof. Dr. Gerhard Manteuffel and Dr. Dorothea Brückner for their inspirations to this thesis from a biological point of view.

In addition, I want to thank Rait Harnett for improving my English in some of my papers and Uwe Siems who wrote the simulator software SimRobot that was a useful tool for the development of the methods presented in this thesis. Also thanks to all students of the project SAUS that developed and constructed the autonomous wheelchair that was used for several experiments.

Furthermore, I wish to thank all colleages within the graduate program, especially Dr. Andreas Bühlmeier, Dr. Christoph Herwig, Dr. Matthias Heger, and Holger Dürer for their commitment to the SAUS project and, together with Dr. Thomas Wittmann and my colleague in the "spatial cognition" project Rolf Müller, for fruitful discussions.

Last but not least, I want to thank my family for their support and encouragement, especially my wife Gabi, who, in addition, proof-read this work although she has been working on her own Ph. D. I also wish to thank my son Adrian although he contributed only very little to this thesis (cf. figure 3.5a on page 75), simply for being there.

Chapter 1

Introduction

The navigation of autonomous mobile systems (robots) is still an unsolved problem. They operate in a three-dimensional and changing world. In this environment, they have to navigate in real time. Therefore, there has to be a suitable technique for navigation. Methods that are based on three-dimensional models (Rembold 1988, Munkelt *et al.* 1993, Lanser *et al.* 1994, Lin and Dillmann 1995, Burschka *et al.* 1995) are very complex: it takes a lot of time to match the sensory data with three-dimensional objects stored in large databases. Thus, it makes sense to search for methods that are less complicated. One possibility is to follow biological or psychological models because even very "simple" animals are able to navigate quite accurately.

Cotter (1993) defines *navigation* as follows:

> Navigation is the art and science of conducting a ship, airplane, or spacecraft safely and expeditiously to a specific destination. In the broadest sense anyone faced with the problem of "finding the way" is a navigator, whether he or she travels on land, at sea, in the air, or in space.

Following this definition, navigation means to find and to cover a way from a start position to a target position. It is unimportant whether the navigating agent is e.g. a seaman, a pilot, a bird, an insect, or an artificial autonomous mobile system. It is always the sensory equipment and the movability of the agent that determine the kind of navigation that it is able to perform (Krieg-Brückner *et al.* 1998).

In robotics, one can distinguish between global and local navigation. The global variant corresponds to the aforementioned definition. Prescott (1994) calls this kind of navigation the *way-finding* skill. The local navigation describes the adaptation of the global navigation's instructions to the agent's kinematical restrictions, e.g., a walking machine would locally navigate in a completely different way than a vehicle. The global navigation does only deal with the kinematical restrictions of the navigating agent in so far as they limit its general movability, e.g., a flying agent is able to use other ways than a swimming one.

In the sequel, global as well as local navigation will be discussed.

1.1 Global Navigation

The global navigation strategies are predetermined by the locomotion capabilities of each individual navigating agent, e.g., a bird can fly straight over very long distances whereas most other animals and humans have to go around many obstacles to reach a certain target. Although most creatures navigate in three-dimensional space, this chapter will only analyze two-dimensional navigation.

Many different navigation techniques are used by insects, animals, and humans. Several classifications were developed to structure this field into different categories. In the following, three of these classifications will be briefly described.

1.1.1 The Classification of Trullier *et al.*

Trullier *et al.* (1997) compiled an exhaustive review of work concerning the area of natural and artificial navigation. They classified navigation into the four categories *guidance*, *place recognition-triggered response*, *topological navigation*, and *metrical navigation*:

Guidance is the leading of an agent to a certain target position by external cues. The cues can either be gradients of a certain quality, the image of the target, or the image of the surroundings. The latter two variants are also called *piloting* (Gallistel 1990). Examples for gradient-based guidance are the olfactory navigation in bees, the mental *grid map* in birds (Wiltschko 1997), and the scent-tracing in ants. The image of the surroundings or the bearings of salient landmarks are used in the homing behaviors of honeybees (Collet *et al.* 1986, Cartwright and Collet 1982, 1983, 1987, Wittmann 1996) and in bird navigation as a *mosaic map* (Graue 1963). Honeybees can also be guided by sequences of landmarks and are assumed to be able to *protocount* (Chittka and Geiger 1995). Rats use landmarks near the target (beacons) for piloting (Leonard and McNaughton 1990).

In robotics, guidance has been used in several navigation approaches. For example, Weisbin *et al.* (1989) developed the *HERMIES-IIB* robot that gets close to a target by dead reckoning and is then visually guided to its final position. In the approach of Balkenius and Kopp (1997), a robot selects a set of visual landmarks during a training phase and is afterwards able to follow the trained trajectory autonomously—guided by the trained landmarks. In contrast, Hong *et al.* (1991) used complete one-dimensional panoramic views of the surroundings to guide a mobile system through a corridor. A similar approach has been used by Franz *et al.* (1997a) that imitates the homing behavior of honeybees, i.e. the homing is not performed on a straight line. Another application for guidance is autonomous parking. Fäustle *et al.* (1994) realized a parking vehicle that is guided by a white frame around the parking-space, whereas the *NEUROPA*-system is guided by a parking sign (Oberdorfer *et al.* 1995). Much research effort has been invested in systems that are guided by roadsides. Whereas the system developed by Ishikawa *et al.* (1988) is only able to follow a white line on the floor, the vehicles of Turk *et al.* (1988), Thorpe *et al.* (1988), and Sharma and Davis (1988) are able to track the borders of real streets and use this information for route-guidance.

Place Recognition-Triggered Response is the combination of the recognition of places and the carrying out of actions. An agent can learn a certain navigation task by associating locations with movement actions. In robotics, several navigation methods have been developed that try to implement this combination. Most of them differ in one important aspect: instead of associating *places* with actions, they associate *perceptions* to actions, e.g. in the implementations of Brooks (1990), Nehmzow (1992), Owen and Nehmzow (1997), and Maes (1991, 1992). Thus, they are not able to distinguish between different places that look similar. Duckett and Nehmzow (1997) called this the problem of *perceptual aliasing*. Kuipers and Byun (1991) tried to get around this problem by generating *distinctive places* as parts of *distinctive routes* in a simulated environment. Duckett and Nehmzow (1997) followed a similar but more sophisticated approach because they used a real robot, and therefore they had to deal with sensory noise and uncertainty. In natural systems, place recognition seems to be a solved problem, e.g. in the brain of rats, specialized cells (place-cells in the hippocampus) are activated if the animal is at a certain location (O'Keefe and Dostrovsky 1971, O'Keefe and Conway 1978, O'Keefe 1979). Inspired by these findings, Mataric (1991) developed a model for robot navigation. However, the model was only tested in a simulation.

The combination of (sensory) states and associated actions is also the basis of the *reinforcement learning* theory (Minsky 1961). With Q-learning (Watkins 1989), the agent learns the actions that maximize the success of the agent's behaviors in all further states—either the expected mean success (Watkins 1989, Barto *et al.* 1983, Sutton 1991) or the guaranteed minimal success (Heger 1996). An extension of this concept is *hierarchical reinforcement learning* (Lin 1993). In this approach, the task is distributed over several hierarchical levels. On the lowest level, the sensorimotoric linkages for elementary behaviors are learned, e.g. wall-following and obstacle avoidance. All further levels do not directly connect the sensory inputs with the actuators. Instead, on these levels, the sensory input is associated with behaviors of the next lower level. However, the approach requires a lot of preparatory work because all different levels must be trained in succession (Dunekamp and Meyer 1998).

Another problem of reinforcement learning is the determination of *states*. As sensor readings normally consist of real values, they have to be discretized to a few states. In some situations, this might destroy the relevant information. Hence, Moore (1993) and Moore and Atkeson (1995) developed the *Parti-Game* algorithm that dynamically divides the state-space. Some improvements of this approach have been made by Heger (1996) in his *Parti-Game 2* algorithm. This method was also applied by Kollmann *et al.* (1997) to the wheelchair that was used for some of the experiments that will be presented in chapter 3.

Topological Navigation is the navigation in topological networks. Such networks consist of connected routes, and hence form *route maps* (Mecklenbräuker *et al.* 1997). An agent can navigate between arbitrary *vertices*, i.e. places, in these topological structures but it is neither able to reach places that are not in the map nor can it travel routes that are not part of the topological network. Planning in topological structures is easier than in a metrical environment (Gutmann and Nebel 1997). Several approaches were used in robotics to generate topological descriptions of the environment. For example, Nehmzow and Smithers (1991, 1992) developed a vehicle that builds a circular map on the basis of a detection of convex and concave corners. In the approach of Dudek *et al.* (1997), a robot builds a topological map between metrical sub-maps. Hence, the system is able to navigate with metrical precision in each of the subareas but does not have to maintain a global metrical position. The transitions between the subareas are performed at unique positions that can be compared to the *distinctive places* of Kuipers and Byun (1991). A similar approach was followed by Giralt *et al.* (1984). They used a topological map for navigation and enriched it by information about the distances between the places in the map and the boundary dimensions of these places.

Another kind of topological networks are *view graphs* (Schölkopf and Mallot 1995, Franz *et al.* 1997b). In these graphs, each vertex is not a place. Instead, it is a view, i.e. a position in combination with a viewing direction. In such graphs, route descriptions consist in associations between views and actions. Schölkopf and Mallot (1995) also showed that view graphs can be transformed into *place graphs* in which vertices represent places and edges stand for the routes between these places.

In contrast to Kuipers and Byun (1991) who developed an approach that first generates a topological map in order to achieve a metrical map afterwards, Gutmann and Nebel (1997) used a method that first constructs a metrical representation of the environment from laser-scans, and then their method uses the scan-points in the map to generate a topological representation. After that, the topological representation is employed for navigation.

Metrical Navigation is the navigation based on the use of metrical information. In animals and insects, there seems to be only one kind of metrical navigation, i.e. dead reckoning. In addition, some animals can acquire a metrical representation of their visual surroundings (Trullier *et al.* 1997). In contrast, in robotics, many navigation approaches are based on metrical information. Much effort has been invested in either employing predefined maps or in autonomously generating new ones. As mentioned above, Kuipers and Byun (1991) developed a method that generates a metrical representation of the environment on the basis of a topological description acquired earlier (cf. section 1.1.2). A very popular kind of metrical world representations is *occupancy grids*. For example, Elfes (1991) and Matthies and Elfes (1988) developed a method for metrical mapping and self-localization on the basis of a probabilistic model of the measurements of the utilized sensors. They also showed how a preexisting map can be integrated in their model. Burgard *et al.* (1997) implemented a similar approach that was used on the robot *RHINO* employed a as museum guide in the museum for German history for three days. During this time, it guided people to 1200 exhibits and was always able to track its position although the rooms were crowded with people.

Mapping was also successfully performed using range-sensors, e.g. laser range finders (Edlinger and Weiß 1995, Kirchner 1997) and radar-sensors (Rozmann and Detlefsen 1995). For walking machines, three-dimensional mapping is required because if obstacles are not to high, the machine can climb over them. Hence, Hebert *et al.* (1988) and Bares *et al.* (1989) developed a 3-D vision mapping method for their hexapod *Ambler*. Asade (1991) presented an algorithm to convert range images into height maps.

Another metrical mapping technique is the *Parti-Game* algorithm (Moore 1993, Moore and Atkeson 1995, Heger 1996, Kollmann *et al.* 1997). This method relies on a preexisting metrical self-localization mechanism. On the basis of reinforcement learning, it builds a grid-based representation of the environment with an adaptive resolution. In this representation, it is not distinguished between free space and occupied space, instead, it is only stored whether transitions between grid-cells are possible for the navigating agent. Nevertheless, as this approach requires a localization mechanism, it may sometimes be obsolete if the self-localization system also generates a metrical representation of the surroundings.

1.1.2 The Spatial Semantic Hierarchy

Kuipers (1996), Kuipers *et al.* (1993), and Kuipers and Byun (1988, 1991) analyzed the task of navigation. In contrast to Trullier *et al.* (1997), they structured their classification as a hierarchy. Their *Spatial Semantic Hierarchy* consists of the five levels *sensorimotor system*, *control level*, *causal level*, *topological level* and *metrical level*. They assumed that navigating agents will use some of these levels but not necessarily all of them.

Sensorimotor System. On the first level of the hierarchy, the sensorimotor system of the robot provides continuous sensors and effectors, but no direct access to the global structure of the environment, or the robot's position or orientation within it.

Control Level. The second level of the spatial semantic hierarchy is called *control level*. At this level, egocentric sensorimotor information is represented, without knowledge of fixed objects or places in an external environment. Kuipers (1996) defined *distinctive states* as the local maximum found by a hill-climbing control strategy, climbing the gradient of a selected sensory feature, or distinctiveness measure. Lee (1996) realized this on a real robot with twelve sonar sensors. He divided this strategy in two steps: first, the robot hill-climbs to a distinctive location, and then it uses another strategy to reach a distinctive orientation. The second part of the control level consists in some trajectory-following control laws that take the robot from one distinctive state to the neighborhood of the next, where again hill-climbing can find a local maximum, reducing the position error and preventing its accumulation. Some trajectory-following strategies have been implemented by Lee (1996) but also by others, e.g. by Duckett and Nehmzow (1997). In chapter 5, a route-following approach for an autonomous wheelchair will be described.

Causal Level. The causal level of the hierarchy consists of views, which describe the sensory images at distinctive states and actions that represent sequences of control laws by which the robot moves from one view to another. A causal graph of associations among views, actions, and resulting views represents both declarative and imperative knowledge of routes or action procedures. The representation of such route knowledge for a wheelchair will also be described in chapter 5. In this approach, the distinctive states are represented by *routemark constellations*. These constellations do not require a hill-climbing behavior. They were chosen because hill-climbing is hard to realize for vehicles with normal car-kinematics.

Topological Level. Kuipers (1996) defined this level as follows:

> At the topological level of the hierarchy, the ontology consists of places, paths, and regions, with connectivity and containment relations. Relations among the distinctive states and trajectories defined by the control level, and among their summaries as views and actions at the causal level, are effectively described by the topological network. This network can be used to guide exploration of new environments and to solve new route-finding problems. Using the network representation, navigation among distinctive states is not dependent on the accuracy, or even the existence, of metrical knowledge of the environment.

Hence, the topological level of the Spatial Semantic Hierarchy is similar to the *topological navigation* in the classification of Trullier *et al.* (1997) that has been discussed in the previous section.

Metrical Level. At the metrical level of the hierarchy, the topological level is extended to include metrical properties such as distance, direction, shape, etc. Geometrical features are extracted from sensory input, and represented as annotations on the places and paths of the topological network. Kuipers (1996) presented a three step approach for this purpose: first, his agent acquires so-called local 1-D geometric information, i.e. information about the distances between adjacent places in the topological network and angles between the connections of these places. Next, it builds a representation of the surroundings of the paths and places. In a third and last step, these representations are combined to a single global metrical map. Kuipers (1996) emphasized that the generation of this metrical information is neither necessary for exploration, map-learning, or route-learning, nor for navigation in his navigation approach.

As criticism, it can be asked why the metrical level is the highest level in the Spatial Semantic Hierarchy although metrical navigation techniques, e.g. dead reckoning (Wittmann 1995), can be found even in very simple living beings, e.g. in desert ants (Müller and Wehner 1988) and gerbils (Mittelstaedt and Mittelstaedt 1982), but also in humans (Gallistel 1990).

1.1.3 The Taxonomy of Navigation

As a third categorization, Krieg-Brückner *et al.* (1998) presented the *Taxonomy of Navigation* (cf. also Werner *et al.* 1997, Schmid *et al.* 1997). This classification is more abstract than the other two that have already been discussed. They also introduced the *Spatial Cognition Triangle* (cf. figure 1.1). This triangle visualizes that spatial knowledge, its representation, and its acquisition are intimately related with perceptual information on the one hand, and intended navigational behavior on the other. As a major aspect of the taxonomy, navigation in open or enclosed space and navigation in networks of passages are distinguished. It also differentiates between *landmarks* and *routemarks*. Constellations of landmarks are used as *location designators* in open or enclosed space, i.e. 2-D or 3-D space, whereas routemarks are utilized to determine a position along routes, i.e. 1-D space.

The taxonomy structures spatial knowledge about an environment into location, route, and survey knowledge (cf. figure 1.2). A location is often characterized by a view of its surroundings (Schweizer and Janzen 1996, Franz *et al.* 1997a, Schölkopf and Mallot 1995), also sometimes called a *snapshot* (Cartwright and Collet 1983). These views are described as consisting of a constellation of landmarks (Collet and Kelber 1988) or

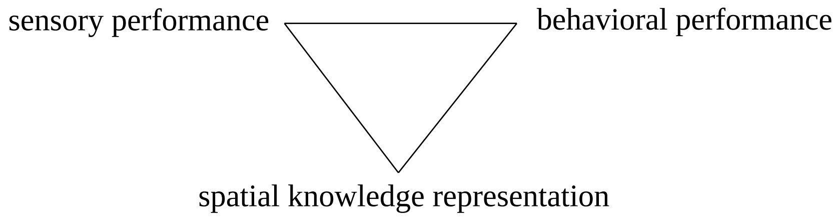

Figure 1.1: The Spatial Cognition Triangle (Krieg-Brückner *et al.* 1998).

cues (Poucet 1993) that identify the location. A route is often identified with a sequence of locations or views (Schweizer and Janzen 1996). According to Krieg-Brückner *et al.* (1998), survey knowledge is obtained as an abstraction and integration of specific routes. The mental representation of survey knowledge is referred to as *cognitive map* (Tolman 1948, Scholl 1987). For example, in insect research, there was a long discussion whether insects such as honeybees have a cognitive map (Gould 1986) or not (Wehner and Menzel 1990, Dyer 1991).

The Taxonomy of Navigation consists of the four levels *basic behaviors*, *elementary navigation tactics*, *tactical navigation*, and *strategic navigation* (cf. figure 1.2):

Basic Behaviors. Krieg-Brückner *et al.* (1998) describe basic behaviors as atomic tactics, the simplest to be investigated—or realized in a robot. As examples, they mention wall-following while avoiding obstacles (Nehmzow 1995) in tunnels, corridors, roads, or trails, and turning into designated passages at junctions. Basic behaviors should be used in the more complex elementary navigation tactics.

Elementary Navigation Tactics. Most of the behaviors that were described as *guidance* or *place recognition-triggered response* by Trullier *et al.* (1997) belong to this level. As elementary tactics in space, Krieg-Brückner *et al.* (1998) mentioned directional navigation guided by a compass (Wiltschko and Wiltschko 1995, Collet and Baron 1994), dead reckoning (Gallistel 1990, Müller and Wehner 1988, Mittelstaedt and Mittelstaedt 1982) using a homing vector accumulated from proper motion. Honeybees acquire such target

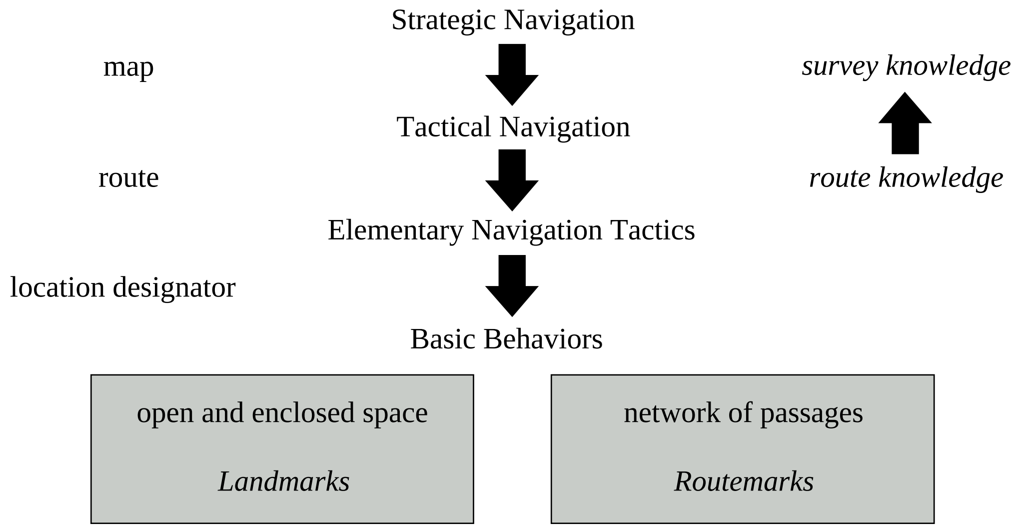

Figure 1.2: The Taxonomy of Navigation (Krieg-Brückner *et al.* 1998).

vectors by communication with their sisters via the bee-language (v. Frisch 1967). During actual navigation, the course is then computed relative to a compass direction, determined from, e.g., a magnetic compass, the polarization of light (Rossel and Wehner 1986) in a hazy sky, or the direction of one particular landmark such as the sun.

Alternatively, the representation may be a particular configuration of several global landmarks, i.e. a *view*, and guidance is used to reach the target position. In landmark navigation, these landmarks are fixed, e.g. prominent buildings, trees, lighthouses. In celestial navigation (Wehner 1983), landmarks are moving over the day, e.g. the sun, and over the year, e.g. the stars, and these movements have to be compensated for by an internal clock (circadian rhythm).

In contrast to other categorizations, *searching* is also classified as an elementary navigation tactic. Although this seems not to be real navigation, it is part of the navigation strategies of many living beings. For example, desert ants (Müller and Wehner 1988) and also honeybees (Cartwright and Collet 1983) start moving with a search-pattern as soon as they do not find the target where they expect it. Already 600 million years B C, multicellular organisms searched for food in spiral-shaped or meander-shaped search-patterns (Raup and Seilacher 1969). Prescott and Ibbotson (1997) showed that it is possible to reproduce these patterns with a robot that is controlled by a simple behavior-based program.

Krieg-Brückner *et al.* (1998) emphasized that navigation tactics in space seem to be equally applicable to open or enclosed space, e.g. an ocean, a town square, or a room.

Following their aforementioned differentiation, Krieg-Brückner *et al.* (1998) also described elementary navigation tactics in networks of passages. These can take advantage of the limited alternatives within a passage, e.g. binary or n-way branching. Characteristic views, depending on the direction from which the junction was approached, may be associated with the decision of selecting a particular target passage (Franz *et al.* 1997a, Schölkopf and Mallot 1995). According to the taxonomy, these views may be characterized by local landmarks or routemarks such as prominent visual objects, odors, sounds or tactile percepts, as long as they are stable, fixed and persistent. In robotics, artificial routemarks were often used for this purpose, e.g. beacons (Janda 1993) or visual marks of simple shape as will be presented in chapter 5.

Tactical Navigation. On this level, Krieg-Brückner *et al.* (1998) introduced the concept of *generalized routes*. In contrast to other approaches that described routes as sequences of locations, they distinguished between *homogeneous routes* that always use a single tactic and *heterogeneous routes* where each route segment may be traveled using a different tactic. They disagreed with the opinion of others, e.g. Herrmann *et al.* (1997), that navigation tactics can be canceled by the use of *inverse tactics*. Instead, they propose the use of *threading* for backtracking along a traveled route. This will also be described in chapter 5.

Strategic Navigation is the level of planning and the use of maps. This includes the combination of routes (route maps) as well as metrical maps. According to Krieg-Brückner *et al.* (1998), different tactical aspects of navigation may lead to different maps with different kinds of abstractions for tactic-oriented spatial knowledge contained in locations, e.g. by introducing topological or Cartesian relations (Barkowsky *et al.* 1997). Maps need not to be global, but may be used locally, e.g. *local charts* (Poucet 1993). Therefore, the strategic level includes the topological and the higher (non dead reckoning) metrical levels of the classification of Trullier *et al.* (1997) and the Spatial Semantic Hierarchy of Kuipers (1996).

On the one hand, planning detours or shortcuts requires only simple—maybe partial—route graphs that can be overlaid with additional quasi-survey knowledge, e.g. dead reckoning. On the other hand, premeditated navigation absolutely requires a map.

1.2 Local Navigation

In the previous section, several classifications for the field of global navigation have been given. The different techniques for local navigation can also be structured. In 2-D space, an agent can move in three degrees of freedom, i.e. two translational and one rotational. Although most living beings are able to briefly move independently in translation and rotation, these movements are most often joined together over longer distances, i.e., most animals move in the forward direction. In mobile 2-D robotics, there are two major types of autonomous mobile systems: robots with wheels and robots with legs, i.e. walking machines (cf. Röfer 1994b). The latter are currently less wide-spread, because so far, the required local control tasks are not sufficiently solved.

1.2.1 Autonomous Mobile Systems with Wheels

The category of driving robots can be divided into three classes with regard to the robot's movability:

Omni-Directional Robots. This class of robots is able to drive along any trajectory. The three movement components "x-direction", "y-direction" and "rotation" are completely independent of each other. Although mobile systems of this kind can move very flexibly, most of them have a cylindrical body. They are either equipped with special wheels (Mecanum-wheels) (Hörmann and Rembold 1991), or with a special steering mechanism, e.g. a synchro-drive, that allows any translation direction without a rotation. In the latter solution, rotations are provided by an additional motor that turns the robot's body. Members of this category are the frequently used experimental platforms *Real World Interfaces B21* (cf. figure 3.3 on page 73) that was, e.g., used by Burgard *et* *al.* (1997), and *Nomad 200* that was, e.g., used by Owen and Nehmzow (1997). There also exists an omni-directional wheelchair (Hoyer and Hoelper 1993, Hoyer *et al.* 1994).

Holonomic Robots. The orientation of robots of this kind is either connected to the driving direction, or it cannot be actively influenced.

- In the first case, the robot has to turn around its axis to change the driving direction. Hence, it has two degrees of freedom for its disposal that most often consist in the speeds of the two drive wheels that can be controlled separately. As autonomous mobile systems of this kind have to turn on the spot to change their driving direction, their bodies are often of cylindrical shape to enable them to perform rotations without collisions. The linkage between driving direction and orientation can be an advantage because the sensors can be fixed as being task-dependent, e.g., sensors can be oriented towards the front to recognize obstacles in the driving direction. An example for this is the *Pioneer*-robot that was, e.g., used by Gutmann and Nebel (1997).

- In the second case, the robot does not alter its orientation if it changes the driving direction. This can be realized, e.g., by a so-called synchro-drive (Röfer 1995a, 1995b, 1995c). Thus, the autonomous mobile system has the two degrees of freedom *speed* and *driving direction*. This restricted kinematic—because the rotation is missing—enables the system, e.g., to monitor landmarks without the need to compensate rotations. Nevertheless, a change of the orientation cannot be avoided during longer drives because of the slippage of the wheels (Duckett and Nehmzow 1997). As the robot has no possibility to correct these errors, this kind of autonomous mobile system can only be used for tasks that do not require a certain orientation. In addition, the sensory equipment has to be equally distributed around the whole robot because it has no preferred driving direction.

Kinematically Restricted Robots. Such robots most often have the classical kinematics of cars, i.e., they have a driving axle and a steering axle. Hence, they have two degrees of freedom in their movability: *speed* and *lock*. With these degrees of freedom, they can only move along curvy trajectories. As they are not able to turn on the spot, they often have to shunt to reach a certain orientation. Although the driving direction is dependent on the orientation of the robot, the sensors must perceive information from a wide range because the mobile system moves nearly sideways if it drives a narrow curve. Robots of this kind are often designed to transport heavy loads, and therefore they have no cylindrical body which impedes shunting, e.g., the rear of a robot that is back-steered first moves to the right in a left curve. Such robotics platforms were used by Mergel *et al.* (1994) and Wichert (1994). The wheelchair that has been used for the experiments presented in this work also falls in this category (Röfer 1997a).

1.2.2 Walking Machines

Whereas the local navigation of a driving robot only has to control two or three degrees of freedom, walking machines have a much larger number of movabilities. As a result, they are able to actively unlink the movement of their body from the structure of the ground. Driving robots require continuous paths to be able to move, e.g. streets. In contrast, walking machines only need a few small areas on the ground where they can put their feet. These areas of support can be selected in a way that enables the machine to overcome large obstacles or edges, i.e. hurdles that often cannot be surmounted by vehicles.

The advantages of walking machines are also their biggest weakness: because of the large number of joints, the control of a walking machine is very complicated, e.g., a human cannot control such a machine by himself or herself over a longer time (Liston and Mosher 1968). Hence, the user of a walking machine has at least to be supported by an automatic control system. In spite of substantial research in this area, a solution has not been developed that takes full advantage of the potential of walking machines.

One reason for this is the large number of sensors that is necessary to measure the state of the machine and the structure of the ground. The latter is required to find areas on the ground that are solid enough to support the machines weight. For each leg, a trajectory has to be determined that ensures that the machine does neither stumble nor fall over.

Walking machines can be categorized into three different classes:

Walking Machines with a Mechanically Predetermined Gait. The first walking machines generated their gait with the help of a sophisticated mechanic, e.g. the *Mechanical Horse* that was developed by L. A. Rygg but was never built (Raibert 1991). Machines with a static gait do not employ the superiority of going instead of driving. They cannot select advantageous areas of support for their feet or compensate for the unevenness of the ground. Therefore, they cannot be used in unstructured terrain. Walking machines with a mechanically predetermined gait most often have only one degree of freedom that determines the direction and the speed of the gait. Today, machines of this type can only be found as toys.

Walking Machines with a Statically Stable Gait. Machines of this category are always in a statically stable state during their walking movement, i.e., they would not fall over if their movement stopped abruptly. Most often, such machines have six legs and move with the *tetrapod* gait that is also used by many insects (Cruse 1991, Cruse *et al.* 1994). During this gait, three feet always have contact to the ground. If the weight of the body is shifted in an appropriate way, a stable gait can also be realized with only four legs (Röfer 1994a, 1994b) (cf. figure 1.3). Six-legged walking machines have most often 18 degrees of freedom, i.e. three joints per leg. Examples for this kind of machines are the walking machines *Hexapod* (Raibert 1991), *Lauron* (Cordes *et al.* 1993), *Attila* (Brooks 1987), and the Adaptive Suspension Vehicle (Waldron *et al.* 1984). Also six-legged but with a different arrangement, the development of the walking machine *Ambler* has begun at the Carnegie Mellon University (Bares *et al.* 1989, Krotkov *et al.* 1991). Ambler should move autonomously on foreign planets and meanwhile be tolerant of failing parts of the mechanics. Because of its special construction, up to two legs can break down without immobilizing the system. Another proposed task for the use of walking machines is the inspection of sewer systems. For this purpose, walking machines should be equipped with eight legs (Neubauer 1994). The Munich Walking Machine (Roßmann and Pfeiffer 1995, Eltze *et al.* 1993) addresses this area.

Walking Machines with a Dynamically Stable Gait. Walking machines with a dynamically stable gait use the dynamics of the movement to keep their balance. This requires very fast and precise movements. Nevertheless, it allows the realization of very sophisticated movement possibilities, e.g. running or jumping. Dynamically stable gaits have been developed for machines with one (*One-Legged Hopping Machine*), two (*Biped Running Machine*) and four legs (*Quadruped Running Machine*) (Raibert 1991). The Biped Running Machine was the first walking machine that was able to turn a somersault—at least in 90% of all attempts. Most walking machines that use a dynamically stable gait need an immense computational effort to keep their balance and function only on flat ground (Berns and Fiegert 1991). First attempts to let a machine learn a dynamically stable gait have been made by Snaith and Holland (1992). Their four-legged walking machine moved with a speed of 25 cm/s and a step distance of 10 cm. The control system was based on a system of reflexes and neural hardware.

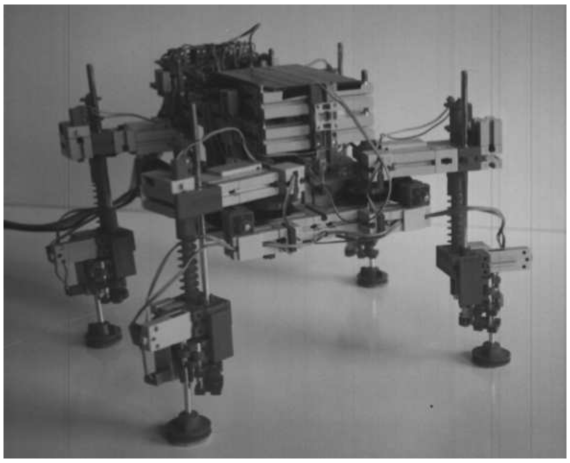

Figure 1.3: The quadruped walking machine *VierLING* (Röfer 1994a, 1994b).

1.3 Vision

The previous two sections have presented several navigation techniques. In order to be able to navigate, an agent must be capable to perceive information from its environment. Hence, it has to be equipped with sensors. Navigating agents can use a variety of different sensors for perceiving their surroundings. These sensors can be divided into two groups: *active sensors* and *passive sensors*. Active sensors modify the environment to be able to perform their measurements, e.g., by sending out sonar signals or laser light. Passive sensors only perceive quantities that are already present in the surroundings; they do not emit signals. The advantage of active sensors is the ability to control the moment and the intensity of the measured quantities. The major drawback is that multiple active sensors can influence each other and therefore distort their measurements. This is the justification for the use of passive sensors. In nature, most sensors are passive, apart from, e.g., bats that emit sonar pulses. The most sophisticated sense of many living beings is the visual one; it allows to perceive very exhaustive information about the environment, e.g. color, depth, spatial relations between objects, etc. All this is very natural for humans, nevertheless, it is hard to realize for artificial systems.

1.3.1 Depth Estimation

The major topic of vision research is the reconstruction of the three-dimensionality from two-dimensional images. There were several approaches to solve this problem that were all based on the idea of comparing images taken at different positions. Two methods to realize this are *structure from stereo* and *structure from motion*:

Structure from Stereo. In stereo vision (stereopsis), two images of the surroundings are taken at different positions at the same time, using two cameras. In order to determine the depth information, two processing steps have to be performed: in a first step, a correspondence needs to be established between features from the two images that correspond to physical feature in space. Then, the depth can be reconstructed using triangulation, provided the position of centers of projection, the effective focal length, the orientation of the optical axis, and the sampling interval of each camera are known. A review about the research performed in this area was given by Dhond and Aggarwal (1989). Barnard (1991) compared stereo vision in artificial and natural systems. As an exception from most other researchers, Wilcox *et al.* (1987) used three cameras instead of only two to achieve more reliable depth estimations.

Structure from Motion. In motion vision, several images are taken in sequence by a moving camera. On the one hand, this can be seen as *virtual* stereo vision because the different positions of the two stereo cameras are simulated by moving a single camera. On the other hand, motion vision can do more then just calculating the depth because it is possible to determine the movement of the camera itself, and hence the motion of the living being or robot that is equipped with this optical sensor.

Helmholtz (1909) was the first that described the phenomenon that for an *active observer* (Gibson *et al.* 1955), the visual field seems to flow towards and around the observer. This *optical flow field* (Gibson 1950, 1966, 1979) starts at the point towards which the observer is heading and it collapses where the observer comes from (cf. figure 1.4). These two points are called the *focus of expansion* (FOE) and the *focus of contraction* (FOC). The latest brief review of this area of research was given by Herwig (1996). It is interesting to note that there were approaches to combine motion and stereo vision to get more reliable depth information, e.g. by Grosso *et al.* (1991).

1.3.2 Image-Based Homing

Another field of vision research is the direct use of images for navigation—without reconstructing depth information. Most of this work were inspired by the *homing behavior* of, e.g., honeybees that was analyzed by Cartwright and Collet (1982, 1983, 1987) (cf. figure 1.5). They developed a model for the homing behavior of honeybees. They supposed that bees store a 360° (panoramic) *snapshot* of the surroundings when they are at a feeding site for the first time. Later, when they return to the feeding site, they compare their current retina image with the stored snapshot and, by the deviations between these two images, they are guided to the original place. In the model of Cartwright and Collet (1983), the snapshots only contain binary image information, which requires a lot of preprocessing for real applications.

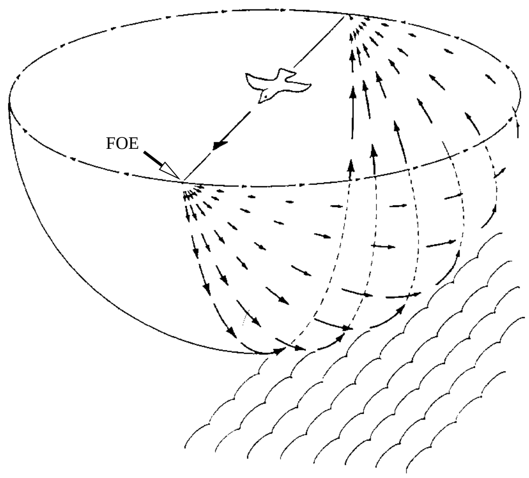

Figure 1.4: Flow induced by motion (Gibson 1966).

Hong *et al.* (1991) implemented a similar method on a real robot. In their approach, one-dimensional 360° intensity images are used as snapshots. The current view is searched for a fixed number of *characteristic points*. Then, the stored target image is searched for image areas that correspond to these characteristic points. From the deviations between the characteristic points in one image and their counterparts in the other image, the homing direction is calculated. Nevertheless, in the approach, only small deviations can be determined and only small rotations of the robot can be compensated for.

In the approach of Wittmann (1996), the deviations between stored target image and current image are determined by a correlation method in combination with a resolution pyramid. But also in this approach, the homing agent has to remain in a constant orientation.

Nelson and Aloimonos (1988) analyzed the calculation of rotations and translation directions from spherical motion fields. They showed that the determination of the motion parameters in 3-D space can be decoupled to calculating rotation and translation direction in each of three one-dimensional circular flow fields. Their approach for extracting the motion parameters from one-dimensional panoramic flow fields consisted in trying all combinations of rotation and translation direction and using the pair that fits best to the data. This results in a computational effort of $O(n^3)$ if the flow field contains n vectors and both parameters should be calculated with the same precision as the flow field's resolution. Hence, this approach is not suitable for real-time applications. In addition, Nelson and Aloimonos (1988) have ignored the question of the generation of the spherical motion field.

During the work on this thesis, other researchers have addressed the field of circular image processing, too. Chahl and Srinivasan (1997) developed their *image interpolation*

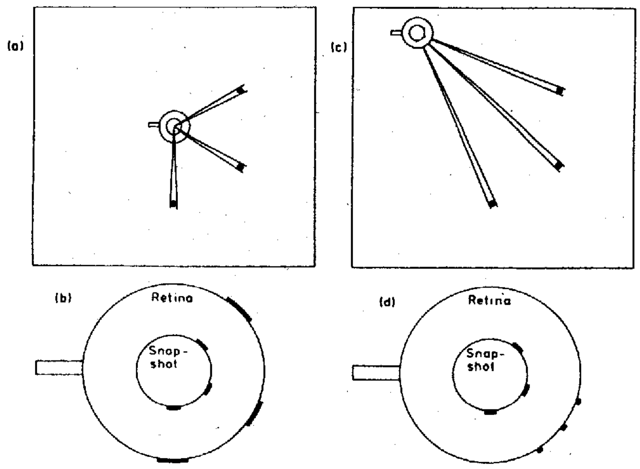

Figure 1.5: Snapshot matching in honeybee navigation (Cartwright and Collet 1983).

algorithm. In this algorithm, two panoramic images are taken at reference positions. Then, a third one is taken between these references. The image interpolation algorithm is able to determine where the third image has been taken between the two reference positions. On a real robot, Chahl and Srinivasan (1997) have only tested to determine either translation or rotation; in a simulation, their algorithm has calculated both at once.

Another approach came from Franz *et al.* (1997b). In their method, it is assumed that the distances between the camera and all objects in the surroundings are the same. Based on this assumption, they determined rotation and translation direction by minimizing an error function. As they used a gradient descent method, their approach is very fast. However, as the distances to the surrounding objects are not constant in general, this method does not always calculate the correct direction to the target position. Franz *et al.* (1997b) showed that their robot would normally reach the goal position but not on a straight line. Hence, the method is suitable for navigation in free space but unsuitable for indoor environments.

1.4 Structure of the Thesis

This thesis contributes to two research fields: on the one hand, it will introduce an algorithm for the processing of panoramic images, and on the other hand, it will present two navigation approaches and a metrical self-localization method that can also be used for navigation purposes.

Earlier work in the field of panoramic image processing has several weaknesses: either the problem of the spherical flow field estimation is not handled (Nelson and Aloimonos 1988) or the robot must remain in a constant orientation (Hong *et al.* 1991) which is impossible for kinematically restricted mobile systems. Hence, this thesis will present a complete implementation; starting with the flow field estimation, via the determination of the motion parameters and a navigation technique to the realization on a real, kinematically restricted robot.

In addition, a route navigation technique will be presented. Although much research has been performed that includes some route navigation elements, the topic of error recovery has only addressed by few researchers in this field, e.g. by Kuipers (1996). Hence, this thesis will present a route navigation approach that performs error recovery with minimal information. It has been implementated on the same kinematically restricted robot.

This thesis will present the following research results:

- A robust approach for the determination of the deviations (optical flow field) between two panoramic images that is based on the assumption of *neighborhood preservation*, i.e., the order of objects is supposed to be the same in both images (chapter 2).

- The calculation of both the rotation *and* the translation direction between the positions where the two panoramic images were taken; in contrast to the models of Cartwright and Collet (1983), Hong *et al.* (1991), and Wittmann (1996). Several alternatives in the implementation will be compared under different environmental conditions, e.g. under different levels of perturbation of the images used (chapter 2).

- The embedding of this image processing method in a navigation approach. This approach is based on *teaching* and is especially designed to overcome the main limitation of the flow field calculation, i.e. the assumption of a constant sequence of objects in both images (chapter 3).

- The implementation of the complete system on a simulated holonomic robot and on a real, kinematically restricted autonomous mobile robot, i.e. an electric wheelchair. The performance of these implementations will be analyzed (chapter 3).

- An approach, how the developed image processing method can be used for the metrical self-localization without artificial landmarks. The precision of the position estimations will analyzed under different environmental conditions, e.g. under different levels of noise in the used images (chapter 4).

- A further navigation method that is based on several *basic behaviors* and the detection of *artificial routemarks*. As an exception from the rest of the thesis, no panoramic images have been used in this navigation technique. A major aspect of this method consists in the considerations on the recovery from navigational errors (chapter 5).

Chapter 6 will summarize the most important results of this thesis and compare them with older and recent results in the areas of spherical image processing, robot homing, and route-based navigation.

Appendix A will give an overview of the software architecture of the wheelchair.

In some equations, a notation of *lists* is employed. Appendix B will briefly describe this notation.

Chapter 2

The Panama Algorithm

This chapter will present the image processing method *Panama* (*Pan*oram*a Ma*tching) that calculates the optical flow field between two one-dimensional panoramic pictures and extracts some motion components from this flow field. Panoramic pictures are colored, one-dimensional 360°-takings of a special camera that perceives only a horizontal stripe of the surroundings.

The two motion components *rotation* and *bearing direction* describe the spatial relationship between the positions where these two images have been taken. They represent the rotation between these positions and the bearing direction from one of the positions to the other. In addition, the images' *distance* is calculated. This value determines the size of the translational flow field, and therefore it corresponds to the spatial distance between the two positions where the pictures have been taken. As the dimensions of the surroundings are unknown, the distance does not give a metrical length between the positions. It is only relative to the current environment and can thus only be used to determine whether one position is closer to a second position than to a third one.

Panama differs from other flow field based vision methods, e.g. "structure from stereo" (Dhond and Aggarwal 1989) and "structure from motion" (Barron 1984, Aggarwal and Nandhakumar 1988, Heeger and Jepson 1992) because the pictures used are taken neither simultaneously nor in short sequence. Whereas other methods try to determine the movement direction of the camera itself (Herwig 1996) or the relative distance to the objects in the images, *Panama* calculates the motion parameters from two pictures that have been taken at positions that are up to several meters apart. Therefore, the demands on *Panama* differ heavily from those on other methods:

- The optical flow field between the two images can be very large. The positions where the pictures have been taken can be far apart and in addition, they can be rotated up to $\pm\pi$ against each other. Therefore, the possible flow vectors lie in the range of $[-\pi \ldots \pi[$, and thus the flow field is considerable larger than in methods that are based on planar images. Hence, this vector field is not really a *flow* field, because it is not motion that is determined. However, as *flow field* and *motion field* are established terms, they are used in this thesis to describe the image deviations because they represent the same information.

- Neither the spatial nor the temporal distance between the positions where the pictures have been taken is known. Therefore, it is impossible to calculate the distances to the objects in the images.

- The method should be robust against small changes in the environment or the illumination because of the long chronological distance between the taking of the two pictures.

To cope with these demands, *Panama* uses two distinctive features:

1. As it is assumed of the bee (Lehrer 1994), *Panama* uses color images to store target positions. Therefore, different picture regions can be better distinguished and confusions can be avoided, even if the surroundings have changed in some places.

2. *Panama* uses a strict neighborhood preservation. It is assumed that the sequence of objects is the same in both images. This assumption stabilizes the calculation of the optical flow field. Moreover, it provides for a continuous flow field.

   Neighborhood preservation is often associated with the self organizing feature map (SOFM) (Kohonen 1982). In contrast to this network model, *Panama* uses an

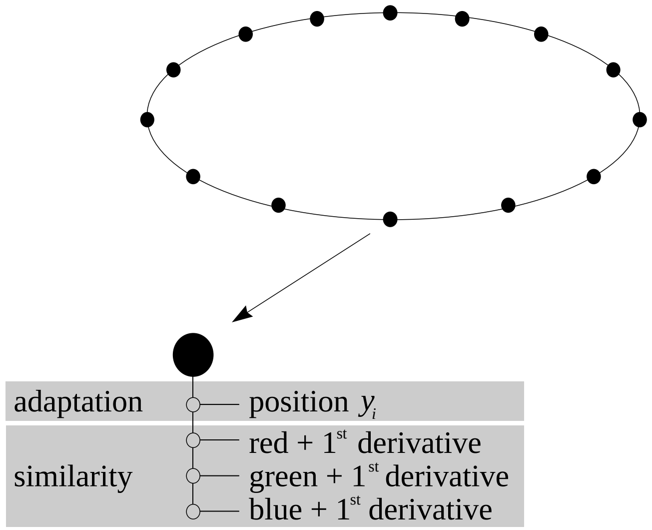

Figure 2.1: Illustration of *Panama*'s differentiation in the use of the weight vectors between similarity calculation and adaptation.

adaptation rule that never changes the pixel's order. Similarly to the SOFM, weight vectors containing information calculated from the pixels are topologically ordered in a network. In contrast to the SOFM, this topological structure is a circular chain (cf. figure 2.1). While the SOFM always adapts all components of the network's vectors, *Panama* divides each vector in two parts: One part (the color information) is used in the search for highest similarity, and only the other part (the pixel's position) is changed in the adaptation process. Figure 2.1 visualizes this separation of the weight vectors.

First, this chapter will deal with the particular properties of panoramic pictures. Then, the calculation of the optical flow field between two of these images will be presented. A discussion of several methods for determining the rotation and the bearing direction is followed by an analysis of *Panama*'s computational complexity. Finally, some experiments with *Panama* and their results are presented.

2.1 Panoramic Images

Panama uses one-dimensional, panoramic images. These are pictures that consist of a circular sequence of pixels, i.e., the last pixel is again followed by the first one. A sensor that is able to take such images can be imagined as a line camera with a horizontal opening angle of 2π. In comparison, the vertical opening angle is very small so that the camera recognizes only a thin stripe of the surroundings but this stripe contains the full range

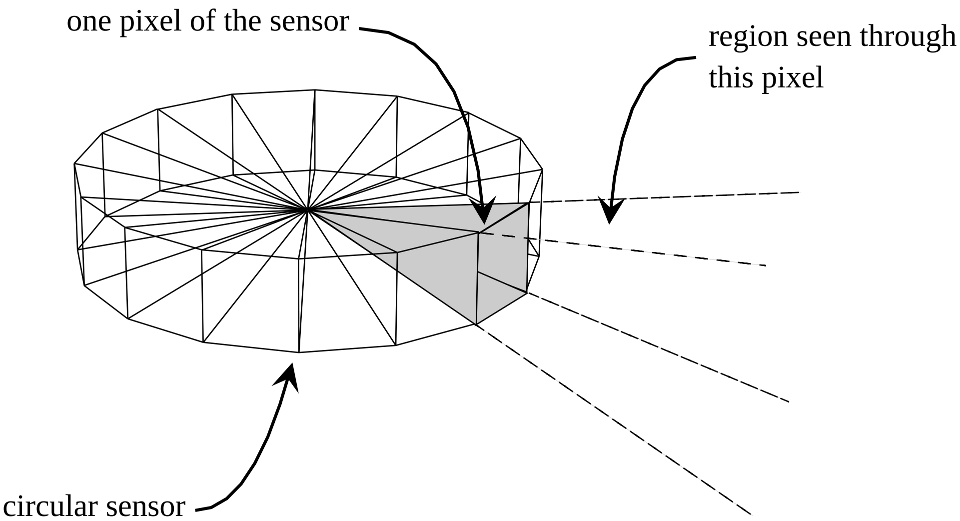

Figure 2.2: Illustration of a sensor that takes panoramic images.

of 2π. Figure 2.2 schematically shows this kind of sensor. Section 3.3.2 will describe the realization of such a sensor with a normal CCD-camera. For clarification, figure 2.3 shows a simulated scene and figure 2.4 visualizes a panoramic picture that the sensor would take in this scene.

A basic condition for image processing methods to function with these pictures is that the camera is only moved along the viewable plane. This is true for most of the cameras that are mounted on autonomous mobile systems: the vehicles move only along the horizontal plane, and therefore the camera must also be oriented horizontally to recognize all the movements of the autonomous system.

Panoramic images have some advantages in comparison to planar pictures (Nelson and Aloimonos 1988):

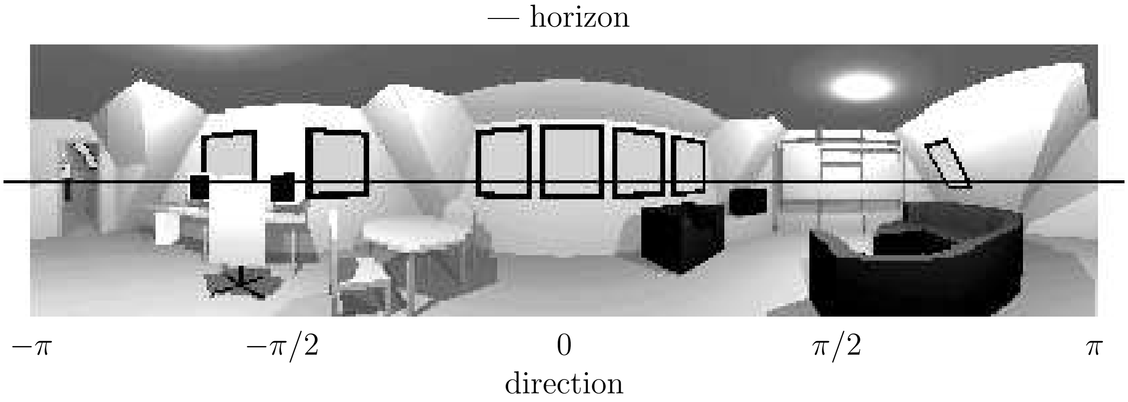

Figure 2.3: A simulated panoramic view. A circular picture only consists of the image information near the horizon.

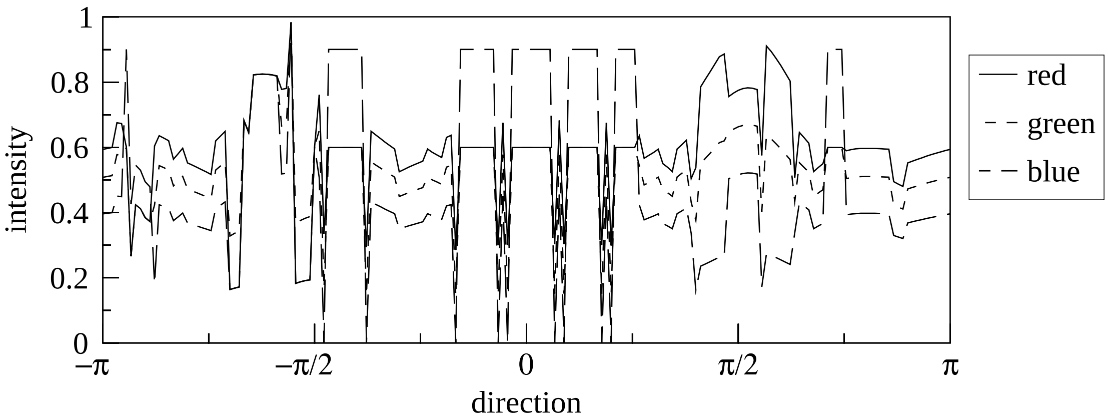

Figure 2.4: A panoramic image calculated from the horizon of the picture in figure 2.3.

- As circular images always contain the whole surroundings, the picture seen cannot leave the view range partly or completely during a rotation. Therefore, the rotational flow field can always be determined.

- The rotational component of the flow field is equal for all pixels because of the circular geometry of the camera (cf. figure 2.5c). In contrast, the flow is not constant over the whole picture in planar images (cf. figure 2.5b). This makes the calculation of the rotation in panoramic images easier because nothing about the physical dimensions of the camera has to be known.

- Even with translational movements, all image information is preserved: the different picture regions will just become smaller or larger. Only if there are objects in the images, which hide other objects behind them, some information is lost during the camera movement. However, this is also true for planar pictures.

In this work, all panoramic images are taken by a camera oriented horizontally. In contrast to other pictures, the sequence of the pixels runs from right to left. Seen from the bird's eye view, the pixels are ordered in counterclockwise direction. This order has been chosen because the images are used to calculate angular information—as rotation and direction—which is usually visualized in counterclockwise direction, too.

Mathematically, panoramic pictures can be described as a vector I with n components that are the images' pixels p_i. In the presented implementation, the pixels consist of the color components red, green and blue. It is conceivable that the pictures are generated from totally different information, e.g. the surface properties (roughness, temperature) of the objects in a particular direction.

$$\begin{aligned} I &= \begin{pmatrix} p_0^I & \cdots & p_{n-1}^I \end{pmatrix} \\ \bigwedge_{i=0}^{n-1} p_i &= \begin{pmatrix} r_i \\ g_i \\ b_i \end{pmatrix} \end{aligned} \tag{2.1}$$

As the images are panoramic, a modular arithmetic for the indices of the vector I has to be defined that ensures that the calculations with these indices consider the circularity and do not leave the range $[0 \ldots n[$:

$$\begin{aligned} a \dot{+} b &\stackrel{\text{def}}{=} (a+b) \mathbin{\dot{\mathrm{mod}}} n \\ a \dot{-} b &\stackrel{\text{def}}{=} a \dot{+} (-b) \end{aligned} \tag{2.2}$$

In addition, an arithmetic is needed to calculate the shortest distances in this range. The operations always result in numbers in the range $\left[-\frac{n}{2} \ldots \frac{n}{2}\right[$ because of the pictures' circularity:

$$a \ddot{+} b \stackrel{\text{def}}{=} \begin{cases} a \dot{+} b & \text{if } a \dot{+} b < \frac{n}{2} \\ a \dot{+} b - n & \text{if } a \dot{+} b \geq \frac{n}{2} \end{cases}$$
$$a \ddot{-} b \stackrel{\text{def}}{=} a \ddot{+} (-b) \tag{2.3}$$

As the modulo operation *mod* is defined as *remainder* in many programming languages, e.g. in *C* (Kernighan and Ritchie 1977) and *Miranda* (Bird and Wadler 1988), all defined operators use the $\dot{mod}$ operator that implements a modulo operation that generates results that are always in the range of $[0 \ldots n[$:

$$a \;\dot{\text{mod}}\; b \stackrel{\text{def}}{=} \begin{cases} a \bmod b & \text{if } a \geq 0 \\ b - 1 - (-1 - a) \bmod b & \text{if } a < 0 \end{cases} \tag{2.4}$$

2.2 Optical Flow Field

Panama is a correlation based method for the calculation of the optical flow field between two images. Such a method determines, where the objects of one picture can be found in the other. *Panama* solves this problem on the level of pixels. It determines where each pixel of one picture J is placed in the other image I. If two pictures have been taken at positions that have different distances to a particular object, this object is represented by a greater number of pixels in one image than in the other. Therefore, a one-to-one relation between the pixels of these two pictures cannot occur. Instead, the points of one picture

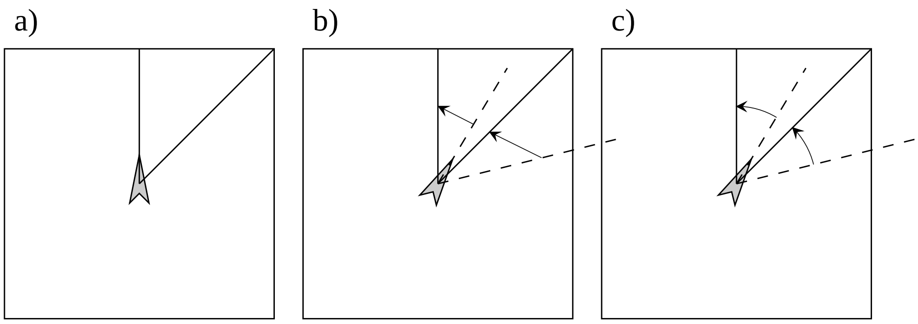

Figure 2.5: This illustration shows the difference between the rotational flow in plain pictures and in panoramic images. a) The camera's original position with two bearings. b) The camera's position after a rotation. The arrows indicate the flow in a plain picture between the bearings in (a) and now. The flow is different for the two bearings. c) Same as (b) but the arrows indicate the flow in a circular image. The flow is the same for all bearings.

can be found *between* the pixels of the other. As a result, *Panama* does not compare each pixel of one image with each of the other. Instead, it calculates the similarity of each pixel in the first picture with each pair of adjacent pixels in the second image. This establishes where the similarity to the pixel in the first picture is best between two neighboring points.

The determination of the flow field can be separated into three tasks. First, the correlation matrix is calculated that contains the assessment of similarity between each pixel of one image and each adjacent pixel-pair of the other. Next, this information is used by an iterative process that determines the matching between the pixels of both pictures. Finally, the optical flow field is calculated from this matching information.

2.2.1 Correlation Matrix

The similarity between each pixel of the image J and an adjacent pixel pair of the other picture I is stored in the correlation matrix C. In addition, the place is stored where the highest similarity occurs between the two neighboring pixels and the single pixel. As both images consist of n pixels, the matrix has $n \times n$ entries:

$$C^{I,J} = \begin{pmatrix} c_{0,0}^{I,J} & \cdots & c_{n-1,0}^{I,J} \\ \vdots & \ddots & \vdots \\ c_{0,n-1}^{I,J} & \cdots & c_{n-1,n-1}^{I,J} \end{pmatrix} \tag{2.5}$$

The similarity between the pixels is not directly established from the color information, but is determined instead from *weight vectors* that have been previously calculated for every pixel. Weight vectors can consist of many different components, for example of the original color values, their change or smoothed values. The weight vectors of a picture I are defined as

$$W^I = \begin{pmatrix} w_0^I & w_1^I & \cdots & w_{n-1}^I \end{pmatrix} \tag{2.6}$$

The most simple way to define such a vector is to copy the image without any change:

$$W^I = I \tag{2.7}$$

Several possibilities for the definition of W^I will be analyzed in section 2.5.

A common method to calculate the similarity between two vectors is the Euclidean distance. As the similarity between one vector and two others must be calculated, this method has to be extended. To realize this, a geometric approach is pursued. It connects two adjacent weight vectors of I by a straight line. The shortest distance between the straight line and the weight vector of a pixel from picture J is used to assess the similarity. It is calculated by dropping the perpendicular from the single vector to the straight line.

If the perpendicular point is not between the two weight vectors on the straight line, the closest point inside the valid range is used instead. This range includes the first weight vector but does not contain the second, because that one is part of the valid range between the next vector pair. Figure 2.6 shows the three possible cases.

The similarity $s_{i,j}^{I,J}$ is stored in the correlation matrix C for each pixel combination. In addition, it is stored where the similarity has its maximum. This information is represented as a real numbered index $x_{i,j}^{I,J}$. It is calculated by adding the index of the first of the two adjacent pixels to an offset in the range $[0\ldots1[$, which states where the most similar point on the straight line between the two weight vectors is:

$$\begin{aligned} \bigwedge_{i=0}^{n-1}\bigwedge_{j=0}^{n-1} c_{i,j}^{I,J} &= \begin{pmatrix} s_{i,j}^{I,J} \\ x_{i,j}^{I,J} \end{pmatrix} \\ \text{where} \quad & s_{i,j}^{I,J} = |b - af| \\ & x_{i,j}^{I,J} = i + f \\ & a = w_{i+1}^{I} - w_{i}^{I} \\ & b = w_{j}^{J} - w_{i}^{I} \\ & f = \max\left(\min\left(\tfrac{ab}{a^2}, 1-\varepsilon\right), 0\right) \\ & \varepsilon \approx 0, \varepsilon > 0 \end{aligned} \tag{2.8}$$

2.2.2 Matching Process

The pixels of the two images I and J are matched in an iterative process under the assumption that the order of objects is the same in both pictures. As a result, the pixels are not always assigned to the point with the greatest similarity in the other image. Instead, a compromise is found that preserves the order of pixels. It pairs the points that are as similar as possible. The matching is done by moving the pixels of picture J. This is realized by defining the positions of the pixels in this image being variable. These positions are initially identical with the indices of the corresponding pixels. In contrast to

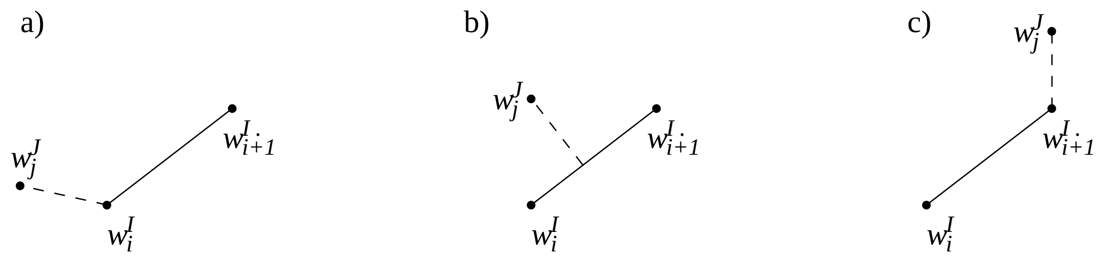

Figure 2.6: The shortest distance between w_j^J and the connecting line from w_i^I to w_{i+1}^I in equation (2.8). a) $f = 0$; b) $0 < f < 1 - \varepsilon$; c) $f = 1 - \varepsilon$.

the indices, the positions of the points are real numbers and so the pixels can be moved to every place in the range $[0 \ldots n[$.

The matching occurs in t_{max} steps. In every step t of the adaptation process, one pixel of picture J is selected. Then, the neighborhood of its current position is searched for the most similar pixel-pair from the other image in a range of $\pm\sigma_t$ pixels. It is only necessary to search the corresponding part in the row of the correlation matrix to which it belongs. Once this has been done, this pixel is moved to the position of the most similar point that is stored in the matrix as well. The pixel's vicinity of $\pm\sigma_t$ pixels is moved in the same direction in a way that does not change the order of the pixels' positions. The neighborhood $]-\sigma_t \ldots \sigma_t[$ initially contains the whole picture. The adaption process continues until the neighborhood shrinks to include only the pixel. This allows the pixels to move to their globally most similar positions in the beginning of the process. While the process runs, they can select their partners from a continuously shrinking number of positions. With time, fewer pixels move and distances moved in one step decrease.

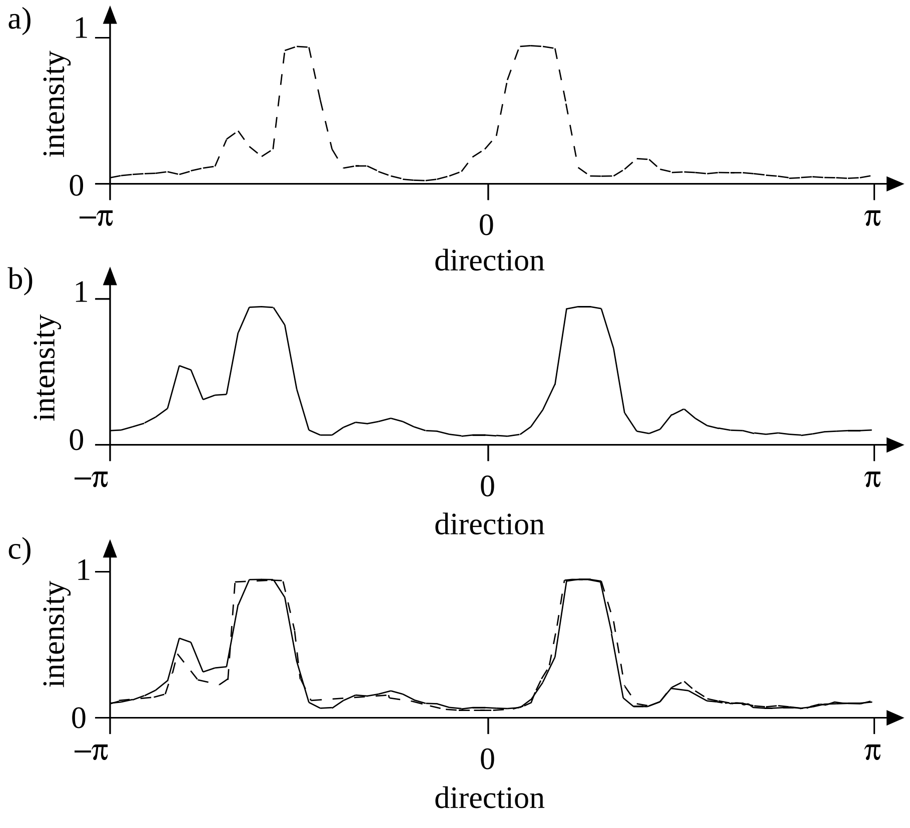

Figure 2.7: Illustration of matching two monochrome images. a, b) Two images that have been taken at different positions. c) The adaptation of image (a) to image (b).

Finally, each pixel should reach its optimal position, with respect to the basic condition of neighborhood preservation. Figure 2.7 shows the results of such an adaptation process. The images in this sample have been taken by the robot that is shown in figure 3.1a on page 62.

The size of the neighborhood σ_t that is searched and adapted is determined from the value of the adaptation step t. The decrease of σ_t is not linear, it is greater in the beginning and gets smaller with time:

$$\sigma_t = 1 + \frac{\frac{n}{2} - 2}{t_{max}^2} \left(t_{max} - t\right)^2 \tag{2.9}$$

In every step t, one pixel is selected to be adapted. The pixels should be selected in a sequence that has a uniform distribution. One possibility to achieve this is to select the pixels at random but this results only in a uniform distribution if the number of adaptation steps t_{max} is very big. As the number of steps should not be unusually big in comparison to the number of pixels, it is better to use a static sequence. If the number of pixels is a power of 2, a function can be defined that generates a sequence of n numbers with a uniform distribution. To produce a sequence with more than n numbers, the series is repeated.

The function Q generates the succession that is illustrated in figure 2.8 from the sequence $0 \ldots n-1$. Q interprets its parameter as a binary value with $\log_2 n$ digits. To achieve a uniformly distributed series of numbers, it inverts the order of these binary digits. As the produced sequence always ends with $n-1$, that is—because of the circularity—in direct neighborhood to the starting point 0 of the following repetition, the next sequence

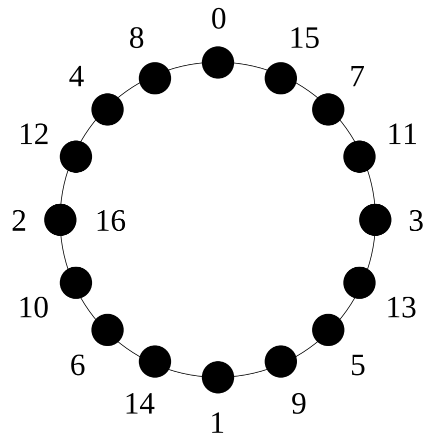

Figure 2.8: The sequence of indices generated by equation (2.10) for $t = [0 \ldots 16]$ and $n = 16$. The numbers represent the 17 states of t, their positions the resulting index.

is rotated by $\frac{n}{4}$. As a result, a uniform distribution of the adapted pixels is achieved for all t steps:

$$\begin{aligned} j_t &= \left(Q^n_{t \bmod n} + \tfrac{n}{4}\left\lfloor \tfrac{t}{n} \right\rfloor\right) \bmod n \\ \text{where} \quad Q^n_t &= \begin{cases} t \bmod 2 & \text{if } n = 0 \\ 2^n\,(t \bmod 2) + Q^{n-1}_{\left\lfloor \frac{t}{2} \right\rfloor} & \text{if } n > 0 \end{cases} \end{aligned} \tag{2.10}$$

The positions of the pixels of the image J are stored in vector Y. This vector is initialized with the indices of the points, i.e. the numbers from 0 to $n-1$:

$$\begin{aligned} Y_{init} &= \begin{pmatrix} y_0 & y_1 & \cdots & y_{n-1} \end{pmatrix} \\ &= \begin{pmatrix} 0 & 1 & \cdots & n-1 \end{pmatrix} \end{aligned} \tag{2.11}$$

In every step t, one pixel $p^J_{j_t}$ is selected. The correlation matrix C is searched for the most similar pair of pixels in the neighborhood of $\pm\sigma_t$ pixels around the position Y_{j_t}.

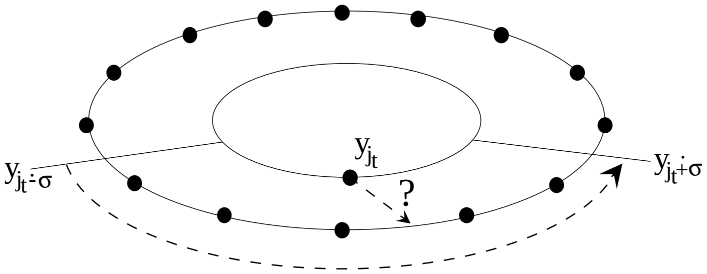

Figure 2.9: Illustration of the search range around the position Y_{j_t} of the selected pixel $p^J_{j_t}$

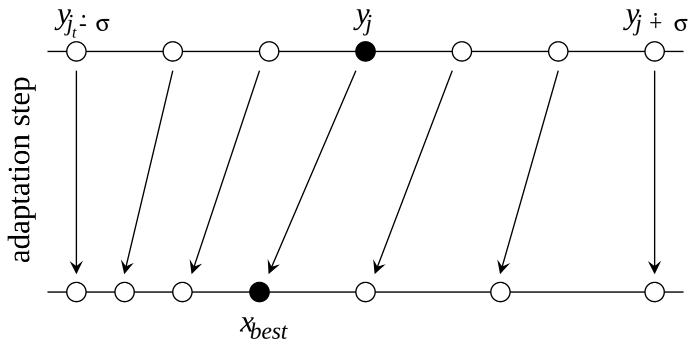

Figure 2.10: Illustration of the adaptation step in equation (2.13).

Figure 2.9 illustrates this search range. The result of this search is the position of the best matching point x_{best}:

$$
\begin{aligned}
s^{I,J}_{i_{best},j_t} &= \min_{i=\left\lfloor y_{j_t} \dot{-} \sigma_t \right\rfloor + 1}^{\left\lceil y_{j_t} \dot{+} \sigma_t \right\rceil \dot{-} 1} s^{I,J}_{i,j_t} \\
x_{best} &= x^{I,J}_{i_{best},j_t}
\end{aligned}
\tag{2.12}
$$

After the most similar position has been found, the pixel is moved to this place. In addition, all neighboring points with the indices $\left] j_t \dot{-} \sigma_t \ldots j_t \dot{+} \sigma_t \right[$ are moved in the same direction. The greater the distances between the positions of these neighbors and Y_{j_t} are, the less the pixels are moved. This preserves the order of their positions. Figure 2.10 illustrates the change of the positions during an adaptation step.

$$
\bigwedge_{i=0}^{n-1} y_i^{new} = \begin{cases}
x_{best} \dot{+} \frac{x_{best} \ddot{-} y_{j_t \dot{-} \sigma_t}}{y_{j_t} \ddot{-} y_{j_t \dot{-} \sigma_t}} \left(y_i \ddot{-} y_{j_t} \right) & \text{if } i \in \left[1 \dot{-} \sigma_t \ldots 0 \right] \\
x_{best} \dot{+} \frac{x_{best} \ddot{-} y_{j_t \dot{+} \sigma_t}}{y_{j_t} \ddot{-} y_{j_t \dot{+} \sigma_t}} \left(y_i \ddot{-} y_{j_t} \right) & \text{if } i \in \left[1 \ldots \sigma_t \dot{-} 1 \right] \\
y_i & \text{otherwise}
\end{cases}
\tag{2.13}
$$

At this point, it is necessary to consider the implementation of the algorithm on a real machine. As equation (2.13) is applied very often and most programming languages use an arithmetic of limited precision, it is possible that the order of the y_i changes or two adjacent positions get equal. These inaccuracies may cause several problems, e.g. a division by zero during the next use of equation (2.13). Therefore, it is necessary to check for a change in the order of positions and to cancel the last application of equation (2.13) if such an alteration is detected.

Y is correct as long as all neighboring positions are different and there is exactly one pair of y_i and $y_{i \dot{+} 1}$ with y_i greater than $y_{i \dot{+} 1}$. The latter is the case at the point where the circle closes, i.e., where a position near n is followed by a position near 0:

$$
\left(\bigwedge_{i=0}^{n-1} y_i \neq y_{i \dot{+} 1} \right) \wedge \bigvee_{i=0}^{n-1} \left(y_i > y_{i \dot{+} 1} \wedge \bigwedge_{j=0}^{n-1} i = j \vee y_i < y_{i \dot{+} 1} \right)
\tag{2.14}
$$

2.2.3 Optical Flow Field

After the matching process, the vector Y contains the adapted positions of the pixels in J. The difference between their original positions and their matched positions is the optical flow field Y' between the two pictures:

$$
\begin{aligned}
Y' &= \begin{pmatrix} y'_0 & \cdots & y'_{n-1} \end{pmatrix} \\
\bigwedge_{i=0}^{n-1} y'_i &= y_i \ddot{-} i
\end{aligned}
\tag{2.15}
$$

The calculated flow field vectors lie in the range of $\left[-\frac{n}{2}\ldots\frac{n}{2}\right[$. As the neighborhood preservation has been used to calculate Y, the flow field Y' is continuous. Therefore, the flow vectors do not cross each other, with one exception: if there are flow vectors up to $\pm\frac{n}{2}$, it is possible that adjacent vectors jump from $-\frac{n}{2}$ to $\frac{n}{2}$ and back (cf. figure 2.11c). This may confuse further processing. It would be better to have a continuous flow field in this region, too. Therefore, the flow field must be converted to a continuous one. To achieve this, some flow vectors must be greater than $\pm\frac{n}{2}$ (cf. figure 2.11d). The corrected flow field Y'' is calculated by adding an offset vector O to Y':

$$\begin{aligned} Y'' &= Y' + O \\ O &= \begin{pmatrix} o_0 & \cdots & o_{n-1} \end{pmatrix} \end{aligned} \tag{2.16}$$

The calculation of O is based on the assumption that the absolutely smallest flow vector $y'_{i_{min}}$ is correct. First this index i_{min} has to be determined:

$$\left|y'_{i_{min}}\right| = \min_{i=0}^{n-1} |y'_i| \tag{2.17}$$

Starting with vector $y'_{i_{min}}$, the algorithm runs to the left and to the right through the vector field and calculates the o_i in a way that ensures that the sign in the resulting Y'' vector changes only around 0 and not around $\pm\frac{n}{2}$. As $y'_{i_{min}}$ is assumed to be correct, $o_{i_{min}}$ must be zero. Then, O is calculated for the flow vectors to the left and right of $y'_{i_{min}}$. The calculation is performed recursively because o_i is needed to determine o_{i-1} or o_{i+1}, depending on which side—left or right—is calculated:

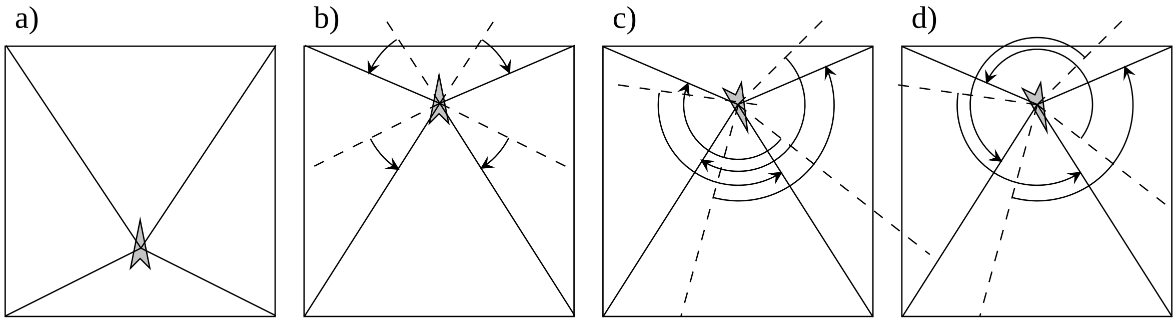

Figure 2.11: Illustration of the correction of the optical flow. a) The camera's original position with the four bearings of the corners. b) The camera's position after a translation. The arrows indicate the flow Y' between the bearings of the corners in a) and now. c) The camera's position after a translation and a rotation. The arrows indicate the flow Y' without correction. d) Same as (c) but the arrows now indicate the corrected flow Y''.

$$
\begin{aligned}
& o_{i_{min}} = 0 \\
& \bigwedge_{i=i_{min}-1}^{0} o_i = \begin{cases} -n \operatorname{sgn}(y'_i) & \text{if } \left|y'_i - y'_{i+1}\right| > \frac{n}{2} \wedge \operatorname{sgn}(y'_i) \neq \operatorname{sgn}\left(y'_{i+1} + o_{i+1}\right) \\ 0 & \text{if } \left|y'_i - y'_{i+1}\right| > \frac{n}{2} \wedge \operatorname{sgn}(y'_i) = \operatorname{sgn}\left(y'_{i+1} + o_{i+1}\right) \\ o_{i+1} & \text{otherwise} \end{cases} \\
& \bigwedge_{i=i_{min}+1}^{n-1} o_i = \begin{cases} -n \operatorname{sgn}(y'_i) & \text{if } \left|y'_i - y'_{i-1}\right| > \frac{n}{2} \wedge \operatorname{sgn}(y'_i) \neq \operatorname{sgn}\left(y'_{i-1} + o_{i-1}\right) \\ 0 & \text{if } \left|y'_i - y'_{i-1}\right| > \frac{n}{2} \wedge \operatorname{sgn}(y'_i) = \operatorname{sgn}\left(y'_{i-1} + o_{i-1}\right) \\ o_{i-1} & \text{otherwise} \end{cases}
\end{aligned}
\tag{2.18}
$$

The resulting flow field is continuous even if the two pictures are rotated against each other up to $\pm\pi$ (cf. figure 2.11d).

2.3 Movement Components

The optical flow field determined can thus be used to calculate the motion components *rotation* and *bearing direction* that describe the spatial relationship between the positions where the two matched images have been taken. In addition, the sum of the translational flow field can be determined that may be useful for navigation purposes as will be shown in the next chapter. This value is called the *distance* between the two images although it gives no metrical measurement.

2.3.1 Calculation of the Rotation

The rotation describes how much the orientation of the camera has changed between the taking of the two images. It can be determined from the optical flow field that has been calculated in the previous section. Each vector of the flow field consists of two parts: a rotational component and a translational component. This is illustrated in figure 2.12. As the pictures are panoramic, the rotational component is equal in all flow field vectors (cf. figure 2.5c on page 24). Based on this fact, two solutions will be presented to determine the rotation.

Solution 1

The first solution uses a straightforward approach. It is based on the idea that the translational flow can be separated into two regions that are divided by the FOE and the FOC (cf. section 1.3 beginning on page 14 and figure 2.14a on page 36). In both parts, the flow vectors point in opposite direction to the translation. Therefore, the flow in one part is clockwise and in the other, it is counterclockwise. The approach assumes that the summed up translational flow on both sides approximately differs in the sign.

Therefore, it is possible to even out the translational flow by summing up all vectors. The resulting sum contains only the rotational flow and, because it is the same in all vectors, the rotation can simply be calculated by dividing the sum by the number of flow vectors. So in this solution, the rotation is the average of the flow field vectors. Therefore, the rotation in radians ω can be determined by

$$\omega = \frac{2\pi}{n^2} \sum_{n=0}^{n-1} y_i'' \tag{2.19}$$

Solution 2

Solution 1 is based on the assumption that the translational flow is equal on both sides of the flow field. This is not always the case. If the surroundings are closer on one side than on the other, the translational flow is not uniformly distributed. This causes solution 1 to calculate a wrong rotation. Therefore, a better solution has to be developed. Nevertheless, one assumption in the previous approach is correct: without a rotation, half of the flow vectors point clockwise and half point counterclockwise. So finding the rotation is equivalent to transforming the flow field into a field that satisfies this criterion. As all flow vectors are only single real numbers, a value must be found which can be subtracted from all vectors, resulting in half of the vectors being positive and half being negative. To find this number all vectors are sorted in increasing order:

$$Y^{sort} = \text{sort}\,(Y'') \tag{2.20}$$

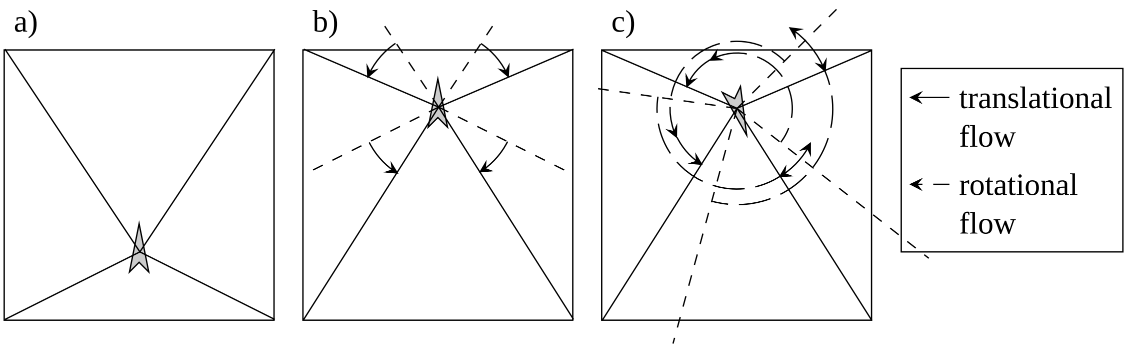

Figure 2.12: Illustration of the optical flow under translation and rotation. a) The camera's original position with the four bearings of the corners. b) The camera's position after a translation. The arrows indicate the flow between the bearings of the corners in (a) and now. c) The camera's position after a translation and a rotation. The solid arrows indicate the translational flow that is identical to the flow in (b), the arrows with the broken lines indicate the rotational flow.

To make half of this sorted vector field negative and half positive by subtracting a single value from all entries y_i^{sort}, it is enough to find a value that makes $y_{\frac{n}{2}-1}^{sort}$ negative and $y_{\frac{n}{2}}^{sort}$ positive. All numbers between $y_{\frac{n}{2}-1}^{sort}$ and $y_{\frac{n}{2}}^{sort}$ satisfy this criterion. Therefore, the center of this range is chosen as the best compromise (cf. figure 2.13). ω is calculated in radians as

$$\omega = \pi \frac{y_{\frac{n}{2}-1}^{sort} + y_{\frac{n}{2}}^{sort}}{n} \tag{2.21}$$

The idea of sorting the vector field seems to be more straightforward than the approach of Nelson and Aloimonos (1988), who determined the rotation by minimizing an error function between the expected translational flow field and the calculated one.

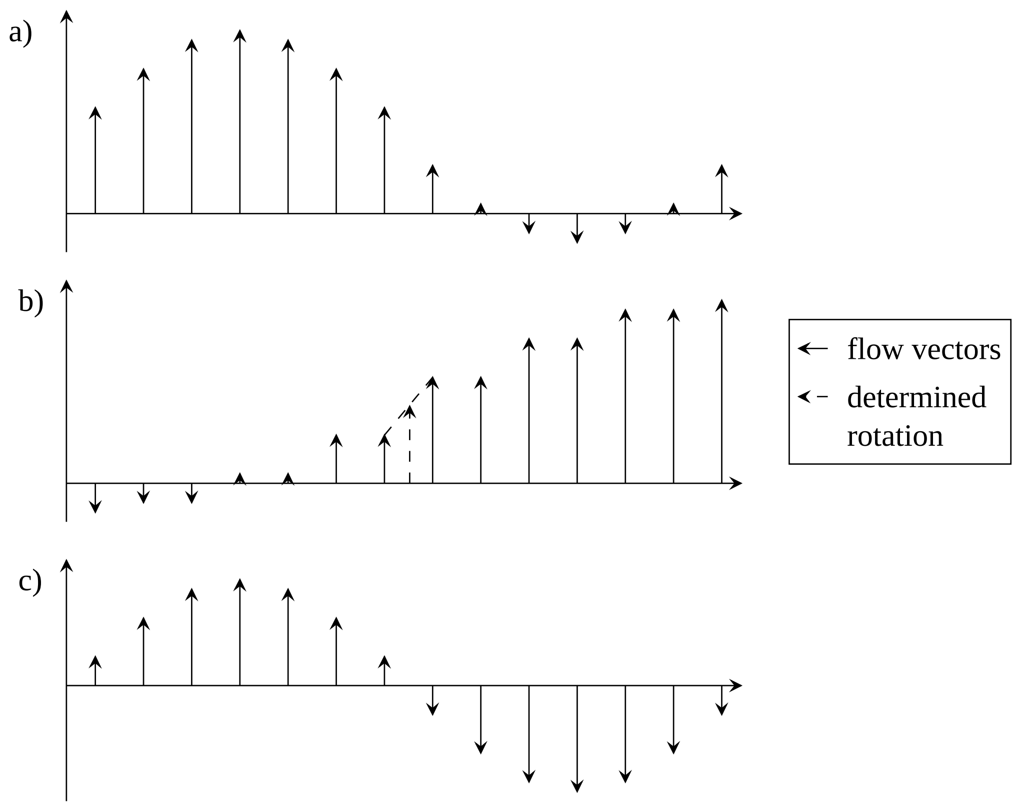

Figure 2.13: Illustration of solution 2. a) The arrows indicate a flow field of translation and rotation. b) The flow vectors are sorted to calculate the rotation from the two centered vectors. c) The flow field of (a) with subtracted rotation.

2.3.2 Determination of the Distance

The *distance* between two images is the size of the translational flow field. It corresponds to the spatial distance between the positions where the two pictures have been taken but it is not a metrical measurement. Instead, it depends on the detachment to the surroundings.

After the rotation has been determined, it must be eliminated from the flow in order to calculate the translational components of the flow field. The rotation ω is simply subtracted from the flow Y'' to get the translational flow field Y''':

$$\begin{gathered} Y''' = \begin{pmatrix} y_0''' & \cdots & y_{n-1}''' \end{pmatrix} \\ \bigwedge_{i=0}^{n-1} y_i''' = y_i'' \dot{-} \tfrac{n}{2\pi}\omega \end{gathered} \tag{2.22}$$

The distance φ between the two images is calculated by summing up the squares of all flow vectors:

$$\varphi = \sum_{i=0}^{n-1} (y_i''')^2 \tag{2.23}$$

2.3.3 Calculation of the Bearing Direction

The bearing direction δ is the direction from the position where picture I has been taken to the position where image J has been taken. δ is relative to picture I or—more precisely—to the direction that the pixel p_0^I is heading to. The bearing direction is the FOE of the flow field between the two images (cf. figure 2.14a). There are several possibilities how δ can be determined from Y'''.

Solution 1

The optical flow field consists of n vectors. The direction of the flow vectors is always perpendicular to their position in the picture (cf. figure 2.14a). Their signs determine whether they point in clockwise or counterclockwise direction. In the first approach, each individual flow vector *votes* for a bearing direction. As shown in figure 2.14a all flow vectors point away from the searched direction. So a single flow vector divides the image in two halves: the part that is in the direction the vector points to and the half that is in the opposite direction (cf. figure 2.14b). The FOE is somewhere in the second half. As each vector can only vote for one position, the center of this half is used. This voting direction is exactly the opposite of the direction the vector is pointing to. Therefore it is perpendicular to the position of the flow vector in the picture as well as the flow vector itself. It can be calculated by adding a quarter of the picture's size n to each vector's position. The sign of the added offset depends on the sign of the corresponding vector:

$$
\begin{aligned}
V &= \begin{pmatrix} v_0 & \cdots & v_{n-1} \end{pmatrix} \\
\bigwedge_{i=0}^{n-1} v_i &= \begin{cases} \begin{pmatrix} \cos(\alpha_i) \\ \sin(\alpha_i) \end{pmatrix} & \text{if } y_i''' \neq 0 \\ \begin{pmatrix} 0 \\ 0 \end{pmatrix} & \text{if } y_i''' = 0 \end{cases} \\
\text{where} \quad \alpha_i &= 2\pi \frac{i}{n} + \begin{cases} \frac{\pi}{4} & \text{if } y_i''' < 0 \\ -\frac{\pi}{4} & \text{if } y_i''' > 0 \end{cases}
\end{aligned}
\tag{2.24}
$$

All the votes are added up as two-dimensional direction vectors. The direction of the resulting vector is used as the desired bearing direction δ:

$$\delta = \arctan \sum_{i=0}^{n-1} v_i \tag{2.25}$$

Although most vectors vote for the wrong direction, the bearing direction δ is determined correctly because of the following considerations: if the optical flow has been calculated error-free for every vote vector v_i, the vectors can be paired. For every vector in one hemisphere there is always a vector in the other hemisphere that completes the first vector—by simple vector addition—to a vector that points in the FOE's direction. Therefore, every pair of vectors together votes for the correct direction.

This first solution is quite similar to the approach of Hong *et al.* (1991) but they used only few flow vectors to calculate the bearing direction and therefore they needed a lot of further processing to improve the precision of the calculation and to compensate errors.

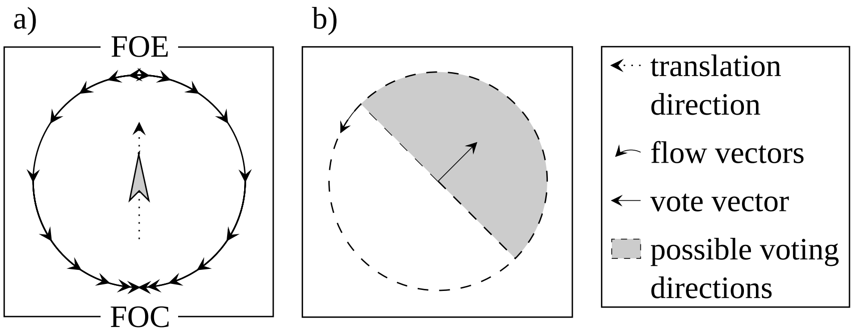

Figure 2.14: Illustration of flow fields and vote vectors. a) Translational flow with positions of FOE and FOC. b) One flow vector together with the possible voting directions and the chosen vote vector.

Solution 2

The previous approach assumes that all flow vectors have been calculated correctly. But if the images are noisy, some flow vectors may point in the wrong direction, especially the small vectors near the FOE and FOC. As a result, some of the vector-pairs, which have been discussed in the previous section, do not point in the direction of the FOE and therefore the bearing direction is calculated with some error. To get around this problem, a refinement is to weight the votes v_i to strengthen large flow vectors and to suppress small vectors in the calculation of the bearing direction (cf. figure 2.15). This reduces the influence of the small vectors near the FOE and FOC:

$$\delta = \arctan \sum_{i=0}^{n-1} |y_i'''| \, v_i \tag{2.26}$$

This refinement has a drawback. If the flow is large in one small portion of the flow field, the FOE may be calculated incorrectly because the vote vectors in this small section are weighted too heavily.

Solution 3

A different approach to calculate the bearing direction is to divide the flow into two halves, a clockwise (negative) and a counterclockwise (positive) flow. The connecting points between these two parts are the FOE and the FOC—the position of the FOE corresponds to the searched direction.

A straightforward approach is to search for one of these two sections. One feasible way is to seek for a continuous section with all vectors pointing in the same direction,

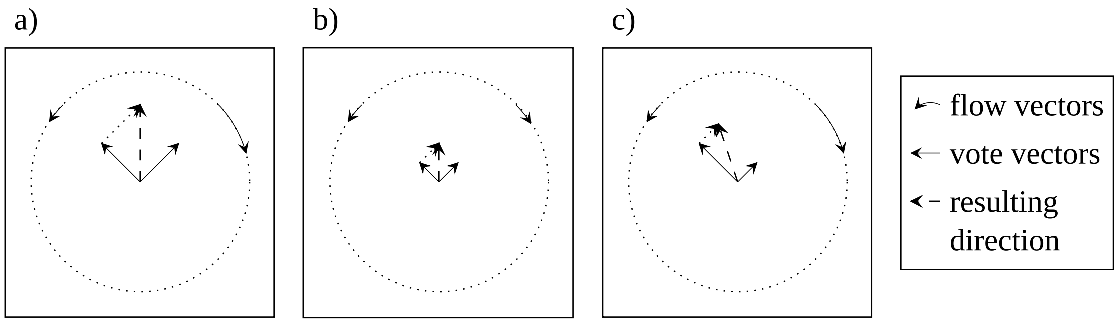

Figure 2.15: Illustration of the effect of weighting the vote vectors. a) Different flow on both sides is ignored in solution 1. b) Solution 2 with equal flow on both sides. c) Solution 2 with a remaining rotational flow. The rotation is nearly compensated in the calculation of the direction.

but because of possible noise, it is better to search for a section, which contains exactly half of the flow field's vectors and which has the smallest or biggest sum of the containing vectors. In this implementation, the smallest sum is used:

$$\begin{aligned} h_i &= \sum_{i \dot{-} \frac{n}{2} \dot{+} 1}^{i} y_j''' \\ h_{i_{min}} &= \min_{i=0}^{n-1} h_i \end{aligned} \tag{2.27}$$

As shown in figure 2.14a on page 36, the section with the negative flow is—viewed in counterclockwise direction–before the FOE and the section with positive flow is behind. The bearing direction is somewhere between the last flow vector of the positive half and the first vector of the negative half. For simplicity, the middle between the positions of these two vectors is used. So the FOE and thereby the bearing direction δ is calculated as

$$\delta = \frac{2\pi}{n} \left(i_{min} \ddot{+} \frac{1}{2} \right) \tag{2.28}$$

Solution 4

The method presented in the previous section assumes that the bearing direction δ is centered between the two adjacent flow vectors. Therefore, it only calculates the bearing direction with a maximum precision that is equal to the angle between two neighboring pixels. It is possible to improve the precision by calculating an offset Δg that determines the position of the real bearing direction between the two adjacent flow vectors. This replaces the static offset $\frac{1}{2}$ in equation (2.28):

$$\delta = \frac{2\pi}{n} \left(i_{min} \ddot{+} \Delta g \right) \tag{2.29}$$

Equation (2.27) calculates the last vector's index i_{min} of the flow field's "negative half". If the indices are viewed as real numbers, the real end i'_{min} of the negative half is somewhere between $i_{min} \dot{-} 1$ and $i_{min} \dot{+} 1$. The bearing direction is at $i'_{min} \dot{+} \frac{1}{2}$ because this is the middle between i'_{min} and the position $i'_{min} \dot{+} 1$ of the first flow vector of the positive half.

i'_{min} is calculated from i_{min} and one of its two neighbors. i_{min} represents the half that has the smallest flow field sum. Therefore, it seems to be sensible to use i_{min} and the index that represents the smaller sum of the two neighbors, to determine i'_{min}:

$$\Delta g = \begin{cases} \frac{1}{2} - \Delta g'_{i_{min}, i_{min} \dot{-} 1} & \text{if } h_{i_{min} \dot{+} 1} < h_{i_{min} \dot{-} 1} \\ \frac{1}{2} + \Delta g'_{i_{min}, i_{min} \dot{+} 1} & \text{if } h_{i_{min} \dot{+} 1} > h_{i_{min} \dot{-} 1} \end{cases} \tag{2.30}$$

The offset between two neighboring minimal halves is calculated by interpolating between the borders of these halves. The connections between each border and its neighbor are interpolated linearly. If they intersect, their intersection point is used as the offset $\Delta g''_{i,j}$. If they do not intersect, $\frac{1}{2}$ is used as the offset instead. Figure 2.16 illustrates the possible two cases.

$$\Delta g''_{i,j} = \begin{cases} \frac{y'''_{i \dotminus \frac{n}{2} \dotplus 1} - y'''_i}{c} & \text{if } c \neq 0 \\ \frac{1}{2} & \text{if } c = 0 \end{cases} \qquad (2.31)$$
$$\text{where} \quad c = y'''_j - y'''_i + y'''_{i \dotminus \frac{n}{2} \dotplus 1} - y'''_{j \dotminus \frac{n}{2} \dotplus 1}$$

The offset $\Delta g'_{i,j}$ should be always in the range $]-1 \ldots 1[$ because i'_{min} should be between $i_{min} \dotminus 1$ and $i_{min} \dotplus 1$. If the intersection is outside this range, $\frac{1}{2}$ is used as the offset instead because the calculation of $\Delta g''_{i,j}$ is invalid for this range:

$$\Delta g'_{i,j} = \begin{cases} \Delta g''_{i,j} & \text{if } \left|\Delta g''_{i,j}\right| < 1 \\ \frac{1}{2} & \text{if } \left|\Delta g''_{i,j}\right| \geq 1 \end{cases} \qquad (2.32)$$

2.4 Computational Complexity

The runtime performance of algorithms is important, especially if they should be used in a real-time environment. Therefore, the algorithm's complexity has to be examined.

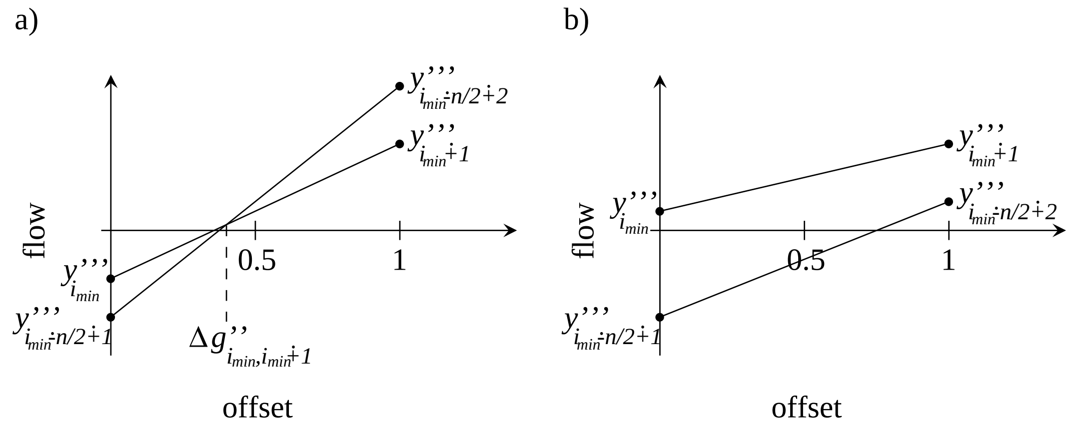

Figure 2.16: Illustration of the calculation of $\Delta g''_{i,j}$ in equation (2.31). a) There is an intersection between the two lines. b) The two interpolated lines do not intersect inside the range.

Panama's major time-relevant parameters are the number of pixels n and the number of adaptation steps t_{max}. The algorithm's complexity depends on how it is implemented, sequentially or in a parallel way. If it is a parallel implementation, the complexity also depends on the abilities of the underlying hardware, especially whether it allows to compute sums and minima in constant time—as analog hardware does. So the complexity analysis is divided into three categories:

1. The sequential implementation.

2. The parallel implementation on normal multiprocessor systems. This hardware allows to process independent calculations at the same time and therefore it reduces time consumption. On the other hand, the demand for hardware is much higher. A parallel implementation does not reduce the algorithm's complexity but it allows to divide it into space (number of processors) and time. As the algorithmic complexity is the same as in a sequential implementation, the complexity concept that is used in this work considers only the runtime. It is assumed that there are always enough processors to calculate as much in parallel as possible.

3. The parallel implementation with the ability to calculate sums and minima of multiple numbers in one step. With conventional parallel hardware, sums and minima can only be calculated in $\mathrm{O}(\log n)$. In contrast, analog hardware allows to compute these functions in constant time. This further reduces the algorithm's runtime.

The following discussion analyses the different parts of the *Panama* algorithm for these three implementation scenarios.

2.4.1 Correlation Matrix

The correlation matrix has $n \times n$ entries. Each entry can be calculated in constant time. Therefore, the expenditure to determine the whole matrix is $\mathrm{O}(n^2)$. All of the matrix's elements are independent of each other. Thus it is possible to calculate them all in a parallel way. This reduces the complexity to $\mathrm{O}(1)$ but increases the number of parallel processes to n^2.

2.4.2 Best Match

The adaptation is done in t_{max} steps. Each step searches for the best matching entry in one row of the correlation matrix. This is done by finding the element with the smallest distance to a selected pixel. In the beginning of the matching process, the whole row of n entries is searched. While the process runs, the search range decreases down to one element. For each element in the search range, the distance is calculated. The result is compared once to find the minimum. Although not all elements are searched in each step, the search complexity is still $O(n)$ in a sequential implementation. As the search is done t_{max} times, the overall complexity is $O(t_{max}n)$.

On parallel hardware, all distances can be calculated at the same time. If the hardware is not able to determine the minimum in one step, it can find it hierarchically with a complexity of $O(\log n)$: in the first step, all distances are divided into pairs and the minimum of each pair is determined. These $\frac{n}{2}$ calculations can be done at the same time. Then, all these minima are paired and their minima are found in one step and so on. Again, the search is done t_{max} times. This results in a complexity of $O(t_{max} \log n)$ on normal parallel hardware and $O(t_{max})$ if the hardware calculates the minimum in one step.

2.4.3 Adaptation Step

In each adaptation step, $2\sigma_t - 1$ elements of Y are changed. Over the whole process, σ_t decreases from $\frac{n}{2} - 1$ to 1. As in the previous section, the sequential complexity of one adaptation step is $O(n)$. The complexity of all steps together is $O(t_{max}n)$. As the calculations of Y's elements are independent of each other, they can occur in a parallel way on a suitable hardware. This reduces the complexity of all adaptation steps together to $O(t_{max})$.

2.4.4 Optical Flow

The determination of the optical flow is divided into three parts:

1. The simple vector addition and subtraction in equations (2.15) and (2.16) has a sequential complexity of $O(n)$ or can be calculated totally in parallel.

2. The minimum search in equation (2.17) has the same complexity as mentioned in section 2.4.2.

3. The determination of the offset vector in equation (2.18). In this recursive function, the three different cases are independent of each other but each case has to be calculated sequentially. Therefore, the complexity is $O(n)$ on every platform.

Altogether, the optical flow is always calculated in $O(n)$ steps because the determination of the offset vector has this complexity.

2.4.5 Rotation

Section 2.3.1 beginning on page 32 has presented two different approaches to calculate the rotation. The first solution simply adds up n numbers. The complexity of building the sum of n elements is as large as finding the minimum in the three different implementation scenarios. The second approach sorts n numbers. One of the fastest sort algorithms is MergeSort (Bird and Wadler 1988). It recursively divides the data into two equal halves until each half consists only of one element. Then, it fits the elements together in the correct order. First it merges the n elements to $\frac{n}{2}$ sorted two-element-sequences. Then, it repeats the process to get $\frac{n}{4}$ sorted four-element-sequences from these $\frac{n}{2}$ two-element-sequences and so on. The algorithm runs through these merging levels until all elements have been fitted together into one sequence. This sequence is the sorted list. As each level merges half of the sequences than the previous one does, $\log_2 n$ levels are needed for the whole process. As the sequences processed by each level are twice as long as they are in the previous one, all levels combine the same number of elements. So, each of the $\log_2 n$ levels merges n elements to sort the vector. This results in a complexity of $\mathrm{O}(n \log n)$.

The merging of two sorted sequences into one is a sequential operation. All these sequential operations on the same level are independent of each other. Therefore, they can be processed at the same time on parallel hardware. Hence, each level takes only as long as one sequential merge operation at this level. Thus the first level takes the time to combine 2 elements, the second to merge 4 elements, the third fits 8 elements together and so on. The last level merges all elements. The resulting complexity for a parallel implementation is

$$\begin{aligned} &\mathrm{O}\left(2+4+8+\ldots+n\right) \\ &\approx \mathrm{O}\left(2n\right) \\ &= \mathrm{O}\left(n\right) \end{aligned} \tag{2.33}$$

2.4.6 Distance

Equation (2.22) determines the translational flow field by subtracting the rotation from all elements of the flow vector. The distance between the pictures is summed up in equation (2.23). Both operations have a complexity of $\mathrm{O}(n)$ in the sequential case. Equation (2.22) can always be calculated in one step on parallel hardware. As discussed in section 2.4.2, the sum in equation (2.23) can be determined in $\mathrm{O}(\log n)$ steps on normal parallel computers or in one step if the hardware is able to calculate it by itself.

2.4.7 Bearing Direction

Four different solutions have been presented to determine the bearing direction in section 2.3.3 beginning on page 35:

- Solutions one and two simply sum up all elements of the vector. The complexity of this operation has been discussed previously.

- Solutions three and four calculate the minimum of sums that add up half of the flow field. A straightforward implementation would have a complexity of $O(n^2)$ but an optimization is possible. Only one sum has to be calculated completely. All other sums can be determined by adding the difference to their predecessor because two sums with adjacent indices differ only in two elements:

$$h_{i+1} = h_i + y'''_{i+1} - y'''_{i-\frac{n}{2}+1} \tag{2.34}$$

  This reduces the sequential complexity to $O(n)$. For parallel implementations, it is better to calculate the sums as in the straightforward approach because all sums can be computed at the same time. This results in a computational complexity of $O(\log n)$ for the normal parallel calculation and in $O(1)$ if the sums and the minimum can be calculated in one step.

2.4.8 Overall Complexity

The overall complexity of *Panama* is equal to the maximum complexity of its algorithmic parts. If t_{max} is assumed to be greater than n, this is the calculation of the best matching element. Therefore, the sequential complexity is $O(t_{max}n)$ and the parallel complexity is $O(t_{max} \log n)$. If the hardware is able to compute sums and minima in one step, the complexity is reduced to $O(t_{max})$. Table 2.1 summarizes the results of the complexity analysis.

	sequential	parallel	parallel min/max
correlation matrix	n^2	1	1
best match	$t_{max}n$	$t_{max} \log n$	t_{max}
adaptation	$t_{max}n$	t_{max}	t_{max}
optical flow	n	n	n
rotation (solution 1/2)	n / $n \log n$	$\log n$ / n	1 / n
distance	n	$\log n$	1
direction	n	$\log n$	1
overall	$t_{max}n$	$t_{max} \log n$	t_{max}

Table 2.1: The complexity of the *Panama* algorithm in sequential and parallel implementations. In addition, parallel implementations that calculate sums and minima in one step and implementations that do not are distinguished.

2.5 Experiments and Results

In this section, *Panama*'s performance in two simulated scenes is analyzed. During the description of the image processing method, multiple approaches have been presented for several implementation aspects and some details have been completely left open. Many of these alternatives will be compared below:

1. The best way to calculate the weight vectors W^I. There are so many ways how these vectors can be generated that only a few possibilities were examined:
    - The best method to calculate derivatives.
    - How many derivatives are needed.
    - Whether the use of smoothing filters improves the performance.
2. How the image resolution n influences the results.
3. How many adaptation steps t_{max} are necessary.
4. Which of both solutions that determine the rotation ω performs better.
5. Which of the four solutions that calculate the bearing direction δ is superior.

All these alternatives have to be compared under different general conditions:

- Different positions where the pictures were taken.
- Different surroundings.
- Different levels of noise.

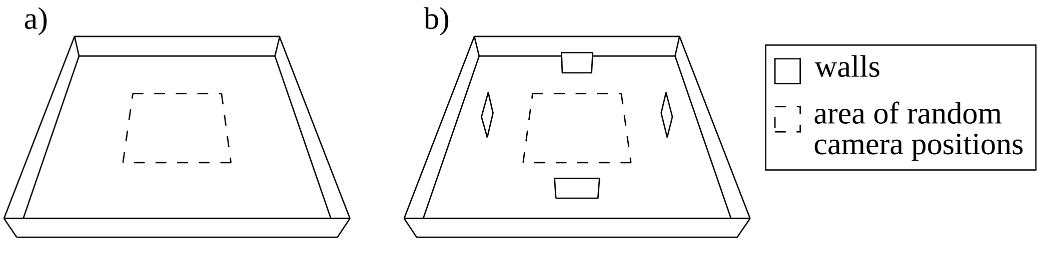

Figure 2.17: The simulated rooms used for the experiments. a) A room without obstacles. b) A room with four obstacles.

As it is impossible to compare all algorithmic combinations, the effect of each individual parameter was analyzed separately. All investigations used the same parameter set and only varied the examined parameter. Each alternative was tested with fifteen different levels of noise l from 2^0 down to 2^{-14}. The frequency distribution of the noise was defined by a cosine function in the range from $-\pi$ to π that was scaled down to the range of $[-l \dots l]$ (cf. figure 2.18). This is the model of noise implemented in SimRobot (Siems *et al.* 1994):

$$p_x^l = \frac{1 + \cos \frac{\pi x}{l}}{2} \tag{2.35}$$

Panama was applied to 100 pairs of images for each noise level. All tests were performed in two different simulated rooms (cf. figure 2.17). Both rooms are of quadratic shape. Each of the rooms' walls has a different color with a continuously changing brightness. All pictures were taken in an area around the center with a size of $\frac{1}{9}$ of the room's base area. The second room has four additional walls. These walls function as obstacles because they introduce a second layer of depth into the scene. This three-dimensionality impedes the matching process because some regions are exclusive to one of the images (cf. figure 2.19). Besides, these walls are closer to the area of possible camera positions. This results in greater distortions between the pictures, i.e., an obstacle was able to take up a region in one image that was up to three times larger than in another image.

While the experiments in the first room show the performance of *Panama* under optimal conditions, the tests in the second room are better to indicate the algorithm's abilities in working with real images because most surroundings contain three-dimensional structures, e.g. closets, columns, or plants.

The parameter set that was used as standard is:

- The filter D_1 (described in the next section) was used to calculate derivatives.

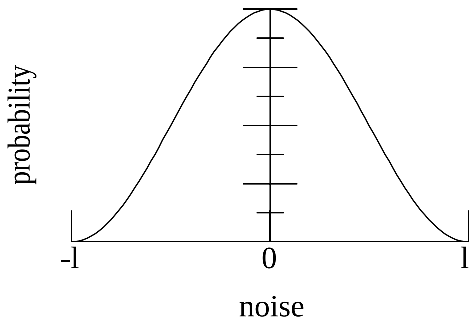

Figure 2.18: The distribution of noise in the range of $[-l \dots l]$.

- The weight vectors have contained the original picture and one derivative of each of its color channels. Only if smoothing was analysed, the derivatives were omitted.
- No smoothing was used.
- The image resolution was 256 pixels.
- The matching was done in 4096 adaptation steps.
- Solution 2 was used for the calculation of the rotation.
- Solution 4 was used to determine the bearing direction.

All results are presented graphically. For each analysed parameter, a set of six charts shows the results of the experiments in room A and a second set shows the results for room B. The various alternatives are judged by comparing the average and the maximum errors between the correct and the calculated values. For each analyzed parameter, the charts show the average and maximum error of the optical flow, the calculated rotation and the determined bearing direction. As the optical flow is independent of the way how the rotation and the bearing direction are calculated, it is left out in the analysis of these two parameters. The distance value that *Panama* calculates is directly determined from the optical flow. Therefore, it is not necessary to analyze it separately.

The six charts are of different importance. The average error is the inaccuracy that can be expected under "normal" conditions. The maximum error describes the worst-case behavior of the algorithm. As *Panama* should work in any case (including the worst-case), the maximum error is the better quantity to judge the performance. As it is the aim of *Panama* to determine the rotation and the direction, the correctness of the optical flow field is of minor significance unless it does not influence the precision of the other two

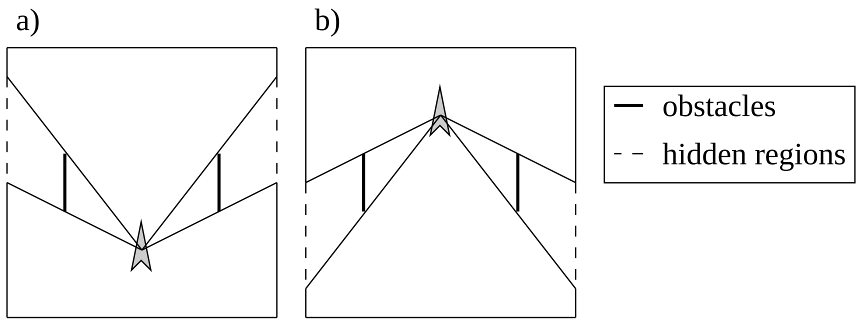

Figure 2.19: Illustration of the influence of obstacles on panoramic pictures. The obstacles hide different parts of the environment when viewed from different positions. Therefore, the image seen from a) cannot be transformed into picture b).

values. Especially in room B, the flow field *cannot* be calculated correctly because the two matched pictures do not contain the same information (cf. figure 2.19). If *Panama* is used in a scenario that does not need the calculation of a rotation, e.g. if controlling a holonomic mobile system, the precision of this parameter may be of no importance. In contrast, if the algorithm is used for steering a non-holonomic vehicle, e.g. a wheelchair, the rotation is of major consequence because it is an important factor in the navigation of kinematically restricted vehicles. Therefore, the significance of the errors in the determination of the rotation and the direction depends on the tasks that should be solved with the help of *Panama*.

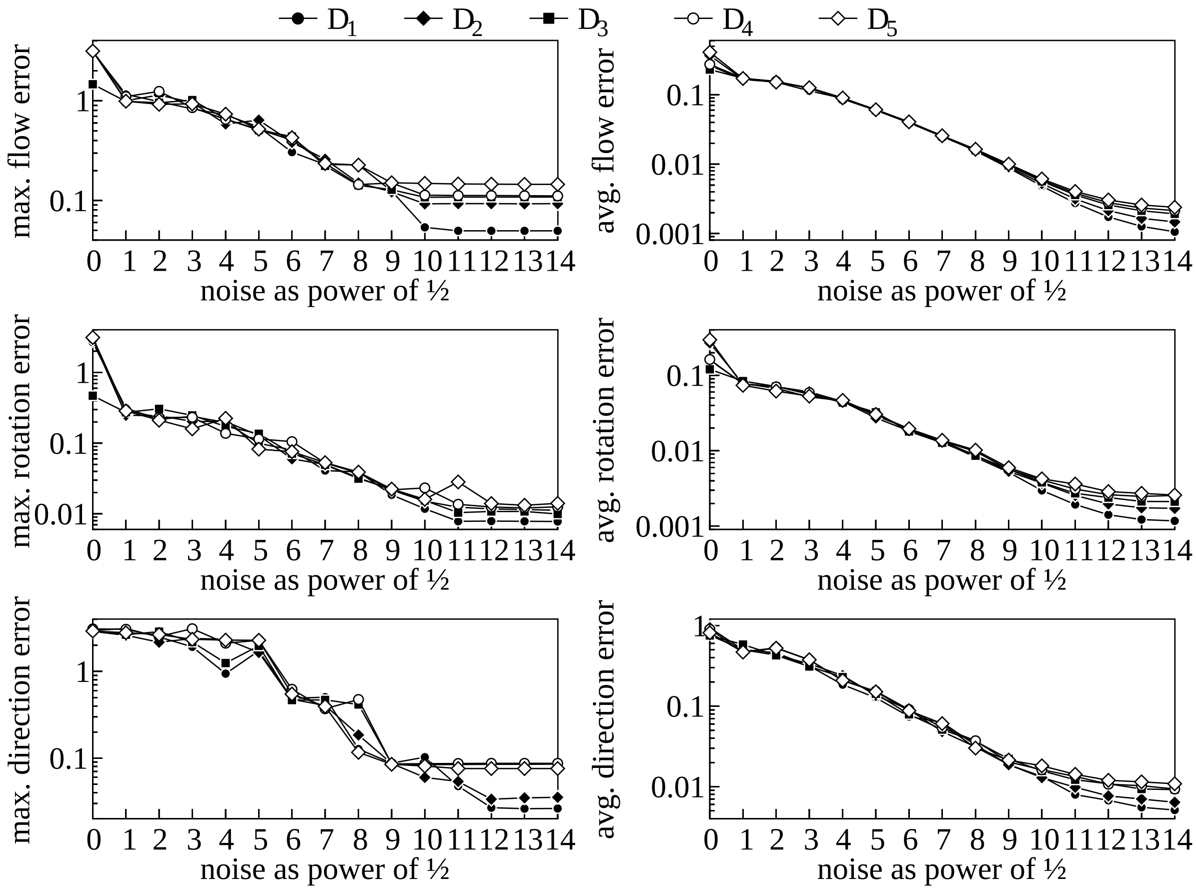

Figure 2.20: Results of the analysis of different derivative filters in room A.

2.5.1 Derivative Filters

If derivatives are used to generate the weight vectors W^I in section 2.2.1 beginning on page 25, there are multiple ways to calculate these derivatives. One possibility is to utilize *derivative filters* (Wittmann 1996). These filters use the neighbors of a pixel p_i to determine p_i's gradient. The different filters D_r vary in the number of adjacent pixels that are included in the calculation and in the weighting of these pixels:

$$D_r = \begin{pmatrix} 0 & -d_1 & -d_2 & \cdots & -d_r & 0 & \cdots & 0 & d_r & \cdots & d_2 & d_1 \end{pmatrix} \tag{2.36}$$

These filters are applied to each color channel separately. p_i's gradient is calculated by the inner product between the filter and each color channel of the image. Before this, the elements of the color channel are rotated so that the inner product multiplies d_0 with p_i, d_1 with p_{i+1} and so on. The filters that were analyzed are:

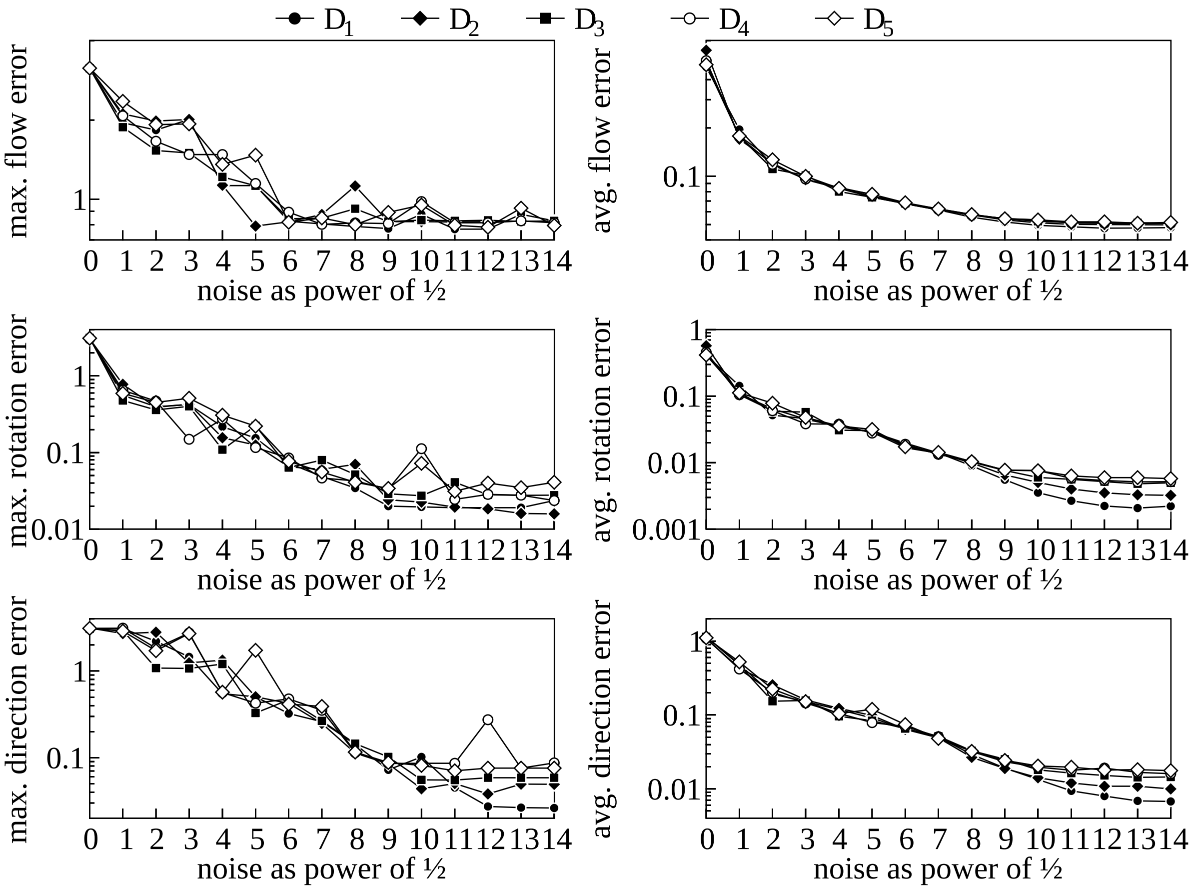

Figure 2.21: Results of the analysis of different derivative filters in room B.

$$D_1 = \frac{1}{2} \begin{pmatrix} 0 & -1 & 0 & \cdots & 0 & 1 \end{pmatrix} \tag{2.37}$$

$$D_2 = \frac{1}{12} \begin{pmatrix} 0 & -8 & 1 & 0 & \cdots & 0 & -1 & 8 \end{pmatrix} \tag{2.38}$$

$$D_3 = \frac{1}{60} \begin{pmatrix} 0 & -45 & 9 & -1 & 0 & \cdots & 0 & 1 & -9 & 45 \end{pmatrix} \tag{2.39}$$

$$D_4 = \frac{1}{840} \Big(0 \quad -672 \quad 168 \quad -32 \quad 3 \quad 0 \quad \cdots$$
$$\cdots \quad 0 \quad -3 \quad 32 \quad -168 \quad 672 \Big) \tag{2.40}$$

$$D_5 = \frac{1}{2520} \Big(0 \quad -2100 \quad 600 \quad -150 \quad 25 \quad -2 \quad 0 \quad \cdots$$
$$\cdots \quad 0 \quad 2 \quad -25 \quad 150 \quad -600 \quad 2100 \Big) \tag{2.41}$$

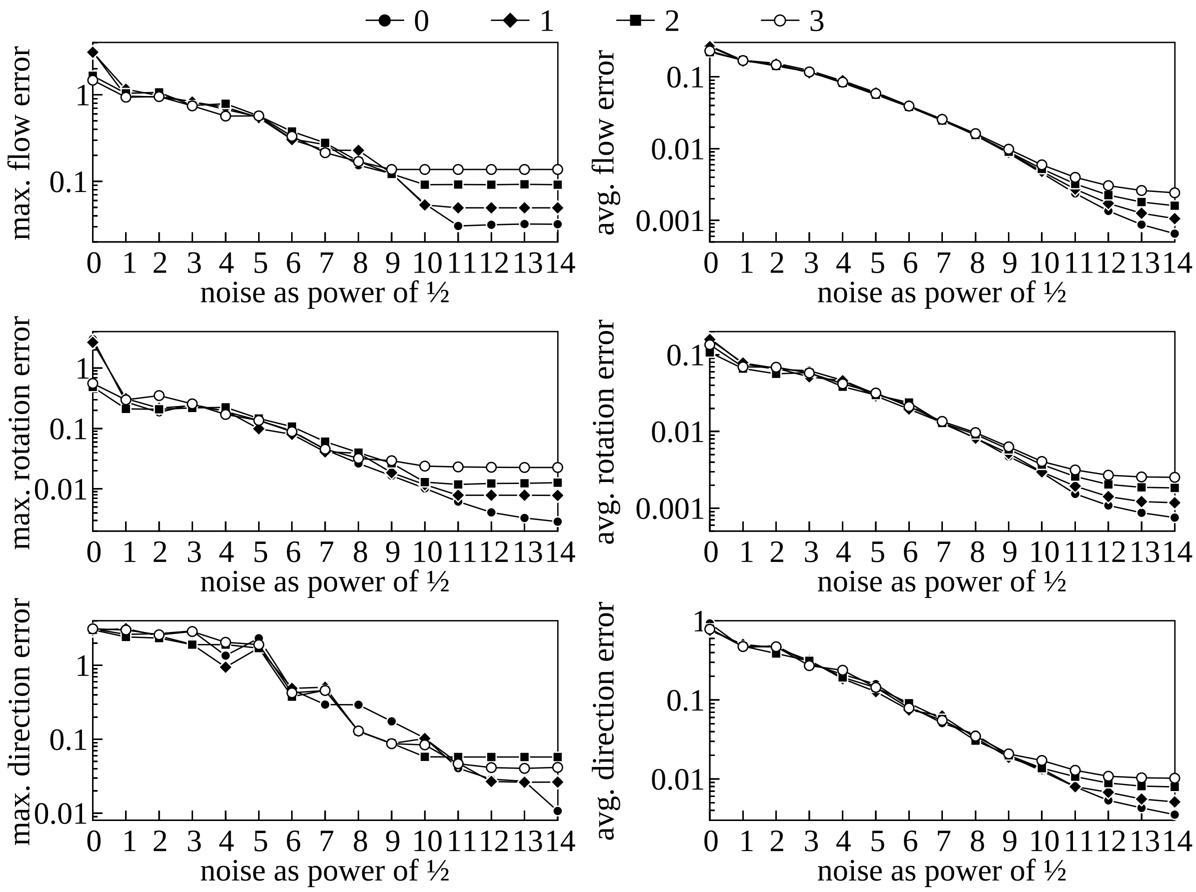

Figure 2.22: Results of the analysis of different numbers of derivatives in room A.

The results that are shown in the figures 2.20 and 2.21 indicate that it is best to use the most simple derivative filter D_1. In both rooms, it was superior under low noise conditions. With high noise, it is unimportant which filter is used. It seems that D_1 was superior because the smaller the filter was, the more stable the derivatives remained if picture regions shrank or expanded during translational movements.

An interesting fact is that the error in flow calculation in room B was dominated by the errors that were caused by the obstacles whereas the used derivative filter did not affect the average and maximum flow errors. Nevertheless, the derivative method influenced the calculation of the optical flow because there were differences in the determination of the rotation and the bearing direction.

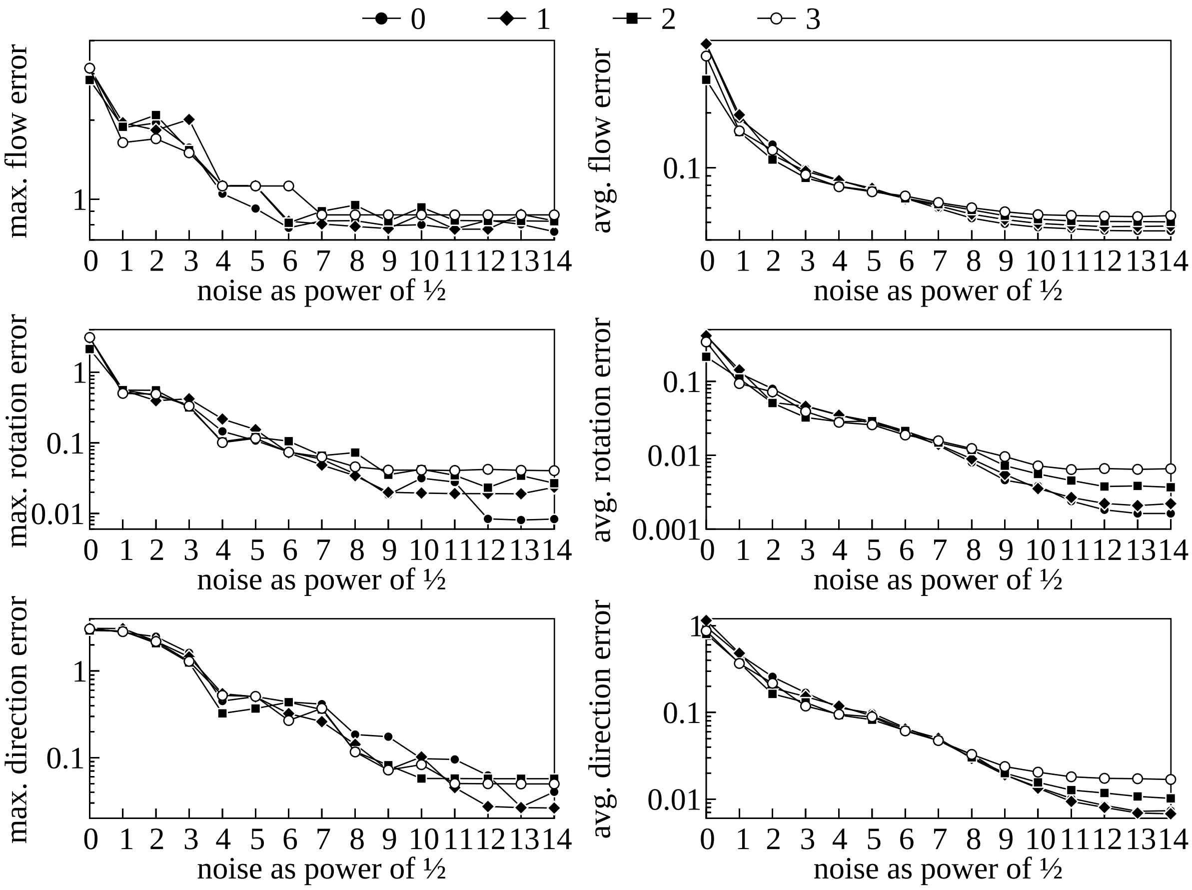

Figure 2.23: Results of the analysis of different numbers of derivatives in room B.

2.5.2 Derivative Count

The simplest way to define a weight vector W^I is to directly use the image I. As an alternative, it is possible to take one or more derivatives of the picture. Another way is to combine the original image with its derivatives. The latter possibility was used in the experiments. The previous section has shown that the filter D_1 performed best in the determination of derivatives. In this section, it is analyzed how many derivatives are suitable. Experiments without and with up to three derivatives were performed.

The charts in figure 2.22 show that all results but the maximum directional error suggest that it is best not to use any derivative in addition to the original picture in room A. Omitting the derivatives was only inferior in the calculation of the bearing direction with a noise of $[2^{-8} \ldots 2^{-9}]$. In room B, the average rotation and bearing directional errors nearly was identical for using none or one derivative but the maximum directional errors were much greater without derivatives (cf. figure 2.23). Therefore, it is better to use one

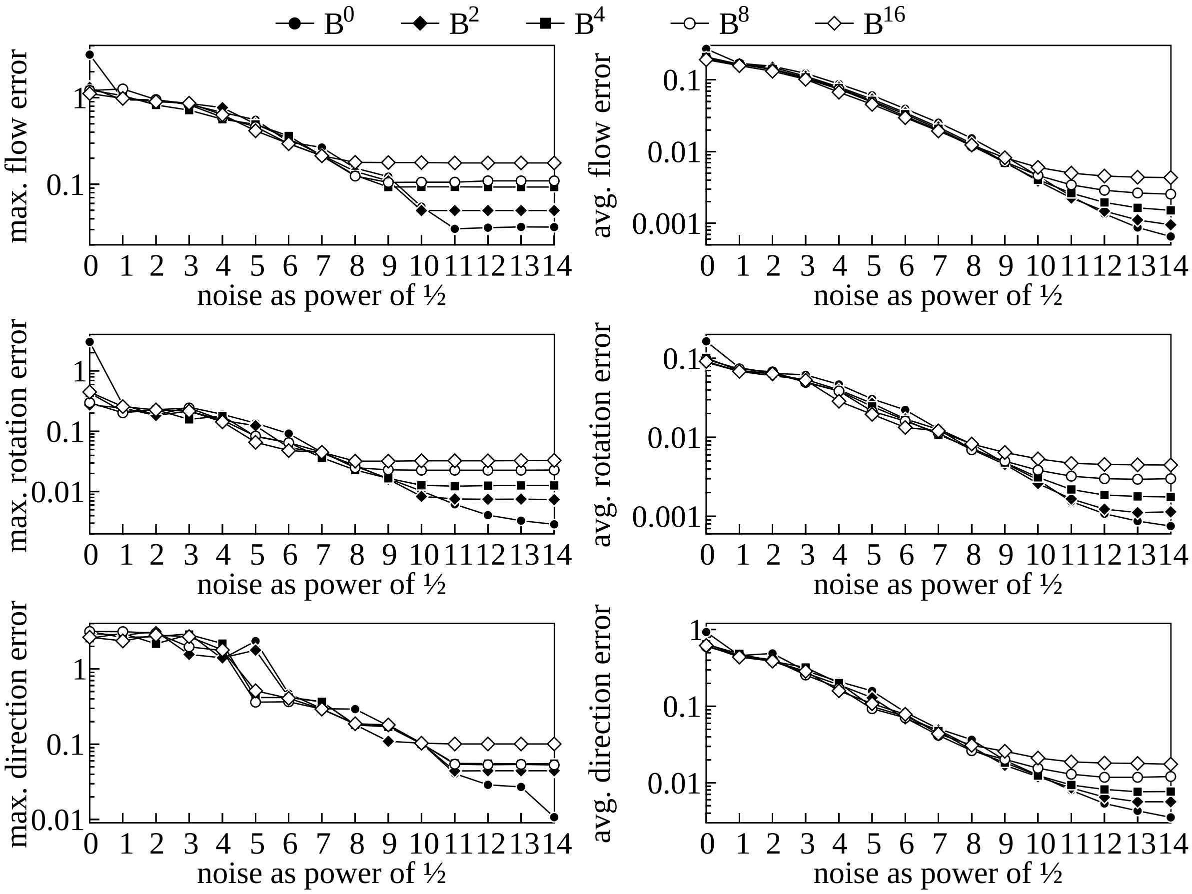

Figure 2.24: Results of the analysis of different smoothing filters in room A.

derivative in the calculation of the weight vectors if the surroundings are structured.

2.5.3 Smoothing Filters

Smoothing filters can be used to reduce noise or to prepare an image for subsampling (Wittmann 1996). They eliminate small signal frequencies from the picture. One possibility to implement these filters are *binomial filters*. As with the derivative filters, there are many binomial filters B^r that differ in the number of adjacent pixels they use:

$$\begin{gathered} B^r = \begin{pmatrix} b_0^r & b_1^r & \cdots & b_{\frac{r}{2}}^r & 0 & \cdots & 0 & b_{-\frac{r}{2}}^r & \cdots & b_{-2}^r & b_{-1}^r \end{pmatrix} \\ \bigwedge_{k=-\frac{r}{2}}^{\frac{r}{2}} b_k^r = \frac{r!}{2^r\left(\frac{r}{2}-k\right)!\left(\frac{r}{2}+k\right)!} \end{gathered} \tag{2.42}$$

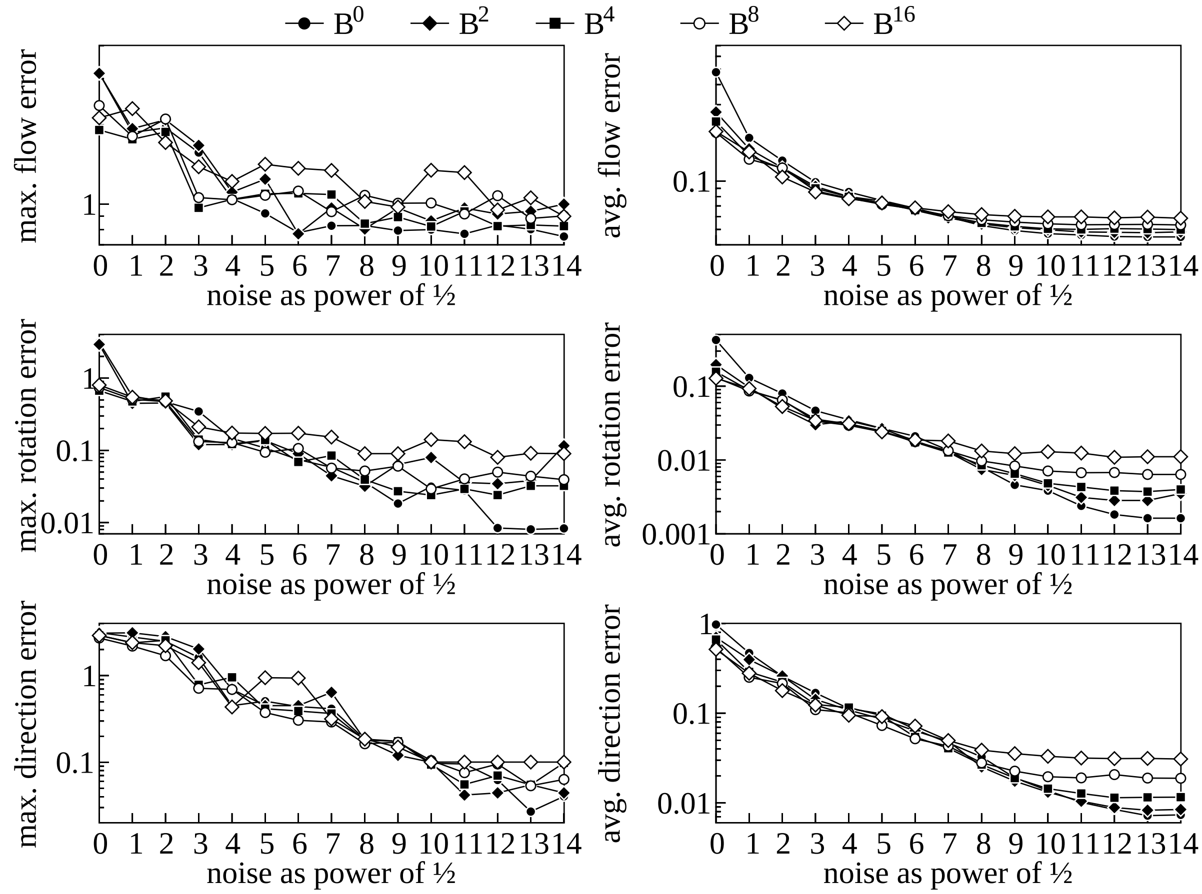

Figure 2.25: Results of the analysis of different smoothing filters in room B.

The use of the filters B^0 (no filter), B^2, B^4, B^8, and B^{16} was tested. The results plotted in the figures 2.24 and 2.25 do not indicate that applying one of these filters improves the matching process. The main reason for these results is the same as in section 2.5.1 beginning on page 48: if there is a translational movement between the two matched images, some of the regions shrink or expand. Therefore, the same regions are represented by a different number of pixels in both pictures. As the smoothing filters use a constant number of neighboring pixels to calculate the smoothed image, the smoothed pictures are more different than the original ones. This eliminates the improvements that smoothing filters achieve in noise reduction.

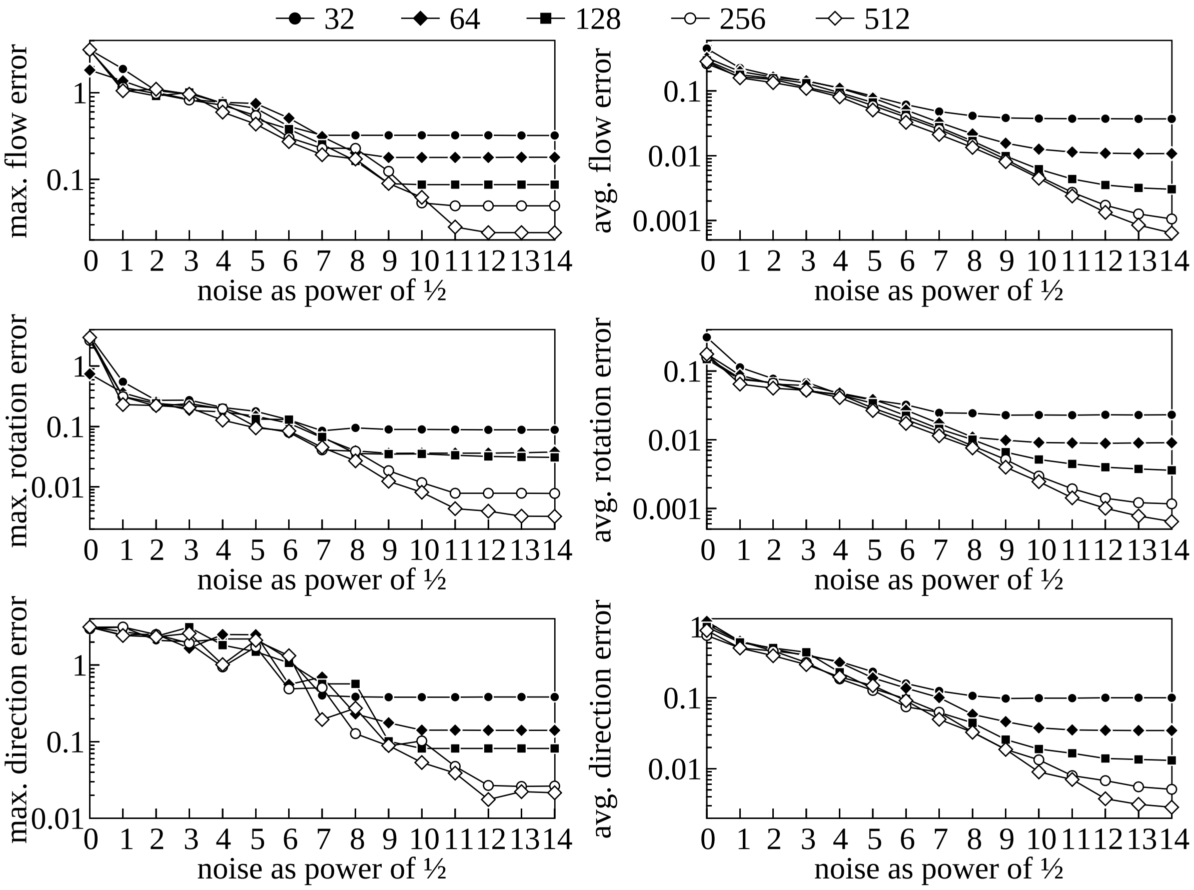

Figure 2.26: Results of the analysis of different image resolutions in room A.

2.5.4 Image Resolution

The image resolution is an important parameter, especially for the runtime of the algorithm. In a sequential implementation, the computational complexity is at least $O(n^2)$. Therefore, it is interesting to know whether the improvements in accuracy of higher resolutions justify the additional computational costs. *Panama* was tested with resolutions of 32, 64, 128, 256, and 512 pixels. The results that are plotted in the figures 2.26 and 2.27 show that the higher the resolution, the smaller the errors. Therefore, for each individual implementation, a compromise has to be found between the runtime and the required precision.

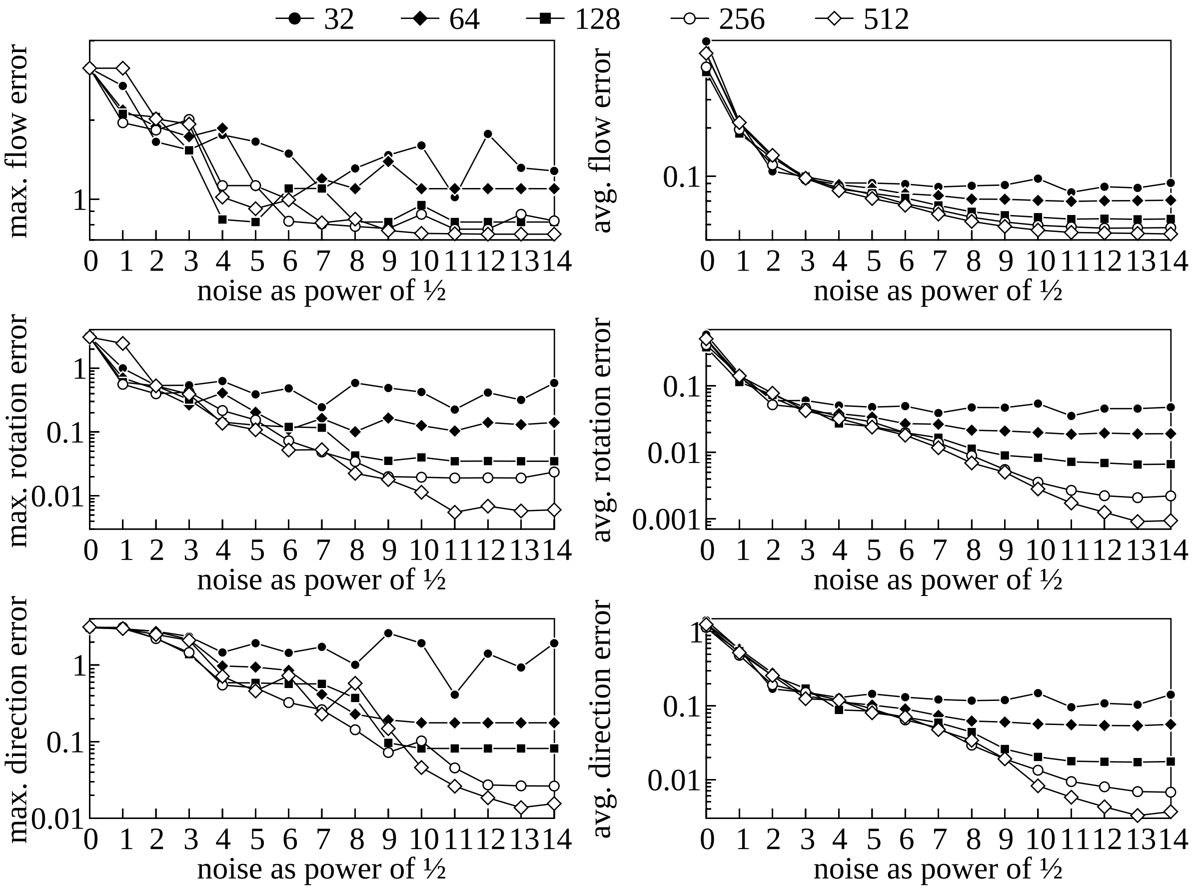

Figure 2.27: Results of the analysis of different image resolutions in room B.

2.5.5 Number of Adaptation Steps

The number of adaptation steps is the other runtime relevant parameter. It obviously has to be greater or equal than the image resolution, in order to adapt each pixel at least once. Therefore, it is even more important. 256, 512, 1024, 2048, 4096, and 8192 adaptation steps were tested. The charts in the figures 2.28 and 2.29 show that the number of adaptation steps should be greater than the image resolution but apart from this the number of steps was not very significant in this simulated scene. Higher numbers performed better but the differences were small. There was an outlier at 2048 steps in room B. Thus at least 4096 steps are recommended for an image resolution of 256 pixels.

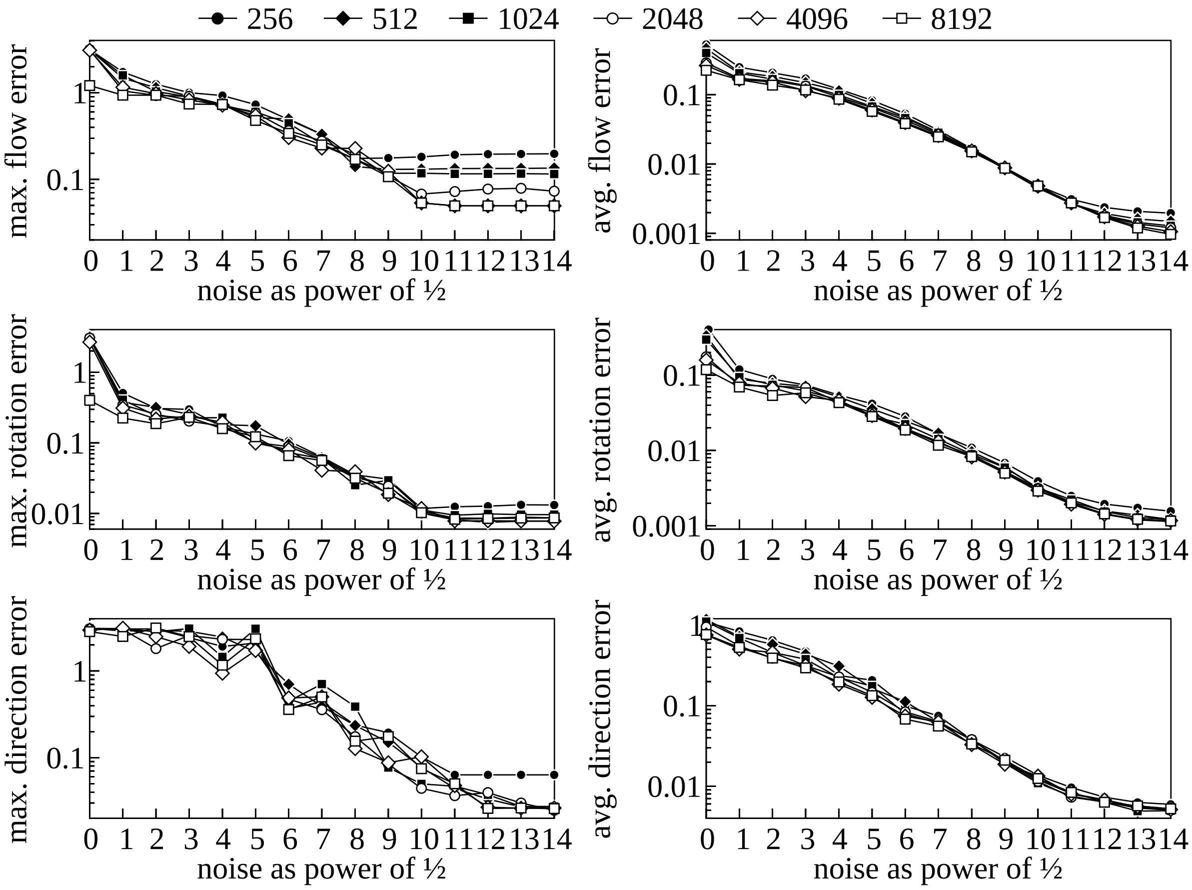

Figure 2.28: Results of the analysis of different numbers of adaptation steps in room A.

2.5.6 Rotation

Two approaches to calculate the rotation have been presented in section 2.3.1 beginning on page 32. The charts in the figures 2.30 and 2.31 indicate that the first solution was only slightly superior in experiments with very high noise. In contrast, with a noise level less than 2^{-5}, solution 2 performed much better than solution 1.

2.5.7 Bearing Direction

In section 2.3.3 beginning on page 35, four approaches to calculate the bearing direction have been presented. They cannot be seen independently of the method the rotation is calculated with, since according to equation (2.22) the determination of the bearing direction is based on the rotation. Therefore, all combinations of the calculation of the rotation and the direction must be examined. The results plotted in the figures 2.30 and

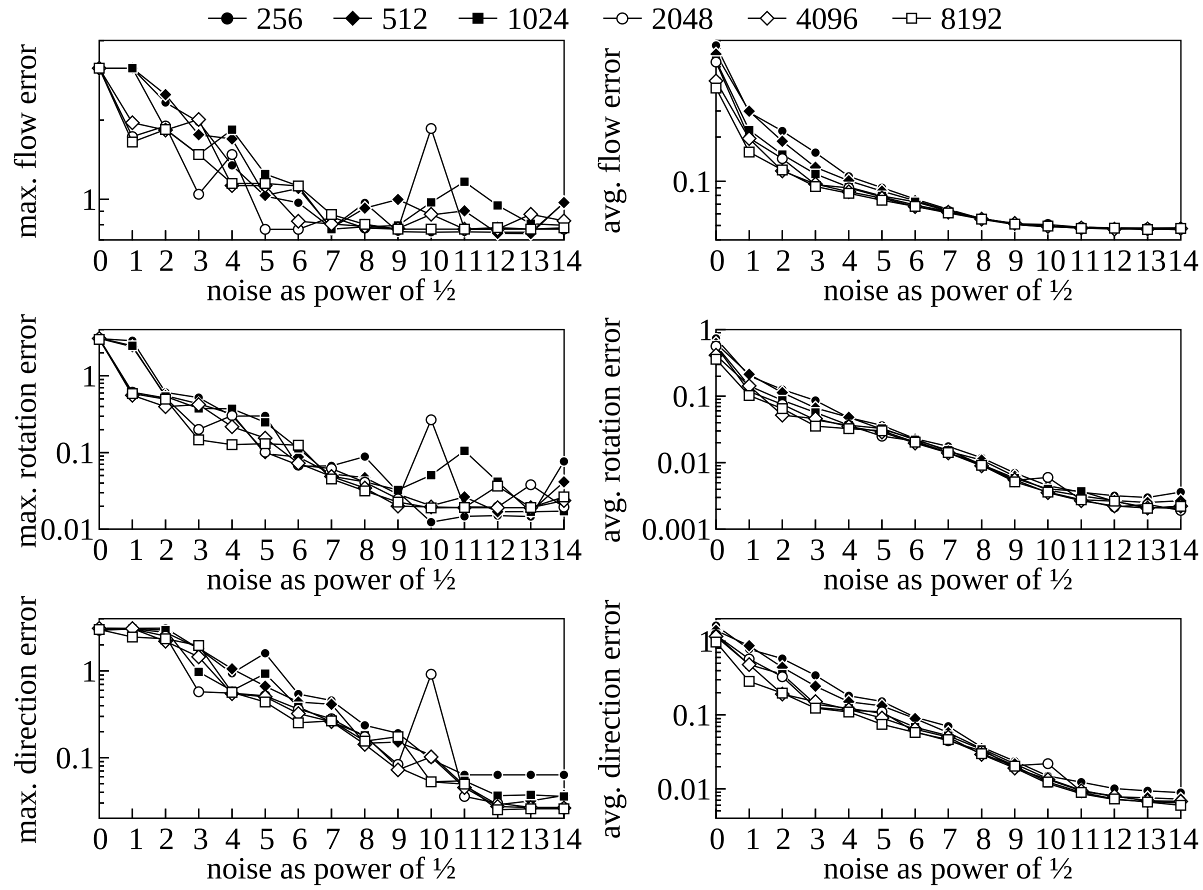

Figure 2.29: Results of the analysis of different numbers of adaptation steps in room B.

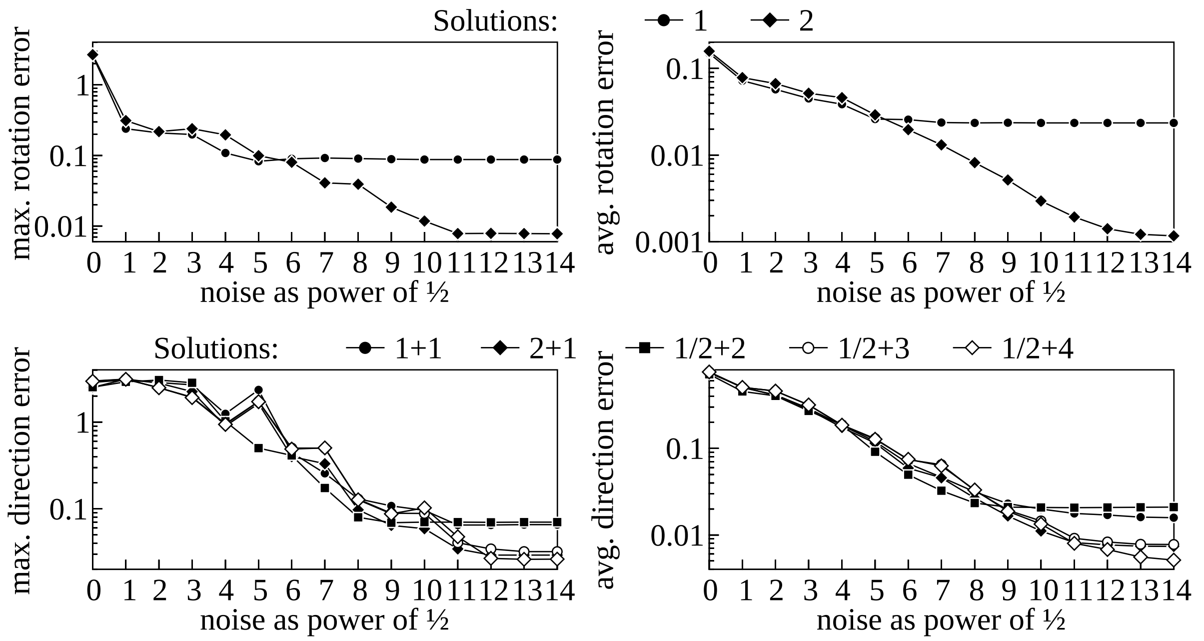

Figure 2.30: Results of the analysis of different solutions to determine the rotation and the direction in room A.

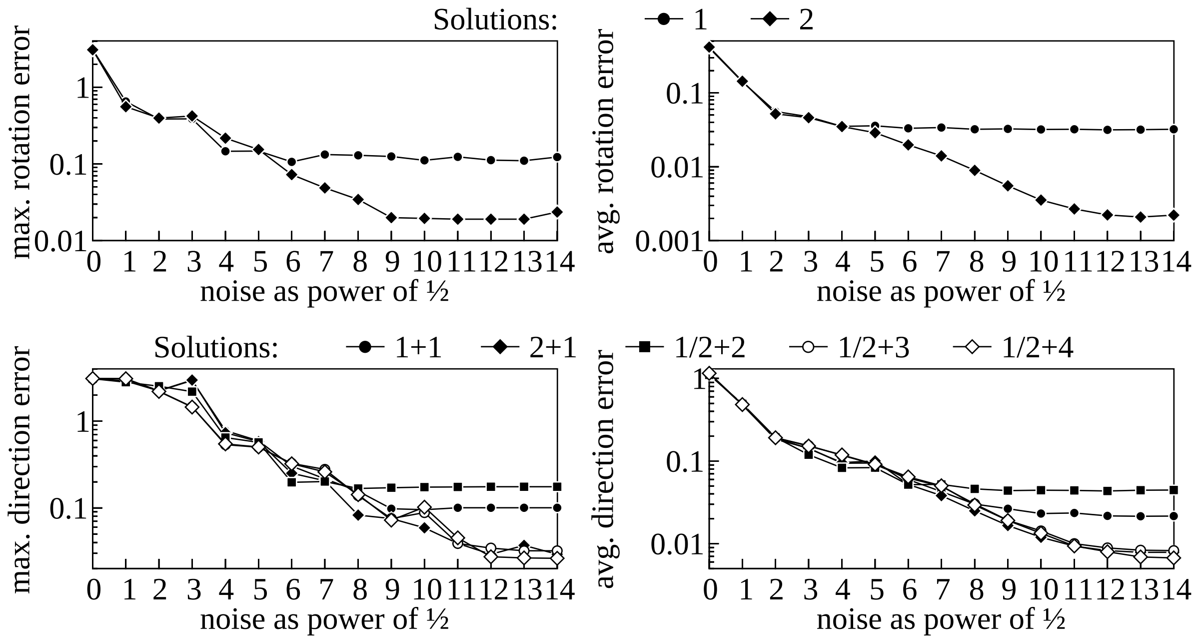

Figure 2.31: Results of the analysis of different solutions to determine the rotation and the direction in room B.

2.31 show that only solution 1 was dependent on the way the rotation was calculated. The other three approaches compensated the large errors that the first method determining the rotation produced.

The first solution to identify the bearing direction was very dependent on a correct calculation of the rotation. If the rotation was determined erroneously as the first approach of the calculation of the rotation did, the error continued to exist in the calculation of the bearing direction. If the first solution was used in combination with the second approach of the determination of the rotation, it performed very well over a wide range of noise levels.

Solution 2 was worst in the calculation of the bearing direction with low noise and was not better than the other alternatives under high noise conditions. Therefore, there is no reason to use this approach. Solution 3 was always inferior in relation to solution 1 in combination with the second method for the calculation of the rotation. Hence, there is no need to use this approach either.

Solution 4 was superior under low noise conditions but with a noise level higher than 2^{-11}, it was inferior in comparison to the first solution in combination with the second approach for the calculation of the rotation.

2.5.8 Results

The experiments show that the weight vectors used for the calculation of the correlation matrix should be simple. It is better to use no derivatives and no smoothing if the surroundings are unstructured or to only use one derivative if there are visible obstacles. The simlest filter should be used to calculate the derivative. Higher image resolutions improve the precision of the determined motion parameters. The number of adaptation steps is less important. A combination of solution 2 for the calculation of the rotation and solution 1 for estimation of the bearing direction performs best in most noise situations. Only if the noise is very low, solution 4 performs better in the calculation of the bearing direction.

For easier comparison, the data of four scenarios has been picked up from all the analyzed alternatives. The tables 2.2 and 2.3 show the errors either in radians or in

	room A		room B	
	low noise	high noise	low noise	high noise
maximum rotation error	0.008	0.08	0.024	0.07
average rotation error	0.001	0.02	0.002	0.02
maximum direction error	0.026	0.4	0.026	0.25
average direction error	0.005	0.06	0.007	0.05

Table 2.2: The selected results in radians.

multiples of a pixel's opening angle. They show the results for each room in a low noise case with a noise level of 2^{-14} and in a high noise case with a noise level of 2^{-6}. All these values were calculated using the standard parameter set that has been described on page 45, except that solution 1 is used to determine the bearing direction under high noise conditions.

With low noise, the rotation and the bearing direction are calculated more precisely in room A than in room B, although the maximum directional error is the same in both rooms. Under high noise conditions, the results are surprisingly inverted. *Panama* performs better in the structured room than in the unstructured one, only the average rotational error is uninfluenced by the different surroundings.

The determination of the rotation is always sufficient for navigation purposes because the maximum error is less than $\frac{\pi}{40}$. Under low noise conditions, the calculation of the bearing direction with a maximum error of $\frac{\pi}{120}$ is more than adequate. In contrast, with high noise, the maximum error of $\frac{\pi}{8}$ seems to be unacceptable for most navigation tasks. A closer analysis shows that the large errors only occur if the two images were taken at positions that are close together, i.e., the translational flow field is very small. Under these conditions it is inevitable that high noise disturbs the calculation of the direction.

2.6 Summary

In this chapter, an image processing algorithm called *Panama* has been presented. In the first section of this chapter, the concept of the panoramic images has been introduced. These pictures are colored, one-dimensional 360°-takings that depict a horizontal stripe of the surroundings. The fundamental advantage of these images is that they reflect—in a certain height—the whole surroundings. In contrast to planar pictures, the set of the depicted objects does not change if the camera is rotated around its vertical axis. A translational movement does only change the set of the portrayed objects in so far as some hidden items become visible and some visible things get concealed behind other objects. Thus, the set of depicted objects in the panoramic images as well as their order stays constant over large ranges of the environment. Therefore, the presented image processing

	room A		room B	
	low noise	high noise	low noise	high noise
maximum rotation error	0.33	3.26	0.98	2.85
average rotation error	0.04	0.81	0.08	0.81
maximum direction error	1.06	16.30	1.06	10.19
average direction error	0.20	2.44	0.29	2.04

Table 2.3: The selected results as multiples of a pixel's opening angle.

method takes advantage of this fact.

In the second section, the first part of the image processing algorithm *Panama* has been described. The method determines the optical flow field between two images in an iterative adaptation process. The most important characteristic of this approach is the strict neighborhood preservation during the matching process that is based on two assumptions: the depicted objects in both panoramic images are the same and their order does not change.

In the third section of this chapter, the second part of *Panama* has been presented. It uses the optical flow field to calculate the spatial relationship between the positions where the two processed pictures have been taken. It determines the rotation between these positions and the bearing direction from one of them to the other. In addition, the "distance" between the positions is calculated. This value does not describe the metrical detachment between the two positions, but instead, it represents the size of the translational flow field between the two images.

After the definition of the algorithm, its runtime complexity, i.e., the number of steps that have to be performed, has been analyzed. It has been shown that in a sequential implementation, the complexity is $\mathrm{O}(t_{max} n)$ if n is the number of pixels in a panoramic image and t_{max} is the number of adaptation steps that are performed during the matching process. In a parallel realization on conventional computer-hardware, the effort is reduced to $\mathrm{O}(t_{max} \log n)$. If the hardware is able to determine sums and minima of arbitrarily many values in one step, the complexity is further decreased to $\mathrm{O}(t_{max})$.

In a large number of simulated experiments, the dependence of algorithm's performance on the selection of its parameters has been analysed. Thus, the best combination of values has been determined. The results have been shown in figures 2.20 to 2.31 and in tables 2.2 and 2.3. To take some examples from the results, the average errors in a structured environment with two different noise levels are repeated here: with a noise ratio of approximately 2^{-14} in the processed images, the average rotational error was only $\frac{\pi}{1600}$ and the average directional error $\frac{\pi}{450}$, respectively. Under high noise conditions with a perpurtubation of 2^{-6}, the average rotational error was $\frac{\pi}{160}$ and the average mistake in the calculation of the bearing direction was $\frac{\pi}{60}$. Although the maximum errors are up to twelve times larger than these average values, the next chapter will show that the method's precision is sufficient to perform navigation tasks in real environments.

Chapter 3

Navigation with Panama

Navigation tasks in robotics are usually of the kind that a robot should move from point A to point B. As has been discussed in chapter 1, such a navigation task can be divided into two parts: the global and the local navigation. The first one leads the robot to its target, e.g., without taking care of local obstacles. Within the scope that has been set by the global navigation, the local navigation carries out the driving maneuvers and also avoids obstacles.

The previous chapter has presented an image processing method, which is the basic component for the navigation strategy that will be introduced in this chapter. In this strategy, all target positions of the navigation tasks will be represented by pictures of the surroundings, which have been taken at these positions by a panoramic camera on top of an autonomous mobile system. This mobile system navigates by comparing the image that it has taken at the target position with the pictures that it recognizes while it is driving. From this operation, it receives the course that it has to follow. The *Panama* algorithm that has been presented in the previous chapter is used for this "comparison".

The presented navigation method takes on the global as well as the local navigation. In contrast to global navigation, local navigation always depends on the kinematics of the robot that is controlled by the algorithm. Therefore, the definition of the navigation approach can be divided into a single section that describes the parts of the approach that are independent of the used platform and several other sections that delineate the adaptations to robots with specific kinematics. The hardware-independent pieces of the approach will be discussed in the section following this introduction.

During the development of the navigation method, it has been implemented on many different robots. These robots can be divided into two classes: kinematically restricted and kinematically unrestricted ones. Two robots of the kinematically unrestricted (holonomic) kind are shown in figure 3.1. As a representative for this class, an implementation for a simulated holonomic robot will be presented in section 3.2. In contrast, section 3.3 will describe the utilization of the navigation approach for a kinematically restricted wheelchair.

Both of these applications have pros and cons. Simulations only reflect a small part of the reality, and hence the results are only applicable to the real world in a limited way. Nevertheless, experiments can be performed with a robot that is not physically available at all. It is possible to record and analyze the results with little effort. The implementation on the wheelchair comes up with better results about the method's behavior under realistic conditions but carrying out experiments is more difficult.

3.1 Navigation Method

The method described in this chapter is a *topological navigation* technique based on a *guidance* method (cf. section 1.1.1). As most of these techniques, the navigation is based on sensory information. A camera that takes one-dimensional panoramic color pictures is used as essential sensor. Some ultrasonic sensors utilized for avoiding obstacles support the camera.

The navigation algorithm is based on the image processing method *Panama* that has been described in the previous chapter. *Panama* generates information from two images that describes the spatial relationship between the positions where these images have been taken. To take advantage of this information, it would be a straightforward approach to represent places as images of their surroundings. Then, *Panama* can be used to return to these places by applying it to the image of robot's current surroundings and the picture

a) b)

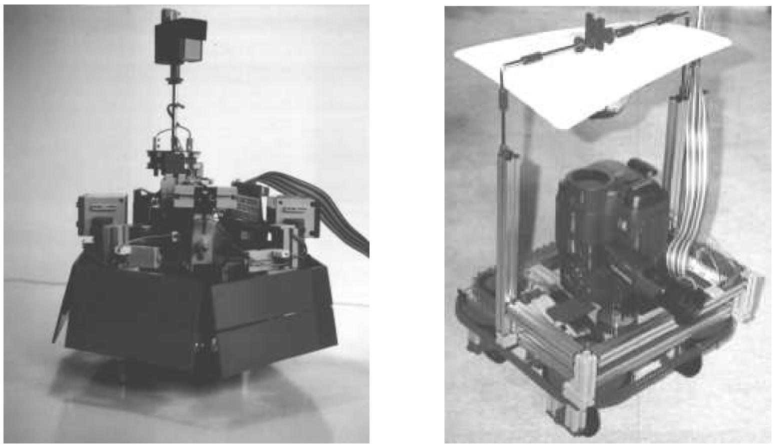

Figure 3.1: Two robots that have been used for the experiments. a) A robot with a rotating photo-transistor for taking monochrome panoramic images. b) A robot with the sensing apparatus that has been used on the wheelchair in section 3.3.

that has been taken at a certain target place. Such a simple approach would only work as long as every place is located in the same room. In addition, this room must not contain obstacles that would disturb the matching process between the images. Otherwise, it would be impossible to determine a correct optical flow field. In this case, the values for *rotation* and *bearing direction* calculated by *Panama* would not reflect the real relations between these positions.

To avoid these limitations, not only the relevant goal positions are represented by images, but also the paths between them. When driving along a trajectory, the positions on it are reached in a certain sequence. Therefore, it is possible to describe a route as a succession of positions. If these positions are represented by pictures of their surroundings, a trajectory can be described by a sequence of images. This allows a robot to model its area of operations by storing a series of pictures for each route that it has to drive along. For each of these routes it has to learn the necessary images during a single training drive. During such a drive, a teacher has to show the robot how to follow the trajectory, i.e., the teacher controls the robot along the path. In contrast to other approaches, the robot neither stores motor commands nor abstract driving instructions. During the training drive, it only records the panoramic images of the surroundings.

The panoramic pictures are recorded in regular distances during the training drive. In both implementations that will be described in the sections 3.2 and 3.3, an image is taken every 10 to 15 cm of driving distance. Even on short trajectories, a large number of images is recorded. Therefore, it is necessary to compress the images sequences. This happens by removing pictures from the series that are not required for the description of the trained route. There are two possibilities when this filtering process can take place:

1. The robot may process the recorded images immediately. As a result, the calculation effort would be uniformly distributed over the whole training path. The coincidence between the taking and the processing of the images would enable the teacher to recognize *where* images have been recorded that the robot really needs to represent the trained trajectory. If the pictures would not be evaluated immediately, this important relationship —at least during the test phase— would be lost.

2. As an alternative, the robot may record all pictures during the training drive and may not process them until it would reach the trajectory's end. By this means, the robot would be completely free in the processing succession of the recorded images because it would have all images at its disposal simultaneously. Nevertheless, a large number of recorded images would result in a comparatively high memory consumption. Besides, the whole processing effort would arise at the end of the trained trajectory instead of being uniformly distributed over the entire training as is the case in the first variant.

Because of the higher flexibility in the processing order, the second approach has been chosen in this work. This flexibility is a prerequisite for the navigation method's use in the controlling of a wheelchair that will be presented in section 3.3. In contrast, the holonomic robot that will be introduced in section 3.2 may also utilize the first approach as presented by Röfer (1995c). At least in a simulation, it is possible to restore the relationship between the positions where the panoramic pictures have been taken and the results of the filtering process for the second approach.

In the selected approach, the robot takes all images during the training drive. Therefore, it has recorded a sequence S of m panoramic pictures I_i when it reaches the end of the path:

$$S = [I_0, I_1, \ldots, I_{m-1}] \tag{3.1}$$

3.1.1 Selection Criteria

The robot removes all images from the recorded sequence that it does not need to cover the trajectory. Which images are required depends on the way the robot uses the images when driving autonomously along the trained trajectory. As the pictures represent places on the trained path, it reproduces the learned trajectory by making its way from one image to the next. More precisely, the robot is driving in the direction of the position where the next image in the sequence has been taken until it has reached it. Then, it continues with the next picture, until it has arrived at the position of the last image in the sequence, i.e. the end of the trajectory. To be able to determine the driving commands necessary to head for the positions on the path, the robot also continuously takes images during its autonomous driving. It relates each of them to the next target picture by the image processing method *Panama*. *Panama* calculates the *bearing direction* of the next goal position as well as the *rotation* relative to it. Based on these values, the driving commands are determined, considering the kinematic restrictions of the robot.

This strategy for the autonomous drive of the robot predetermines the criteria for filtering the recorded images after the training drive: it is not necessary to preserve a picture if the position that it represents is also reached if the robot is directly heading for the subsequent image. This rule can also be applied repeatedly: a whole sequence of images can be removed if the robot still arrives at the positions where these images have been taken if it is directly heading for the picture following this sequence.

The teacher can use the same set of drive commands during the training phase as the robot can utilize while it is driving autonomously. Therefore, the robot can reproduce every learned trajectory if it is able to determine the drive commands correctly. It generates these drive commands by a predefined model from the values *rotation* and *bearing direction* that have been calculated by *Panama*. Therefore, the determination of the correct drive commands depends on *Panama*'s accuracy in the calculation of these two values. As the robot follows the same trajectory during the learning phase as during the independent execution, it is already able to check whether *Panama* extracts the correct parameters from the images taken on this path while it is trained. To establish the correctness of the determined values, the robot needs a reference. The only information available—in addition to the recorded pictures—is the drive commands given by the teacher. Therefore, the robot remembers these commands in addition to the recorded images until the filtering process at the end of the trained path. After the filtering process, these drive commands are no longer necessary. Hence, they can be removed.

The robot determines the relationship of two images with *Panama*, and then it compares the calculated rotation and bearing direction with the movements executed. If the rotation and the bearing direction are contradictory to the performed drive commands, the recorded image sequence has to be divided into partial sequences. This will be described in section 3.1.2. As the robot should not only reach a target position but also pass all positions on the trajectory in front of it, a second criterion for the image selection has to be introduced: if the robot changes its driving direction during the training, the image taken at this turning-point also has to be preserved. Otherwise, the robot will not arrive at this position during its autonomous drive because if the mobile system is heading for a position behind the turning-point from a place in front of it, it will not reach the turning-point (cf. figure 3.2). Therefore, all images taken at turning-points have to be preserved. This includes the last picture of the sequence because it signals the robot that it has reached its goal.

More concrete criteria for the filtering of the image sequences that have been recorded by robots with different kinematics will be presented in sections 3.2.3 und 3.3.3.

3.1.2 Dividing the Image Sequence

The robot tries to represent the trained trajectory by as few images as possible. It divides the recorded picture sequence into partial sequences in a way that only one image of each

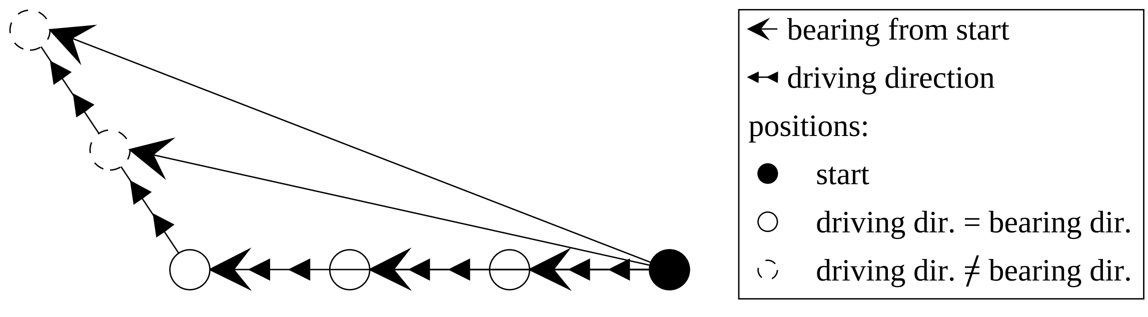

Figure 3.2: Illustration of the influence of a turning-point on the relationship between the driving direction and the bearing direction.

partial sequence is required to describe the trained trajectory sufficiently. As the robot runs through the pictures from the beginning to the end during its autonomous drive, the last image of each partial sequence has to be preserved. The other pictures are unnecessary because the robot is able to reach the positions where they have been taken if it is only heading for the last image of each partial sequence. Therefore, they can be deleted.

First, the image sequence is divided into partial sequences so that each partial sequence contains only images that have been taken while the robot has not changed its driving direction. Each of these sequences is expanded by the last image of the previous sequence. This picture is placed first in the partial sequence because in further processing, it has to be tested whether the robot can reach the last position of one partial sequence from the last position of the previous one. Therefore, the last image of each partial sequence occurs twice in the list of the partial sequences S':

$$\begin{aligned} S' &= split_S \\ &= [S'_0, S'_1, \ldots] \\ &= [[I_0, I_1, \ldots, I_k], [I_k, I_{k+1}, \ldots], \ldots, [\ldots, I_{m-1}]] \end{aligned} \tag{3.2}$$

$$\begin{aligned} split_S &= \begin{cases} [] & \text{if } R = [] \\ split_R + [\text{last}_R + T] & \text{otherwise} \end{cases} \\ &\text{where} \quad n = \text{countlast}^S_{equalDriveCommands} \\ &\qquad\quad T = \text{drop}^S_{\#S-n} \\ &\qquad\quad R = \text{take}^S_{\#S-n} \end{aligned} \tag{3.3}$$

For each of the partial sequences in S', it has to be checked whether the position of the last image can be reached from the positions where the other images in this sequence have been taken. If there is an image that does not satisfy this criterion, the partial sequence has to be divided again. The sequences are partitioned until the position that is represented by the last picture in each sequence can be reached from all the other positions in the same sequence. As the reachability of the last position in each partial sequence is checked, the images are processed from the end to the beginning. The function *countlast* counts from the end of the sequence how many successive pictures satisfy the selection criterion c. The function f determines how many of these images can be assigned to a partial sequence that does not require further inspection because all of its pictures satisfy the selection criterion c. Therefore, only the last image of this sequence is required for a sufficient representation of all positions in this series. The other pictures are not stored. The remaining partial sequences are processed in the same way. The last images of all partial sequences are concatenated to the sequence S'' that is permanently stored by the robot. The calculation of S'' ensures that the pictures that have been inserted twice into S' in equation (3.2) occur only once in S'':

$$\begin{aligned} S'' &= \left[split'_{S'_0}, split'_{S'_1}, \ldots\right] \\ &= [\ldots, I_{m-1}] \end{aligned} \tag{3.4}$$

$$split'_S = \begin{cases} [\text{last}_S] & \text{if } \#R \le 1 \\ split'_R + [\text{last}_S] & \text{otherwise} \end{cases}$$
$$\begin{aligned} \text{where} \quad n &= \text{countlast}_c^S \\ d &= f_n \\ R &= \text{take}_{\#S-d}^S \end{aligned} \tag{3.5}$$

S'' is exactly the sequence of pictures that the robot needs to follow the trained trajectory with respect to its kinematic restrictions. The images can be interpreted as landmarks that the robot uses for its orientation in the environment. It has individually chosen them among a larger number of possibilities, without any help of the teacher.

During the selection process, the images are filtered from the end to the beginning. This is the reason why they are recorded first and then processed after the training drive. If they would be analyzed immediately after taking, the sequence to be processed in forward direction. This would only allow the robot to check whether it is able to *return to the beginning* of the trained path, instead of *driving to its end*. This approach would more likely correspond to the "homing"-behavior of honeybees that has been described in section 1.3.2 on page 15. As it is possible that the robot has to drive along a path in one direction in a different way than in the other, e.g., it should pass an obstacle on the right side in each direction, it would be quite confusing for a teacher to show the robot on the way there how to drive back. Hence, the more costly variant has been chosen that first records all images of a trained trajectory and processes them afterwards.

3.1.3 Division Criterion

The function f that has been presented in the previous section determines how an image sequence is divided into halves after it has been established that only the last n images satisfy the selection criterion c. The image sequence must be divided in a way that all pictures in the first half satisfy this selection criterion. In addition, at least one image at the end of the second half must fulfill this criterion as well because the last position of the first partial sequence must be reachable from the last position of the second partial sequence to preserve the continuity of the trajectory between the two halves. If the last n pictures of the sequence satisfy the criterion, a maximum of $n-1$ images can be assigned to the first partial sequence.

The function f decides how many of the $n-1$ possible pictures are assigned to the first sequence and how many have to remain in the second one. The most compact description of a trajectory is achieved if the maximum permitted number of images is assigned to the first partial sequence:

$$f_n = \max(n-1, 1) \tag{3.6}$$

This approach assumes that the robot will exactly reach the positions represented by the images while driving autonomously along the trajectory. If the robot classifies a goal position as reached too early, it is possible that the robot is out of the region where the direction to the next target can be calculated correctly, and hence it may lose its way.

The safest description of a trajectory is achieved if all recorded pictures are preserved, i.e., none is filtered out. In this case, only one image is assigned to the first partial sequence. Therefore, the process would generate as many partial sequences as pictures:

$$f_n = 1 \tag{3.7}$$

A definition of the function f in a real implementation will be between the two extremes in the equations (3.6) and (3.7). It depends on the robot's ability to reach intermediate goals precisely, e.g., an omni-directional robot will be more accurate than a kinematically restricted one. Therefore, the function f has to be selected depending on the kinematics of the used robot. Two different definitions of f for robots with dissimilar kinematics are given in sections 3.2.3 and 3.3.3.

3.1.4 Topological Map

The algorithm presented previously only enables the robot to learn single trajectories. To permit the robot to move between a larger number of positions, a route has to be taught for each combination of starting position and end position. In the case of n positions, $n(n-1)$ trajectories would have to be learned. As all routes have to be trained manually, it would be a better solution to describe the connections between the positions by a smaller number of trajectories. In addition, it would be favourable to have to teach as short routes as possible.

The connecting routes between the positions must comply with the spatial conditions of the surroundings, i.e., they must follow given structures, e.g. corridors or streets. Therefore, many routes share partial routes with other routes. To reduce the number and the length of the trajectories that have to be learned by the robot, they are divided in a route that each partial route has to be taught only once. To drive from one position to another, the robot has to ascertain the necessary partial routes and it has to combine them to an image sequence that leads from its current position to the desired target.

As the robot has no knowledge about the relations in its environment at all, the dissection of the connecting routes between all goal positions has to be performed by the teacher. This partitioning is achieved by defining intermediate target positions that are placed at the beginning and the end of shared routes. Then, the teacher only trains routes between immediately neighboring positions. This remarkably reduces the number and the length of the trajectories that have to be learned by the robot. To enable the robot to combine the trained routes to longer routes, the starting and the end positions of each trajectory are named. Besides, these markings are important for the user of the robot because they empower him to specify the goal positions for navigation tasks. The names of the positions names are the people's only ability to relate to the internal model the robot has of the environment, i.e. the only language they can use to communicate with each other.

The naming of the positions converts a collection of unrelated trajectories into a topological map. The markings represent the places in the map and the image sequences describe the directed edges between them. This map M is represented as a set of trajectories m_i. Each trajectory contains the name of its starting position f_i, the picture sequence S''_i that describes the route to the target position and the end position t_i itself:

$$\begin{aligned} M &= \{m_0, m_1, \ldots, m_{z-1}\} \\ \bigwedge_{i=0}^{z-1} m_i &= \begin{pmatrix} f_i \\ S''_i \\ t_i \end{pmatrix} \end{aligned} \tag{3.8}$$

It is assumed that there is at most one trajectory between each starting position f and each goal position t in the map M, i.e., each pair of f_i and t_i is unique:

$$\bigwedge_{i=0}^{z-1} \bigwedge_{j=0}^{z-1} i = j \vee f_i \neq f_j \vee t_i \neq t_j \tag{3.9}$$

If the robot is instructed to drive to a certain target, it searches the map for a route from its current position named f to the desired goal position t. In general, there are several possibilities to get from f to t. The robot has to select the route among these alternatives that is best according to a predefined assessment scheme. Many assessments are possible: The best route may be, e.g., the one that is shortest, fastest, most secure, or most economical. It is assumed that the robot learns only during the training drives. Therefore, it is impossible to assess the routes by their security because the teacher controls the robot during the training phase and he will train each trajectory only once.

So far, the only prerequisites the robot has to fulfill have been the abilities to take panoramic images and to execute drive commands. For most of the possible assessments, additional sensors are necessary. If the robot is assumed to be able to measure the journey time or the covered distance, the following assessments are conceivable:

- The robot could measure the duration of each training drive and use it to estimate the journey time of its autonomous moves. This approach assumes that the robot will autonomously cover a route as fast as during training. This will not always be the case, e.g. if a robot slows down when it approaches a partial target represented by a certain image, the duration of following the trajectory depends on the number of images preserved by the filtering process. As the teacher cannot know which images the algorithm will choose, he would control the robot with other speed dynamics. Therefore, the journey time would be different during the training and during the autonomous drive. A robot that slows down in front of intermediate goals will be presented in section 3.2.

- If the journey time depends on the number of images describing the trajectory, this number could be used to assess the route's length. But as the number of images does not state anything about the distances between their positions, this criterion is not able to estimate the duration of the autonomous drives by itself.

- Another alternative is to use the metrical length of a trajectory for its assessment. This requires the robot to be equipped with an odometer. As the distance does not always correlate with the journey time, e.g. if there are many curves in the trajectory, it may be combined with the previous assessment: The number of learned images describes the route's complexity while the metrical length rates the route's straight parts. The two robotics platforms that will be presented in the sections 3.2 and 3.3 both support dead reckoning. Therefore, it would be possible to implement such an advanced assessment on these robots.

- The simplest way to rate routes is to count the number of sub-routes that make up the combined trajectory. As it has not been the aim of this work to develop an advanced route-finding algorithm, this approach has been chosen. In most environments, this simple solution leads to the same results as the more complicate ones. It has been implemented for the simulated robot that will be presented in section 3.2 and has been sufficient for the experiments that have been performed.

The routes that will be found by the search algorithm are represented by a list of names. If the robot searches for a route from f to t, a correct candidate must begin with f and has to end with t. The entries between these two names result from the trajectories defined in the map M:

$$W_M^{f,t} = [f, \ldots, t] \tag{3.10}$$

To determine all possible routes, a function N is defined that generates a set of neighbors of a particular location f in the map M:

$$N_M^f = [t_i \mid m_i \in M, f_i = f] \tag{3.11}$$

From a list of routes L that all start at the same location, the function W' generates another list of routes that contains the shortest routes to all locations in a map M. Each of these generated routes begins with one of the routes in L. It is assumed that all routes in L are sorted by their length, i.e. the first route in the list is always the shortest one. The resulting list is sorted in the same way.

W' extracts the first route P from the list L. As P is the shortest route from its starting location to its end, all further routes targeting at the same location can be removed from the rest of the list, resulting in a list L'. As the shortest route to P's goal has already been found, all trajectories to this location can be removed from M, resulting in a map M'. In this reduced map, all neighbors of P's goal are determined and a list L'' is calculated that contains all the routes in L' and all possible immediate extensions of P. Then, all further extensions of all the routes in L'' are determined by recursively calling W'. As the map M is reduced by one goal location in every step, W' will terminate if either no locations remain in the map or L is empty. The latter will be the case if there are unreachable locations in the map. In addition, all generated routes are free of circles because the map M' used to determine all immediate extensions of P does not contain any trajectories to locations in P.

If the number of places in a route is used as the route's assessment, L can only contain routes with the same length as P and routes that are one trajectory longer than P. If L is already sorted, the concatenation of L' and all immediate extensions of P is also sorted because these extensions are longer than the last route in L' or have the same length. If another method would be used for routes' assessment, the extensions of P have to be sorted by their assessments and then have to be merged with L'. This would generate additional computational costs and would lead to an algorithm that is similar to the A* algoritm developed by Hart *et al.* (1968).

$$\begin{aligned} W'^{L}_{M} &= \begin{cases} L & \text{if } L = [] \vee M = [] \\ P + W'^{L''}_{M'} & \text{otherwise} \end{cases} \\ \text{where} \quad & P = \mathrm{hd}_L \\ & L' = [l \mid l \in \mathrm{tl}_L, \mathrm{last}_l \neq \mathrm{last}_P] \\ & M' = [m_i \mid m_i \in M, t_i \neq \mathrm{last}_P] \\ & L'' = L' + \left[P + [t] \mid t \in N^{f}_{M'}\right] \end{aligned} \tag{3.12}$$

The function W uses W' to produce all routes that begin with position f. Then, it searches the resulting list for all routes that end at t. As the routes in the list are sorted by their assessments, the first route found is the one with the lowest rating. Therefore, it is the route the robot should use to drive from f to t:

$$W^{f,t}_{M} = \mathrm{hd}\left[w \mid w \in W'^{[[f]]}_{M}, \mathrm{last}_w = t\right] \tag{3.13}$$

However, the robot does not need a list of names to cover the route from f to t. Instead, a list of images is required that describes this route. Therefore, the corresponding picture sequences have to be picked out of the map M and have to be combined to an image sequence T that represents the complete route from f to t. As it has been defined in 3.9 on page 69 that there is at most one trajectory from one position in M to a neighboring one, it is possible to determine a corresponding trajectory from M for each adjacent pair of places in W. The concatenation of the image sequences that describe the found trajectories results in the searched picture sequence T:

$$\begin{aligned} T_M^{f,t} &= \text{concat } T'^{\text{hd}_P}_{M,\text{tl}_P} \\ &\quad \text{where} \quad P = W_M^{f,t} \\ &\qquad\qquad T'^{f}_{M,P} = [S''_i \mid m_i \in M, f_i = f, t_i = \text{hd}_P] + T'^{\text{hd}_P}_{M,\text{tl}_P} \end{aligned} \tag{3.14}$$

3.1.5 Hardware-Specific Arrangements

At this place, the hardware-independent description of the navigation method is finished. In the next two sections, the algorithm's hardware-dependent details for a holonomic robot as well as for a wheelchair will be specified:

1. The selection criterion c that decides whether *Panama* has calculated the relation between two images correctly.
2. The driving behavior that derives the drive commands from the information generated by the image processing when the robot repeats a path autonomously.
3. The "target-reached" criterion that recognizes that the robot has arrived at an intermediate goal.
4. The definition of the separation criterion f that determines where to split an image sequence into halves.

3.2 Controlling a Holonomic Robot

In this section, the implementation of the navigation method that has been described in the previous section is presented for a simulated, holonomic robot. The simulation has been performed using the robotics simulator tool SimRobot (Siems *et al.* 1994).

3.2.1 The Robot

The *B21* manufactured by *Real World Interfaces* has been used as a model for the simulated robot. The cylindrical robot has a diameter of 50 cm and a height of approximately 110 cm. In the simulation, the robot is able to drive straight ahead in all directions without turning its body. The robot's drive commands consist of two components: the speed s and the driving direction γ. The speed s can be adjusted between zero and a maximum speed of $s_{max} = 100$ cm/s. The driving direction γ is unlimited. Therefore, any angle can be used.

The simulated robot is equipped with the following sensors:

- A camera that takes panoramic images. It is fitted in the center on the robot's top and perceives a thin stripe of the surroundings in a height of 120 cm. The pictures are taken with a resolution of 256 pixels. As SimRobot models the pixels as points and not as areas, some additional computational effort is required to reduce aliasing effects in the panoramic images. SimRobot determines the color of each pixel from the color of the surface that is visible in the direction of the pixel's center. In reality, the color of a pixel is a mixture of the coloring of all surfaces included in the pixel's *viewing pyramid* (cf. figure 2.2 on page 21). Therefore, SimRobot may calculate the wrong colors and—more importantly—may show a small detail in one image and ignore it in another that has been taken at a different position. To reduce this effect, the panoramic images are calculated with a higher resolution and then scaled down by the factor of four. This reduces the aliasing effects as well by a factor of four but does not eliminate them completely. As a compensation, no noise is modulated on the simulated camera images.

- 32 ultrasonic sensors. These are grouped in two horizontal rings at 16 sensors each. In each ring, the sensors are uniformly distributed around the robot. One ring is fitted at a height of 70 cm. This height has been chosen because the table-tops are

a) b)

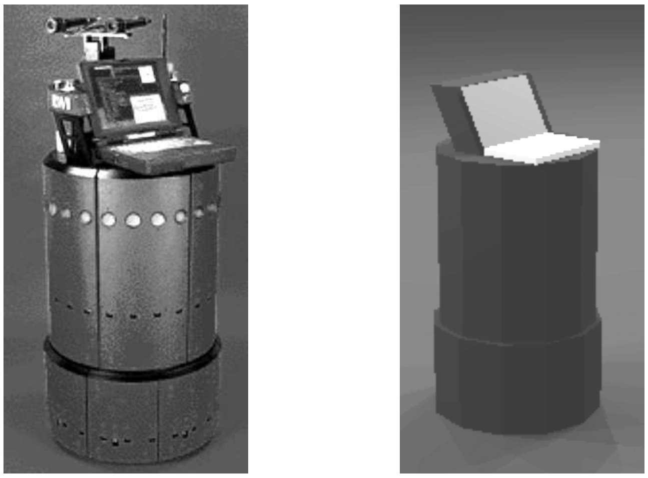

Figure 3.3: The *B21* manufactured by *Real World Interfaces*. a) The original robot. b) The simulated imitation.

at this elevation in the simulated scene (as well as in reality). Therefore, the robot is able to detect them with the sensors included in this ring. The second ring is fixed to the robot at a height of 30 cm. This enables the robot to detect smaller obstacles, e.g. low furniture. The ultrasonic sensors are also simulated by a discrete scanning of the surroundings. Therefore, the robot may overlook thin objects, e.g. table-legs. Typical effects of ultrasonic sensors, as specularity or cross-talks (Lee 1996, Jörg *et al.* 1993), are currently not simulated by SimRobot.

Figure 3.3 shows the model together with its simulated imitation.

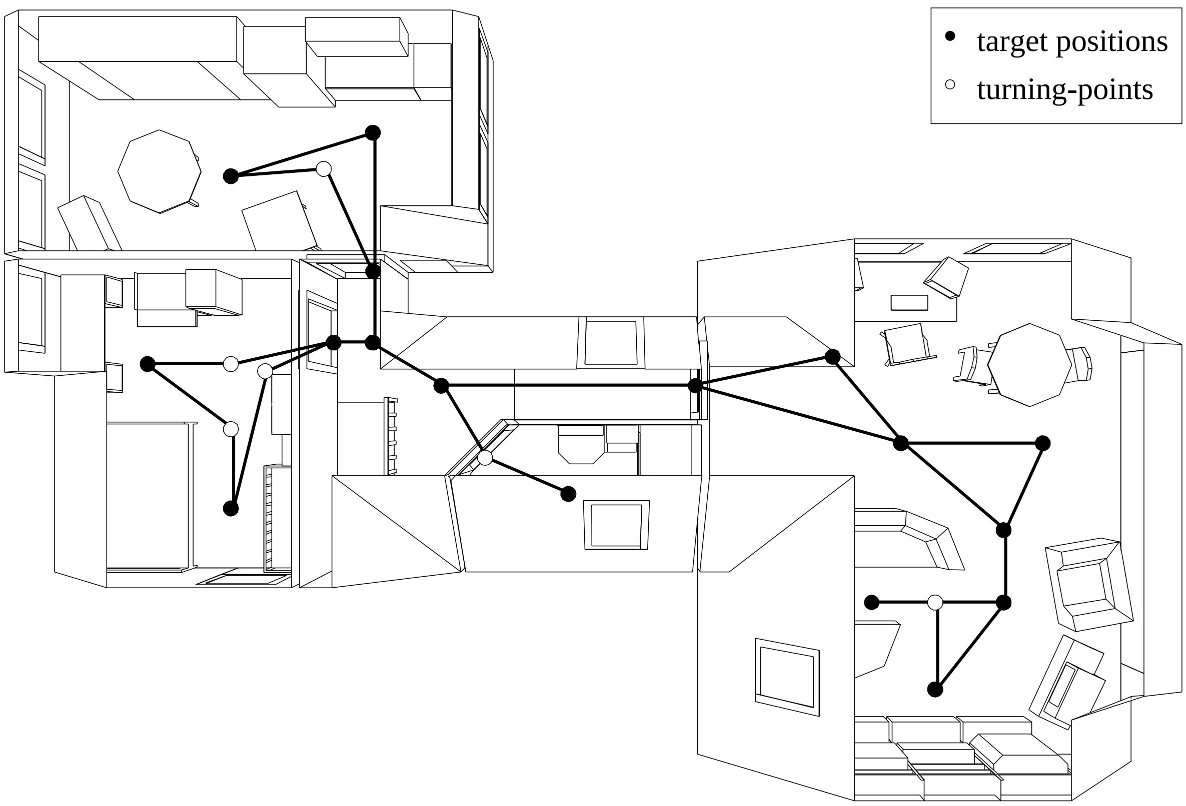

Figure 3.4: The simulated flat. The solid dots mark the 17 target positions. The hollow circles represent turning-points in the connecting trajectories.

3.2.2 The Surroundings

The simulated robot moves inside an attic flat that is shown in figure 3.4 from the bird's-eye view[1]. It consists of four rooms and one corridor. As an attic flat, it differs from the surroundings used in most other robotics applications in that most walls are not perpendicular and no rooms is similar to another. In the flat, 17 positions have been defined that can be used as targets for the navigation of the robot. 21 trajectories connect these goal positions. The robot has to learn each of them in forward and in backward direction to be able to drive from every possible starting position to any other position. Considering the robot's kinematics, the trajectories are composed of straight lines. Again, figure 3.5 shows the original and the imitation. The simulation is not as detailed as reality. As this is the weakness of all simulations, many algorithms have been developed in simulations that do not work in reality. For the navigation method presented in this chapter, the lack of details is more likely detrimental because a more detailed environment supports the matching process of *Panama*.

3.2.3 Hardware-Specific Arrangements

Below, the hardware-dependent parts of the navigation algorithm that has been introduced in section 3.1 will be presented. First, the criterion will be discussed that decides which images from the recorded sequence are required to represent a trained path sufficiently. Then, the method will be described that the robot uses to follow a learned sequence on its own. Part of this process is the "target-reached" criterion that tells the

[1] An MPEG-video that shows the simulated flat is available for downloading from the internet under http://www.informatik.uni-bremen.de/˜simrobot.

a) b)

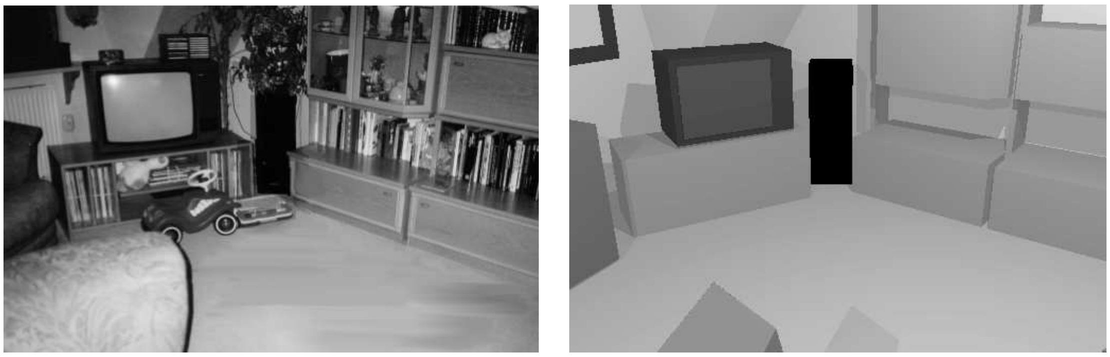

Figure 3.5: The environment of the simulated experiments. a) An original view. b) The simulated view.

robot when it has arrived at an intermediate goal position. Finally, the approach for dividing image sequences into halves during learning will be presented. This function will be discussed at the end because its definition depends on the way the "target-reached" criterion is implemented.

Selection Criterion

The selection criterion c that has been presented in section 3.1.1 determines whether *Panama* has calculated the correct values for the *rotation* and the *bearing direction* between the last image and another image of the same partial sequence. For the rotation as well as for the bearing direction, some criteria can be defined that check the correctness of the calculated values. The kinematics of the robot are distinguished by the fact that it moves in straight lines and does not turn. The latter means that there never is a rotation between the images recorded during a training drive. In this respect, *Panama*'s calculation of the rotation should be nearly zero, tolerating small errors that cannot be ruled out. The advantage of this criterion is that it does not need any information about the driven path. The drawback is that a correct rotation does not guarantee for an accurate bearing direction. Therefore, this assessment alone is uncertain:

$$c_1 = |\omega| < \omega_{min} \tag{3.15}$$

As has been described in section 3.1.1 on page 64, the whole image sequence is divided into parts that contain only images that have been taken while the robot has moved in a constant direction. Therefore, it can be assumed that each partial sequence contains only images that have been recorded while the robot has moved in an unchanging, straight direction. As a result, the bearing direction of the last image in a partial sequence is always equal to the driving direction. Therefore, the bearing direction δ that has been determined by *Panama* is correct, if it corresponds to the robot's driving direction γ_{stored} that has been stored during the teaching process:

$$c_2 = \left|\delta \hat{-} \gamma_{stored}\right| < \delta_{min} \tag{3.16}$$

The operators $\hat{+}$ and $\hat{-}$ limit the result of the operation to the range $[-\pi \ldots \pi[$—analogously to the operators $\ddot{+}$ and $\ddot{-}$ that have been defined in chapter 2 on page 24:

$$a \hat{+} b \stackrel{\text{def}}{=} \begin{cases} (a+b) \,\dot{\text{mod}}\, 2\pi & \text{if } (a+b) \,\dot{\text{mod}}\, 2\pi < \pi \\ (a+b) \,\dot{\text{mod}}\, 2\pi - 2\pi & \text{if } (a+b) \,\dot{\text{mod}}\, 2\pi \geq \pi \end{cases}$$
$$a \hat{-} b \stackrel{\text{def}}{=} a \hat{+} (-b) \tag{3.17}$$

The Rotation and the bearing direction have only been determined correctly if both c_1 and c_2 are satisfied:

$$c = c_1 \wedge c_2 \tag{3.18}$$

Driving Behavior

If the robot drives along the learned trajectory autonomously, it is moving from intermediate goal to intermediate goal. It is continuously taking images of the surroundings while it is driving. It applies *Panama* to the currently taken image and the picture that represents the next intermediate goal position. The algorithm determines the direction to the next intermediate target in the robot-based system of coordinates. The calculated direction can directly be used as driving direction:

$$\gamma = \delta \tag{3.19}$$

During the drive, a certain time passes between the taking of two panoramic images. If the robot should follow the learned trajectory precisely, it must be ensured that it does not shoot past a goal position. This is achieved by reducing the robots driving speed when it is approaching an intermediate target. As a result, the distance that is covered by the robot between the taking of two successive pictures is reduced. The only information the robot can utilize for estimating the distance to the next goal position is the sum of the flow field φ that is determined by the image processing method *Panama*. This value does not describe a metrical range. Instead, it is a relative measure that depends on the current surroundings. Therefore, the robot's speed s is calculated by multiplying φ with a predefined factor τ that must be selected in a way that ensures that the robot will slow down sufficiently early in front of each intermediate target position in its whole area of operations. As the distance φ corresponds—according to equation (2.23) on page 35—to the summed translational flow field, φ is big where the flow is big and thus the distance to the surrounding objects is small. Therefore, τ only has to be chosen in a way that stops the robot sufficiently early at the most narrow place in its area of operations. As the robot is not able to drive with an arbitrary velocity, its speed s has also to be limited by the top speed s_{max}:

$$s = \max\left(\tau\varphi, s_{max}\right) \tag{3.20}$$

"Target-reached" Criterion

The "target-reached" criterion θ determines when the robot has arrived at the next intermediate goal on its trajectory. If this event has been detected, the robot will switch to the next picture in the image sequence that describes the path, and thus it will head for the following intermediate target position. There are two possibilities for the definition of a criterion that utilizes the information generated by *Panama*:

1. The robot can determine whether it has passed a goal position. In this context, “passing” means that the target is behind the robot according to its driving direction. As the simulated robot does not have a preferred driving direction, the “passing” can only be recognized if the bearing of the goal position has changed at least by $\frac{\pi}{2}$ from one measurement to the next.

$$\theta = \left| \delta_t \hat{-} \delta_{t-1} \right| > \frac{\pi}{2} \tag{3.21}$$

   This criterion can only be calculated if at least two images have been taken and analyzed while the robot follows a partial trajectory. If two intermediate targets are placed very close, it is possible that the robot may pass both of them before it would have *detected* that it has arrived at the first one. In this case, the robot would head for the second goal that is already behind it, and therefore it would drive back and pass the second target in the wrong direction (cf. figure 3.6). This would prolong the drive and—more importantly—the robot would be in front of the second goal

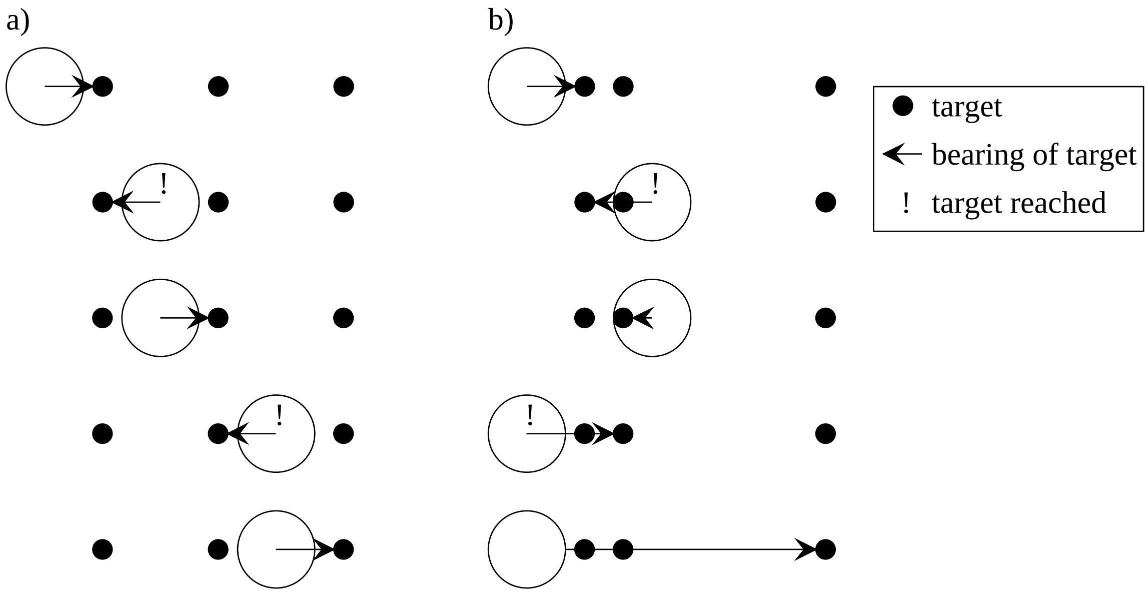

Figure 3.6: Illustration of the robot’s behavior if targets are classified as “reached” when the robot has passed them. a) Normal operation. b) Two successive targets are placed too close.

position instead of being behind it when it starts heading for a third one. If this case has not been taken into account during the filtering process at the end of the training, the robot may lose its way.

2. As an alternative, the robot can use the distance φ that has been calculated by *Panama* to determine when it has reached an intermediate target position. It is assumed that the robot has arrived at an intermediate goal if the value of φ falls below a predefined limit φ_{min}. Thus, φ_{min} has to be selected in a way that ensures that a target position is not classified as reached in the whole area of operations until the robot has come sufficiently close to it. As the distance φ corresponds to the summed translational flow field, according to equation (2.23) on page 35, φ is small where the flow is small, and hence the distance to the surrounding objects is big. Therefore, φ_{min} only has to be chosen in such a way that the robot reaches a goal sufficiently late at the widest place in its area of operations.

$$\theta = \varphi < \varphi_{min} \tag{3.22}$$

The selection of the criterion has been predetermined by the driving behavior described in section 3.2.3 on page 77. The behavior ensures that the robot does not pass an intermediate target while approaching it. This prevents the first alternative from being used for this robot. In contrast, the driving behavior is accommodating towards the second variant: the slowing down in front of intermediate goals ensures that the robot does not shoot past these positions. If it would pass an intermediate target before classifying it as reached, it would be possible that the robot oscillates around this position without ever reaching it, i.e., φ would never fall below φ_{min}. As the robot slows while approaching a goal, φ gets continuously smaller, and therefore it will always reach the value of φ_{min}.

Division Criterion

The division criterion f determines how an image sequence is divided into halves after it has been ascertained that only the last n images fulfill the criterion c. As has already been described in section 3.1.3 on page 67, only a maximum of $n-1$ images can be assigned to the resulting half that does not require any further processing. If this maximum is used, only one image in the second half-sequence preserves the connection to the last image of the first one. Therefore, the robot must reach the intermediate targets very precisely when it is driving along the learned trajectory because if it is heading for the next goal before it really has reached the current one, *Panama* may calculate wrong values and the robot may lose its way.

The simulated robot is not kinematically restricted. Therefore, it is able to reach intermediate targets very precisely. Nevertheless, the utilized "target-reached" criterion does not ensure that a goal position is not classified as reached too early. Therefore, two pictures are assigned to the second partial image sequence instead of only one. This approach is based on the assumption that the "target-reached" criterion may be satisfied in front of the position where the goal image has been taken but behind the position where its predecessor in the sequence has been recorded.

For the simulated robot, the division criterion f is defined as

$$f_n = \max(n-2, 1) \tag{3.23}$$

3.2.4 Obstacle Avoidance

Although the learned trajectories have been taught by a teacher, and therefore they should be passable without collisions, it may be advantageous to provide the robot with an obstacle avoidance. This enables the robot to pass through doorways and to avoid hindrances that have been placed after the training phase as long as they do not impede *Panama*'s matching process. This permits the robot to perform its navigation tasks even after changes in the environment.

The robot utilizes its 32 ultrasonic sensors for obstacle avoidance. These sensors are divided into $k = 16$ groups. Each of these groups contains two sensors that are oriented towards the same direction but that are fitted to the robot at different heights. The sensors of each group are combined to one *virtual sensor*. Each of these k virtual sensors returns the minimum distance measured by the sensors in a group. The measurements of the virtual sensors are limited to a maximum distance d_{max}. As the sensors are fitted around the robot, each sensor "looks" in another direction, relative to the robot's system of coordinates. The distances measured by the sensors are stored in the vector D:

$$\begin{aligned} D &= \begin{pmatrix} d_0 & d_1 & \cdots & d_{k-1} \end{pmatrix} \\ \bigwedge_{i=0}^{k-1} d_i &= \min\left(d_i^0, d_i^1, d_{max}\right) \end{aligned} \tag{3.24}$$

As the sensors are uniformly arranged around the robot's body, they are oriented towards the directions in the following vector:

$$\begin{aligned} A &= \begin{pmatrix} \alpha_0 & \alpha_1 & \cdots & \alpha_{k-1} \end{pmatrix} \\ &= \begin{pmatrix} 0 & \frac{2\pi}{k} & \cdots & (k-1)\frac{2\pi}{k} \end{pmatrix} \end{aligned} \tag{3.25}$$

The avoidance strategy is based on the following considerations: If an obstacle enters the perception range of the sensors, the robot should perform an action that navigates it away from the obstacle. If only a single sensor is considered, the best drive command the robot can execute is to move towards the direction that is exactly opposite the sensors orientation α_i, i.e., the direction $\alpha_i+\pi$ that directly leads the robot away from the obstacle. As length of the movement vector, the distance can be used that the obstacle extends into the sensor's measurement area, i.e. $d_{max} - d_i$. This vector is calculated for each of the k virtual sensors and all these vectors are summed up, resulting in the avoidance vector η. η is the best compromise between all the different avoidance directions that have been determined for the sensors:

$$\eta = \sum_{i=0}^{k-1} (d_{max} - d_i) \begin{pmatrix} \cos(\alpha_i + \pi) \\ \sin(\alpha_i + \pi) \end{pmatrix} \tag{3.26}$$

As the robot should still follow the learned trajectories while avoiding obstacles, the vector η is weighted by the factor λ, and then it is added to the demanded driving vector. Thus, the original movement direction is modified and the robot "slides" past the obstacle. The weighting factor λ should be defined in a way that ensures that the robot will stop in front of an obstacle if it cannot avoid it.

$$\begin{aligned} \gamma' &= \arctan v \\ s' &= |v| \\ &\text{where} \quad v = s \begin{pmatrix} \cos\gamma \\ \sin\gamma \end{pmatrix} + \lambda\eta \end{aligned} \tag{3.27}$$

This kind of obstacle avoidance ensures that the robot passes doorways centered between the door-frame (cf. figure 3.7). The simple avoidance rule generates the following behavior: the robot slows down in front of the doorway, speeds up when it is between the

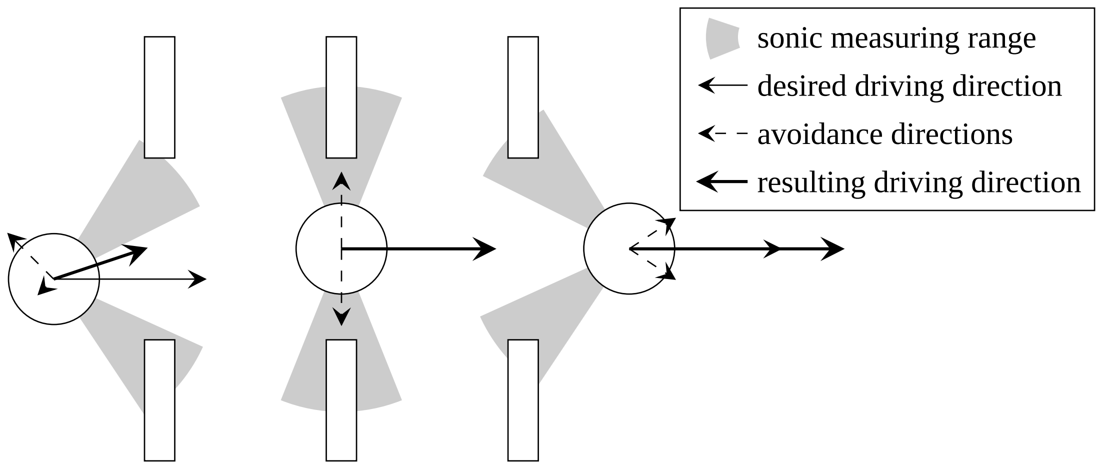

Figure 3.7: Illustration of the simulated robot's obstacle avoidance behavior.

door-frame, leaves the doorway with increased velocity and returns to its normal speed when the doorway departs the sensor range.

3.2.5 Experiments and Results

21 trajectories that are shown in figure 3.4 on page 74 were trained in forward and in backward direction in the simulated flat. The resulting 42 trajectories connect 17 target positions. The panoramic images were taken with a resolution of 256 pixels. All image processing parameters are equal to the default values specified in section 2.5 on page 45. The remaining values are defined as follows: $s_{max} = 100$ cm/s, $\omega_{min} = \frac{\pi}{18}$, $\delta_{min} = \frac{\pi}{18}$, $\tau = 1.2$, $\varphi_{min} = 0.06$ and $d_{max} = 25$ cm.

During the training, the robot determined 76 images necessary to navigate in the scene. The position where these images were taken are visualized in figure 3.8. The number of

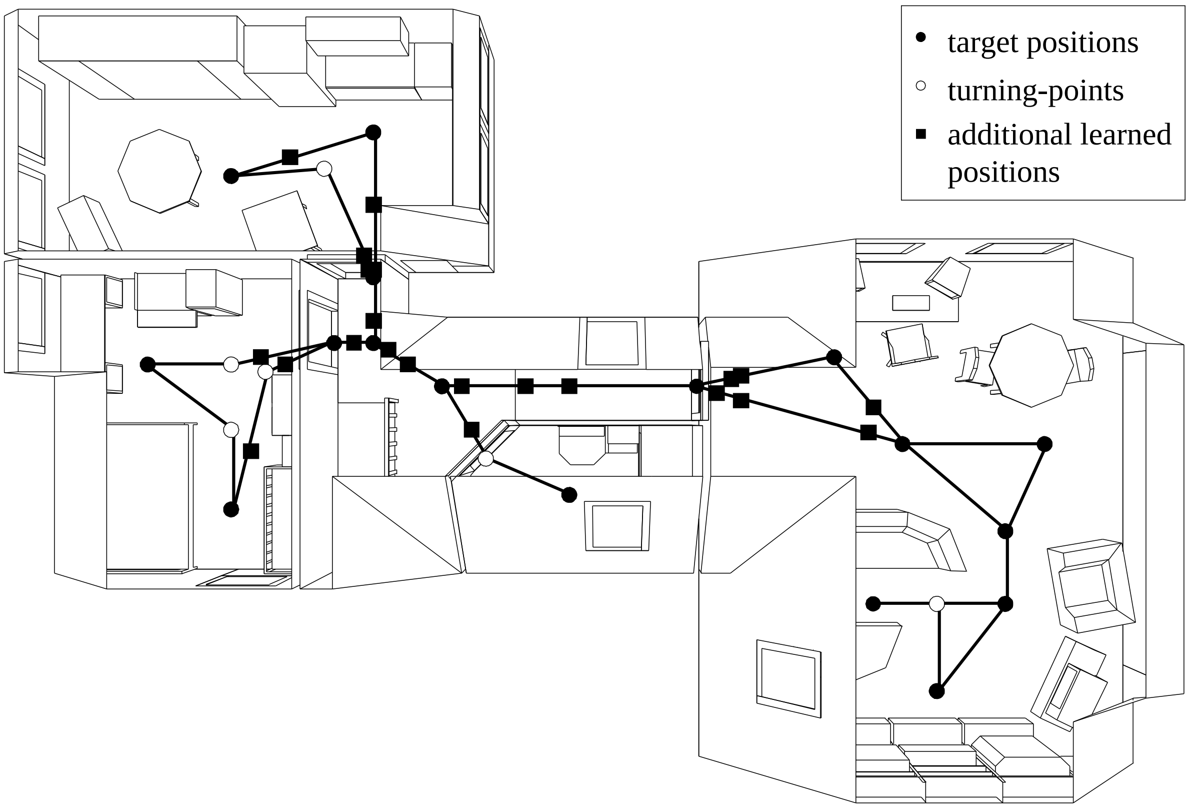

Figure 3.8: The learned trajectories in the simulated flat. The solid dots mark the 17 target positions. The hollow circles represent turning-points in the connecting trajectories. The squares indicate places where additional images were taken.

the preserved pictures is structured as follows:

- 42 images were learned as goal positions of the 42 trajectories.
- 12 pictures were stored because there are six turning-points inside of the trajectories. As all trajectories have to be learned in two directions, this results in the 12 images that were preserved.
- The remaining 22 pictures were learned because the selection criterion c that was presented in section 3.2.3 was not satisfied. Therefore, the image sequences were divided, resulting in these 22 images that the robot kept.

On average, each trajectory was represented by only 1.8 images. If only the number of images that were not learned because of representing a target position or a turning-point is considered, one out of two trajectories required an additional picture. The robot learned many of these images at the positions where its panoramic view of the world changed instantly, e.g. while driving through doorways or passing convex corners (cf. figure 3.8).

The total number of images that represent the 42 trajectories could be further reduced if the pictures taken at the goal positions would be stored more efficiently. In the current implementation, each trajectory contains its own copy of the target image. Therefore, the 17 positions are represented by 42 pictures, i.e. one for each trajectory. If each goal image would only be stored once, the total number of pictures would be reduced from 76 to 51.

While the robot followed the learned trajectories, the deviations between the trained path and the reproduced one always were less than 10 cm. Deviations larger than 2 cm can mostly be attributed to corrections performed by obstacle avoidance because some trajectories run too close to the walls.

3.3 Controlling a Wheelchair

In this section, the implementation of the navigation method that has been described in section 3.1 will be presented on a kinematically restricted, real robotics platform. An electric wheelchair is used as a mobile basis that has been equipped with a computer and numerous sensors.

3.3.1 The Wheelchair

The wheelchair used is shown in figure 3.9. It has a driving axle and a steering axle. The driving axle is at the front of the wheelchair. Its wheels are connected with the motor by a differential gear. The steering is at the wheelchair's back. Therefore, it gives the wheelchair the moving behavior of a car driving backwards. The top speed is 6 km/h and the minimum turning circle is about 4 m. During the experiments, it has been driven with a constant velocity of $s = \pm s_{const} = \pm 20$ cm/s. If s is positive, the wheelchair will drive in forward direction. Otherwise, it will move backwards. The wheelchair's acceleration can be neglected because of the low driving speed; therefore, it has not been measured. The wheelchair takes up to three seconds to set a steering angle. Thus, the vehicle may drive up to 60 cm before a demanded steering angle has been reached.

In this thesis, the steering angle is never used directly. Instead, the wheelchair's steering behavior is described in terms of the *steering radius*. This is the radius of a circle-segment's border on which the wheelchair drives when it has set a certain steering angle. The steering radius is not related to the outer wheels' trajectory as often used for the specification of vehicles' turning circles. Instead, it describes the movement of the driving axle's center. Whereas the effect of the steering angle on the performed trajectory depends on the wheelchair's wheelbase, the steering radius is independent of it. A negative steering radius describes a curve to the left and a positive one represents a curve to the right. In addition, a radius of zero characterizes a straight ahead movement.

The wheelchair's extent is defined by a rectangular frame that encloses the vehicle at a height of 30 cm. The frame is 134 cm long and 71 cm wide. It is equipped with twelve

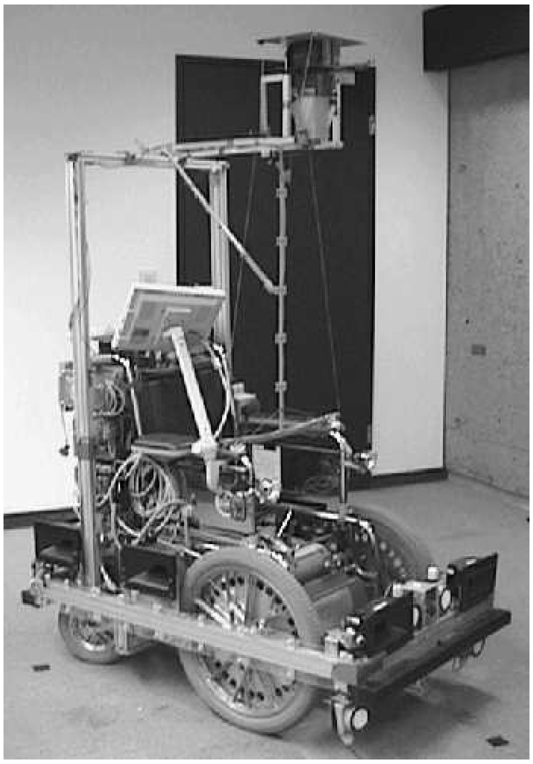

Figure 3.9: The wheelchair with the sensor for taking panoramic images.

tactile sensors, six infrared sensors, and 16 ultrasonic sensors. The tactile sensors are all fitted around the frame, and so they enable the wheelchair to detect every contact with obstacles in the height of the frame. The infrared sensors can only detect whether there is an obstacle within a radius of approximately 15 cm; however, they cannot measure the distance. Two different kinds of ultrasonic sensors are fitted to the wheelchair: half of the sensors have an opening-angle of 80° while the other half only measure in a range of 7°. The arrangement of the different sensors is shown in figure 3.10.

The rotations of the front wheels can be measured. Therefore, the wheelchair can perform dead reckoning. Nevertheless, self-localisation that is based on such odometer measurements becomes more and more faulty with increasing driving distance, and hence it cannot be used over longer distances.

To allow autonomous operation, the wheelchair is equipped with a Pentium 100 personal computer and five micro-controllers that are connected via a CAN-BUS. The micro-controllers are utilized for real-time hardware control whereas the PC is used for higher control tasks and navigation. The software architecture of the control system will be presented in appendix A.

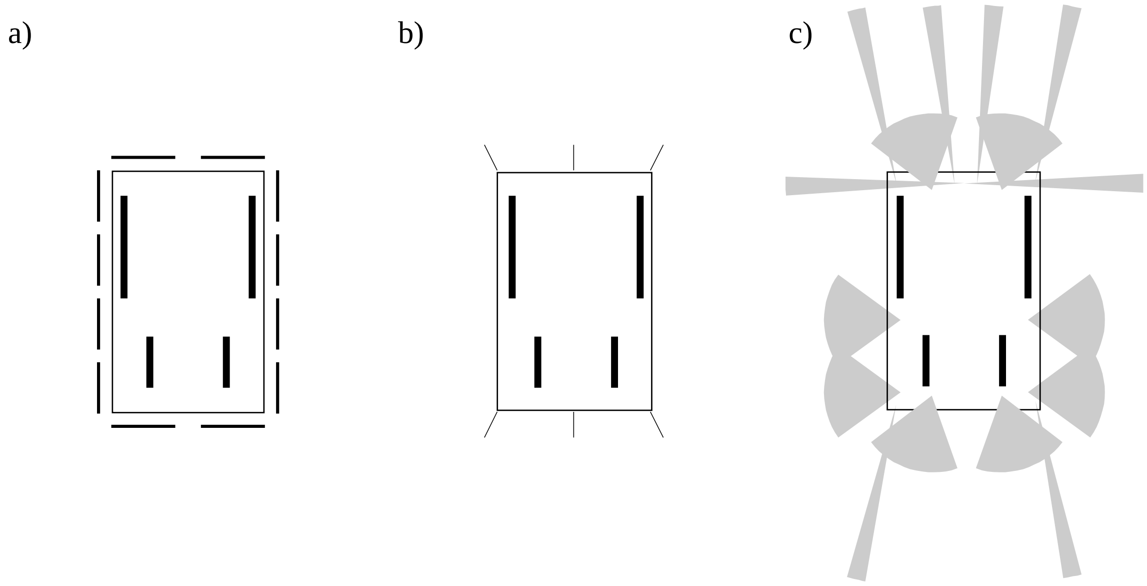

Figure 3.10: Illustration of the wheelchair's sensory equipment. The long, thick black lines visualize the wheelchair's front wheels. a) The twelve tactile sensors. b) The six infrared sensors. c) The 16 ultrasonic sensors.

3.3.2 Sensor for Taking Panoramic Images

Structure of the Sensor

Above the center of the driving axle at a height of 170 cm, the wheelchair is equipped with an apparatus that enables it to take panoramic images (cf. figure 3.11a). The apparatus consists of a camera that is facing upward and a spherical mirror that is mounted above the camera. The camera looks onto the reflecting sphere and sees the surroundings shown on it. The picture contains a complete panoramic view. The vertical opening angle reaches more than 45° upwards and 90° downwards, enabling the camera to see itself. An image taken by the camera is shown in figure 3.11b.

There is a circular region inside the image of the sphere that shows the part of the surroundings that is at the same height as the sphere itself. If the sphere is moved along the horizontal plane, all objects that are visible in this circle only move along the circle. Outside of the circle, the objects also move towards or away from the sphere's center. Therefore, only the circle shows a reflection of the horizontal plane. Thus if the pixels of this circle are extracted and combined to a vector, this vector corresponds to the specification of a panoramic image that has been given in section 2.1.

Calculation of a Panoramic Image

To calculate a panoramic image, the picture of the camera has to be digitized with a framegrabber card. The digitized image can be described as a matrix P that contains the pixels of the image. The matrix is w pixels wide and h pixel high. Each pixel consists of

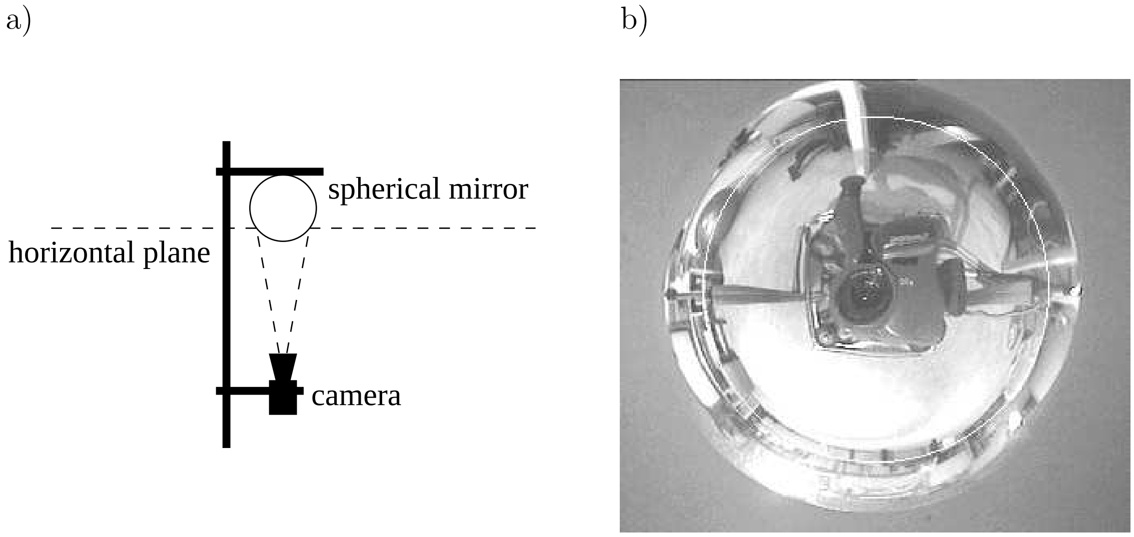

Figure 3.11: Illustration of a technique to take panoramic images.

the intensities of the three color components red, green and blue:

$$P = \begin{pmatrix} p_{0,0} & \cdots & p_{w-1,0} \\ \vdots & \ddots & \vdots \\ p_{0,h-1} & \cdots & p_{w-1,h-1} \end{pmatrix}$$
$$\bigwedge_{i=0}^{w-1} \bigwedge_{j=0}^{h-1} p_{i,j} = \begin{pmatrix} r_{i,j} \\ g_{i,j} \\ b_{i,j} \end{pmatrix} \tag{3.28}$$

To determine a panoramic image from the camera picture P, the center (x, y) and the radius r of the circle must be known. These values have to be predefined for the combination of a camera and a sphere in a calibration step.

To extract a panoramic image, the function Q' enumerates all pixels in the picture P that lie on the circle with the radius r around the center (x, y). It uses the function Q'' that generates a list of the pixels in one quadrant of the circle. Then, it combines the four pixel lists—each for a different quadrant—to a single one. The algorithm for the enumeration of the circle's pixel comes from computer graphics (Foley *et al.* 1990). Inside the camera image, the pixels are enumerated in clockwise direction. As the camera looks onto the mirror from below and this inverts the order of the objects shown, from the bird's-eye view this corresponds to the counterclockwise direction that has been defined as pixel-order for panoramic images in section 2.1. How the quadrants are combined depends on the orientation of the camera on the wheelchair. They are ordered in a way that ensures that the first pixel in the resulting list is always oriented perpendicular toward the back of the wheelchair.

$$\begin{aligned} Q'^{P}_{x,y,r} &= Q''^{sw}_{r-1,0} + Q''^{nw}_{r-1,0} + Q''^{ne}_{r-1,0} + Q''^{se}_{r-1,0} \\ \text{where} \quad Q''^{f}_{i,j} &= \begin{cases} [] & \text{if } i < 0 \\ [f_{i,j}] + Q''^{f}_{i-1,j} & \text{if } i \geq 0 \wedge r^2 - i^2 \leq (j+1)^2 \\ [f_{i,j}] + Q''^{f}_{i,j+1} & \text{if } i \geq 0 \wedge r^2 - i^2 > (j+1)^2 \end{cases} \\ nw_{i,j} &= p_{x-i-1,y-j-1} \\ ne_{i,j} &= p_{x+j,y-i-1} \\ sw_{i,j} &= p_{x-j-1,y+i} \\ se_{i,j} &= p_{x+i,y+j} \end{aligned} \tag{3.29}$$

The number of pixels in the list generated by Q' depends on the circle's radius r. To obtain a panoramic image with a predefined number of pixels n, the elements of the list determined by Q' have to be assigned to the n pixels. As the length of the list is no multiple of n in general, the assignment has to be performed unevenly. The function R divides the pixel list into n partial lists. From each of these lists, one pixel of the resulting image is generated by calculating the average color of the partial list's pixels.

$$
\begin{aligned}
R^n_L &= \left[\tfrac{1}{\#r}\sum r \mid r \in R'^{\#L,n,0}_{L,[]}\right] \\
&\quad \text{where} \quad R'^{m,n,l}_{L,S} = \begin{cases} [] & \text{if } L = [] \\ R'^{m,n,l+n}_{L',S+[h]} & \text{if } L \neq [] \wedge l+n < m \\ [S] + R'^{m,n,l+n-m}_{L',[]} & \text{if } L \neq [] \wedge l+n \geq m \end{cases} \\
&\qquad\qquad \text{where} \quad h = \mathrm{hd}_L \\
&\qquad\qquad\qquad\quad\; L' = \mathrm{tl}_L
\end{aligned}
\tag{3.30}
$$

The function Q calculates a panoramic image with n pixels. It determines the list of all pixels on the circle with the radius r around the center (x, y) in the camera picture P and uses the function R to convert this list into a shorter one with exactly n entries.

A pixel in the panoramic image is formed from multiple pixels of the list $Q'^P_{x,y,r}$, i.e., each pixel represents a part of the surroundings that is placed in a certain range of angles relative to the wheelchair. The viewing direction of the pixel is the angle that is centered inside of this range. As described earlier, the viewing direction of the first pixel in the list that is generated by Q' is perpendicular to the wheelchair's back. To also use this direction for the first pixel of the panoramic image, the list $Q'^P_{x,y,r}$ is rotated before it is reduced to n elements by R. The amount of rotation is defined by the number of entries that corresponds to the size of half of a pixel in the resulting panoramic image. As this resulting list has a fixed length, it is converted to a vector with the same number of elements. This vector is a panoramic image with n pixels.

$$
\begin{aligned}
Q^{P,n}_{x,y,r} &= \mathrm{vector} R^n_B \\
&\quad \text{where} \quad A = Q'^P_{x,y,r} \\
&\qquad\qquad\quad n' = \#A - \tfrac{\#A}{2n} \\
&\qquad\qquad\quad B = \mathrm{drop}^A_{n'} + \mathrm{take}^A_{n'}
\end{aligned}
\tag{3.31}
$$

To further reduce aliasing effects in the generated panoramic images, these are combined from several rings. Instead of using a single circle with a single radius, a range of radii is used to determine the panoramic pictures. Hence, a panoramic image I is calculated as the mean of circles with radii between r and r':

$$
I^{P,n}_{x,y,r,r'} = \frac{1}{r' - r + 1} \sum_{i=r}^{r'} Q^{P,n}_{x,y,i} \tag{3.32}
$$

Limitations

A simple color camera that is equipped with a single CCD-chip has been used for taking the panoramic images. Thus, the different color channels are interfered by high noise, especially with low ambient illumination. If the surroundings are lighted up heavily, the noise is reduced but still remains on a level of ±5% of a color channel's maximum intensity. To also reduce the noise under poor ambient illumination conditions, it is necessary to use a camera with separate CCD-chips for each color channel.

Nearly the whole surroundings are reflected on the spherical mirror. Thus, the camera sees regions with very different illumination conditions. If the exposure is adapted to the darker regions, the bright areas are digitized completely white. Otherwise, if the exposure is adjusted for the bright regions, the dark areas are grabbed nearly black and very noisy. As the camera images are digitized, each color is represented by a fixed number of bits. Therefore, it is impossible to improve the images' color resolution by a subsequent digital brightening because this does not increase the number of color steps. The only possible solution is to reduce the images' contrast before the digitization. To achieve this, either a locally adaptive camera has to be used or a camera that transforms the images' intensity to a logarithmic scale.

As neither a 3-chip-CCD-camera, nor a locally adaptive optical sensor, nor one with a logarithmic mapping was at disposal, the surroundings were illuminated heavily and uniformly to enable the presented method to work with the available camera.

3.3.3 Hardware-Specific Arrangements

As has been described in section 3.2.3 for the holonomic robot, the platform-dependent parts of the navigation method are presented in the following subsections. As the kinematics of the wheelchair heavily differ from those of the simulated robot, the implementation details are completely different but the properties that have to be specified remain the same:

- The selection criterion that determines which images are required for the representation of the trained trajectory.
- The driving behavior of the wheelchair when it is repeating the learned path autonomously.
- The criterion that is used to detect when the wheelchair has arrived at an intermediate target.
- The division criterion that determines where to divide an image sequence into halves after it has been established that at least one of the images in this sequence does not fulfill the selection criterion.

Selection Criterion

The most important property that distinguishes the wheelchair from the holonomic robot is its driving behavior. In contrast to the robot that has been described in section 3.2, the possible movements of the wheelchair depend on its orientation. Thus, it always has to drive along curved trajectories.

To permit a vehicle with these restricted kinematics to reach a position with the correct orientation, an exact path-planning is usually required. This would be performed based on the wheelchair's placement relative to the aspired goal position. As *Panama* only supplies the bearing direction to the next partial target and the rotation relative to it, an exact path-planning based on this information is impossible because no metrical distance is known.

To use *Panama* for controlling the wheelchair, the vehicle's mobility is reduced during the teaching phase. The wheelchair is only trained with trajectories that consist of straight lines and segments of continuous curvature, i.e. parts of a circle. Hence, between two positions on a circular segment, there is a fixed relationship between the bearing direction and the rotation. As shown in figure 3.12, the bearing of one position relative to the other is half as large as the rotation between them. If the goal position is located behind the wheelchair, it is necessary to rotate the bearing direction by π to regain the same relationship between the two values. This relation is only valid if the camera is mounted above the center of the driving axle because this is the only position that is always on the driven curve. At all other locations on the wheelchair, the correlation between the bearing direction and the rotation is influenced by the radius of the performed curve.

$$c = \varphi \, |\tan \varepsilon| < \varepsilon_{max} \tag{3.33}$$

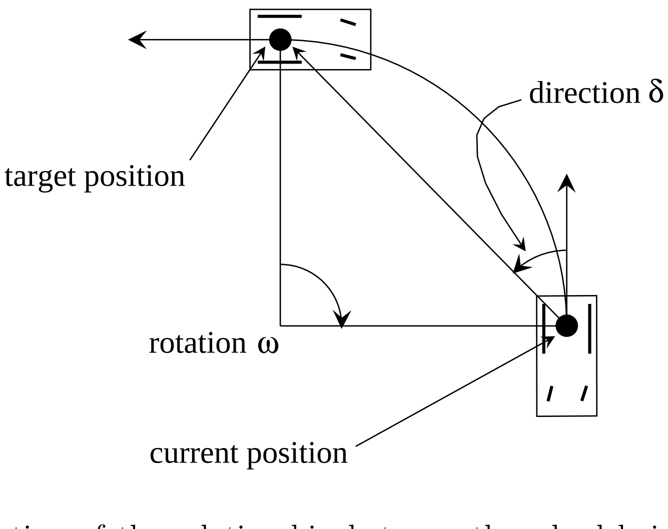

Figure 3.12: Illustration of the relationship between the wheelchair's rotation and the direction to the target position.

Under these preconditions, the values that are calculated by *Panama* should be related in the following ratio:

$$-\frac{\omega}{2} = \begin{cases} \delta \hat{-} \pi & \text{if } |\delta| > \frac{\pi}{2} \\ \delta & \text{otherwise} \end{cases} \tag{3.34}$$

As *Panama* is unable to determine ω and δ precisely, equation (3.34) will never be satisfied. Instead, there will always exist a small error ε that is the difference between both sides of equation (3.34):

$$\varepsilon = \frac{\omega}{2} + \begin{cases} \delta \hat{-} \pi & \text{if } |\delta| > \frac{\pi}{2} \\ \delta & \text{otherwise} \end{cases} \tag{3.35}$$

If ε exceeds a tolerated maximum value of ε_{max}, it is assumed that *Panama* has determined the rotation and the bearing direction incorrectly. If two images have been taken at positions close to each other, the determination of the bearing direction δ can be very imprecise because the translational flow field between the two images is very small. To compensate this effect, the size of the flow field φ is included in the check of the values' accuracy. The resulting condition is used as selection criterion c:

$$c = \varphi\,|\tan\varepsilon| < \varepsilon_{max} \tag{3.36}$$

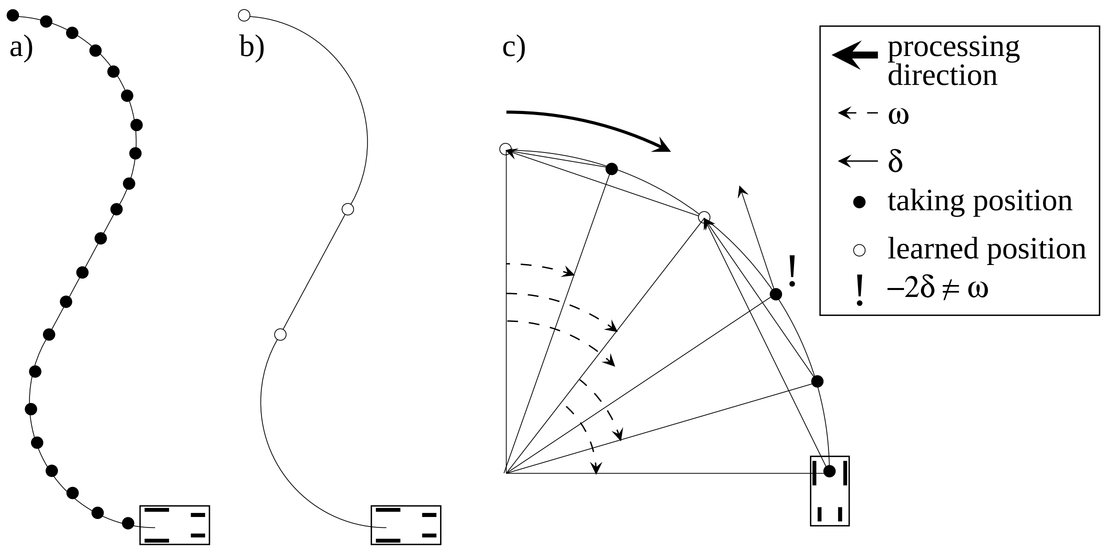

Figure 3.13: Illustration of the three steps of the learning process. a) The recording of the images. b) The division into partial sequences with constant driving direction. c) The filtering of the images for each partial sequence.

Figure 3.13 summarizes the selection process: first, a number of images is recorded. Then, the image sequence is divided into partial sequence with constant driving direction. Finally, in each of these partial sequences, the described selection process is performed.

Driving Behavior

The driving behavior is based on the same relation between the bearing direction and the rotation as the selection criterion. As *Panama* does not determine metrical distances, no path-planning is performed. Instead, the wheelchair takes panoramic images during its autonomous drive and uses *Panama* to calculate the bearing direction and the rotation to the next intermediate target. The whole navigation is performed by reducing the error ε from equation (3.35) by adequate actions.

The wheelchair tries to compensate for the error ε in a certain time period Δt. To achieve this, it sets the steering radius in a way that the wheelchair's rotation will match the current bearing direction after the time period Δt (cf. figure 3.14). Therefore, it has to drive an arc that turns its body by the angle 2ε. Thus, the required radius r has to be calculated as

$$r = \begin{cases} \frac{-\Delta t |s|}{2\varepsilon} & \text{if } \varepsilon \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.37}$$

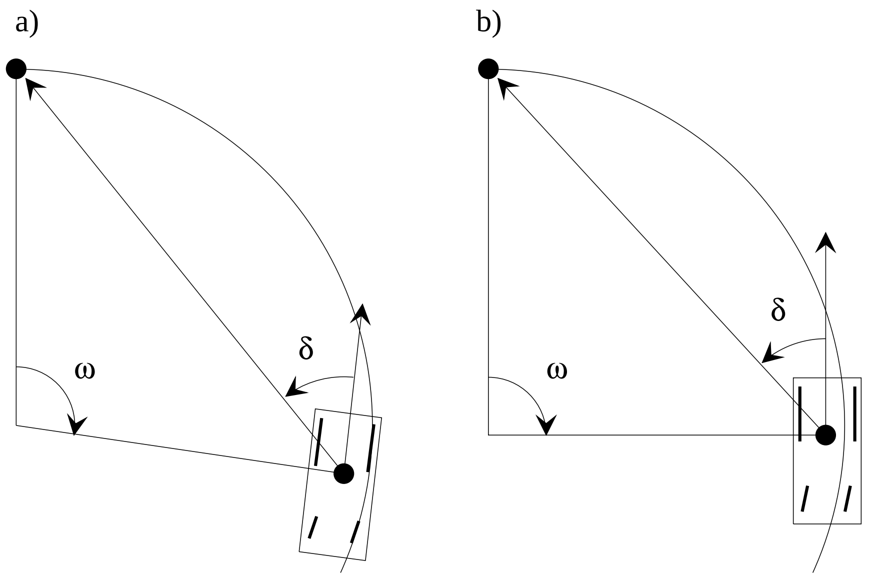

Figure 3.14: Illustration of the wheelchair's driving behavior. a) The rotation and the bearing direction do not satisfy equation (3.34). b) As a result, the wheelchair adapts its rotation to the bearing direction in a).

The driving direction is selected in correlation to the bearing direction to the next intermediate goal. The wheelchair's velocity is always s_{const}:

$$s = \begin{cases} s_{const} & \text{if } |\delta| < \frac{\pi}{2} \\ -s_{const} & \text{otherwise} \end{cases} \tag{3.38}$$

As shown in figure 3.15, a target position can be reached from many different positions with this simple approach even if they are not on a common arc with the goal position. As long as *Panama* determines the rotation and the bearing direction correctly, this navigation method is very robust.

In the practical application, two problems arise in the proximity of intermediate targets: on the one hand the bearing direction cannot be calculated very precisely since the translational flow field's smallness, on the other hand the wheelchair may arrive at a goal position with a small deviation to the side. Thus, during approaching a goal, the wheelchair locates the target position more and more sideways. If the wheelchair would try to compensate for these errors, it has to perform heavy steering movements shortly before arriving at the goal. As a result, it may reach the position more precisely but in a completely wrong orientation.

To avoid this problem, equation (3.38) is enhanced. The more the wheelchair approaches the target position the more it goes over to only reducing the rotation relative to the goal instead of adapting it to the bearing direction. The weighting of the two behaviors depends on the distance φ. Thus, the influence of the bearing direction on the

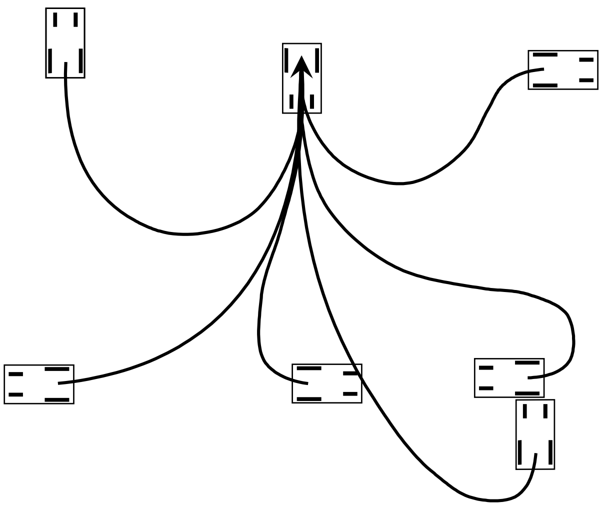

Figure 3.15: Simulation of the driving behavior that is defined in equation (3.37). The wheelchair reaches the target position from various starting places.

wheelchair's control is reduced in the same extend as the sum over the translational flow field decreases, i.e., the bearing direction is not used if it is already inaccurate because of the small optical flow field.

$$
\begin{aligned}
r &= \begin{cases} \frac{-\Delta t|s|}{2\varepsilon'} & \text{if } \varepsilon' \neq 0 \\ 0 & \text{otherwise} \end{cases} \\
\text{where} \quad \varphi' &= \begin{cases} 1 & \text{if } \varphi > \varphi_{max} \\ 0 & \text{if } \varphi < \varphi_{min} \\ \frac{\varphi - \varphi_{min}}{\varphi_{max} - \varphi_{min}} & \text{otherwise} \end{cases} \\
\varepsilon' &= \varphi'\varepsilon - (1 - \varphi')\omega
\end{aligned}
\tag{3.39}
$$

The radius that has been calculated in equation (3.39) can become very small. As the wheelchair is only able to set a minimum steering radius of $\pm r_{min}$, an adapted radius r' is determined that takes the wheelchair's limitation into account:

$$
r' = \begin{cases} r_{min} & \text{if } r \in \,]0 \ldots r_{min}] \\ -r_{min} & \text{if } r \in [-r_{min} \ldots 0[\\ r & \text{otherwise} \end{cases}
\tag{3.40}
$$

"Target-reached" Criterion

The "target-reached" criterion is utilized by the wheelchair to detect whether it has arrived at an intermediate goal. The criterion that has been used for the simulated robot in section 3.2.3 on page 77 classifies a target as reached if the summed optical flow falls below a certain value. As previously discussed, the robot has been able to use the criterion because it slows down when it is approaching a goal position. In contrast, the wheelchair drives with a constant speed s_{const} as has been specified in equation (3.38) on page 93. Therefore, it has to use another method for detecting the arrival at a target position.

The wheelchair uses the first criterion that has also been presented in section 3.2.3 on page 77. In this method, a goal has been reached if its bearing is in opposite direction to the wheelchair's current driving direction, e.g. if the wheelchair is moving forwards and the target is located behind it. In contrast to the holonomic robot, there is a clear distinction between driving forwards and driving backwards. Therefore, the difficulties mentioned can be avoided if the wheelchair stores, for every target position, whether it has to drive forwards or backwards to reach it. This information can be obtained during training. The wheelchair uses this data when it is repeating the trained trajectories to determine the required driving direction as well as to detect whether it has passed a goal:

$$
s = s_{stored}
\tag{3.41}
$$

$$\theta = \begin{cases} |\delta| > \frac{\pi}{2} & \text{if } s_{stored} > 0 \\ |\delta| < \frac{\pi}{2} & \text{if } s_{stored} < 0 \end{cases} \tag{3.42}$$

As the wheelchair does not detect the arrival at the target until it has passed this position, it would shoot past the final position of a trajectory. To avoid this behavior, the last image of a trajectory is not stored at all, but instead, the last but one. If the wheelchair detects that it has passed this next to last position, it will stop, and therefore it will nearly reach the position where the trajectory has also ended during the training.

Division Criterion

The division criterion determines how an image sequence is divided into halves after it has been ascertained that only the last n images fulfill the selection criterion. As has been described in section 3.1.3 on page 67, only a maximum of $n - 1$ images can be assigned to the half-sequence that will only be represented by its last image after the filtering process. If this maximum is used, only one image in the second half sequence preserves the connection to the last image of the first one. Although the chosen "target-reached" criterion ensures that the wheelchair does not switch to the next intermediate goal before it has passed the current one, it may arrive at a target position with a deviation to the side. Therefore, the wheelchair's visual impression of the surroundings may differ from the one that it has had during training. In addition, the images that have been taken by the CCD-camera are very noisy. Thus, it is possible that the wheelchair has been able to correctly match a picture recorded at a certain position with a stored one during training but it is unable to match another image taken at the same position when it is driving autonomously. For these two reasons, a criterion for dividing the image sequence has been selected that has a large allowance for security. Only half of the n images that represent positions where the wheelchair can reach the last position of the partial sequence are categorized as reliable. Therefore, the division criterion is defined as

$$f_n = \max\left(\frac{n}{2}, 1\right) \tag{3.43}$$

3.3.4 Obstacle Avoidance

As has been presented in section 3.3.1 on page 83, the wheelchair contains a driving axle and a steering axle. The resulting kinematics cause the wheelchair's rear to swing out heavily during steering maneuvers. Thus, there is a risk that it collides with obstacles in its neighborhood even while performing minor directional changes.

Although the navigation method described so far considers the limited mobility of the wheelchair with regard to the arrival at a certain position, it does contemplate colliding with obstacles. As the method always tries to compensate for the measured errors in a short period of time, it often generates extreme steering commands that may cause collisions. As it is impracticable to demand that the teacher ensures that all trajectories keep a large enough distance to obstacles for the wheelchair to perform any possible steering maneuver, bumping into hindrances has to be avoided by the use of additional sensors.

As for the simulated robot in section 3.2, the on-board ultrasonic sensors can be used for obstacle avoidance. The implemented variant does not include a universal avoidance strategy as the method that has been described in section 3.2.4. Instead, it only avoids collisions to the sides during curved drives. For this, only uses four of the wheelchair's 16 ultrasonic sensors.

The utilized sensors are fitted to the flanks of the wheelchair's back, two on each side. Each pair of adjacent sensors is used as a single virtual sensor by only using the shorter distance that has been measured by one of these sensors:

$$u_{left} = \min\left(u^0_{left}, u^1_{left}\right) \tag{3.44}$$

$$u_{right} = \min\left(u^0_{right}, u^1_{right}\right) \tag{3.45}$$

The wheelchair avoids collisions at the sides by adapting the steering radius that is demanded from the navigation control. It enlarges the steering radius depending on the measured distances. A bigger radius causes the wheelchair to drive a trajectory with a lower curvature, resulting in less swinging out of the wheelchair's rear. Which of the two distance values u_{left} and u_{right} is relevant for this depends on the wheelchair's current driving direction (forwards/backwards) as well as on its steering direction (left/right). If

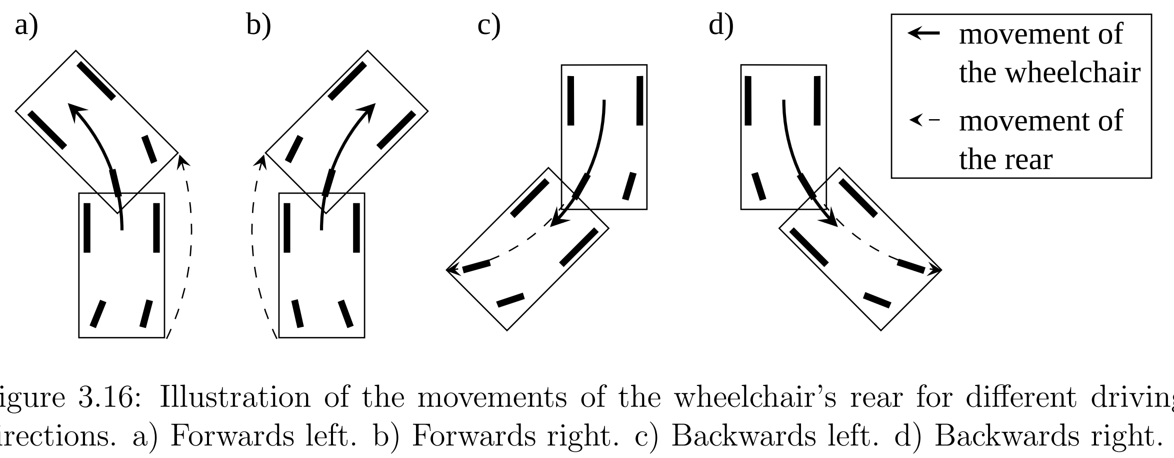

Figure 3.16: Illustration of the movements of the wheelchair's rear for different driving directions. a) Forwards left. b) Forwards right. c) Backwards left. d) Backwards right.

driving forwards, the rear of the wheelchair swings into the direction that is opposite to the steering direction, whereas if the wheelchair is driving backwards, the back moves in the same direction as the whole vehicle. The four different cases are illustrated in figure 3.16.

If a sensor measures a distance that is shorter than a threshold u_{min} in the movement direction of the wheelchair's rear, the steering radius is enlarged. The quantity of enlargement depends on the measured distance as well as on a predefined weighting factor κ. As the wheelchair always drives with a constant velocity s_{const}, the speed is ignored in collision detection. If the wheelchair's velocity would be dynamic, u_{min} has to be adapted to the current speed s continuously.

The calculation of the adapted steering radius r''—considering the different movement directions—is defined as follows:

$$
\begin{aligned}
r'' &= \begin{cases} 0 & \text{if } r' = 0 \\ fr' & \text{if } r' \neq 0 \wedge s > 0 \\ br' & \text{if } r' \neq 0 \wedge s < 0 \end{cases} \\
\text{where} \quad f &= \begin{cases} \frac{u_{min}+\kappa-u_{right}}{\kappa} & \text{if } r' < 0 \wedge u_{right} < u_{min} \\ \frac{u_{min}+\kappa-u_{left}}{\kappa} & \text{if } r' > 0 \wedge u_{left} < u_{min} \end{cases} \\
b &= \begin{cases} \frac{u_{min}+\kappa-u_{right}}{\kappa} & \text{if } r' > 0 \wedge u_{right} < u_{min} \\ \frac{u_{min}+\kappa-u_{left}}{\kappa} & \text{if } r' < 0 \wedge u_{left} < u_{min} \end{cases}
\end{aligned} \tag{3.46}
$$

3.3.5 Experiments and Results

The experiments have been performed in two different surroundings: in a nearly quadratic room and in a long narrow corridor. The room has a size of 7×8 m. The corridor is approximately 18 m long an 1.7 m wide. Both test surroundings are shown in the figures 3.17a/b and 3.17c. As the camera is fitted on the wheelchair at a height of 170 cm, it has been able to perceive only few details from the different surroundings. In the room, it has only "seen" the walls, the window frontage, two doors, a column, a bookshelf and two posters. In the corridor, only the walls, two windows, five doors and one glass door have been visible. As the light shining through the windows would have caused a very uneven brightness distribution, all experiments have been performed without daylight. Four powerful spotlights and the ceiling lights have provided for a uniform illumination. In the corridor, only the ceiling lights have been used.

The experiments have been conducted with an image resolution of $n = 256$ pixels. All image processing parameters but one have been set to the defaults described in section 2.5 on page 45. As the only exception, the bearing direction has determined using the first alternative of the solutions presented in section 2.3.3 because of the high noise level of the recorded pictures. The remaining values have been defined as follows: $s_{const} = 20$ cm/s,

$\varepsilon_{max} = 0.2$, $\Delta t = 1$ s, $\varphi_{min} = 0.1$, $\varphi_{max} = 0.3$, $r_{min} = 85$ cm, $u_{min} = 60$ cm and $\kappa = 10$ cm.

During the training and the autonomous drives of the wheelchair, its position in space has been determined with the help of the onboard odometers. Thus, the position has been recorded simultaneously with the taking of the panoramic images. This enables a detailed documentation and analysis of the performed experiments. Because of the shortness of the trained paths, the error in dead reckoning has been minimal. Therefore, it has influenced the analysis only slightly. The measured trajectories are visualized in figure 3.17.

During the wheelchair's autonomous reproduction of the trained trajectories, it has determined the distance φ, the rotation ω, and the bearing direction δ for each picture

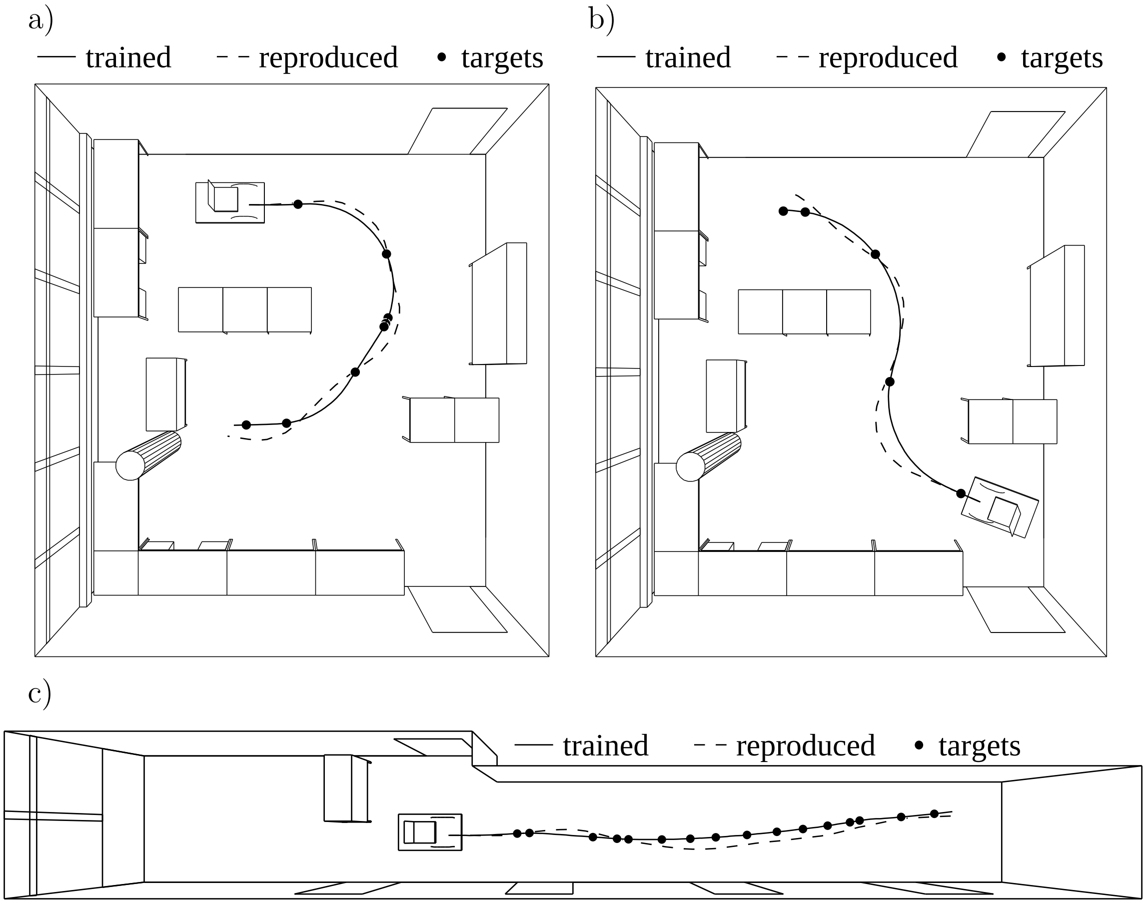

Figure 3.17: Illustration of the trajectories performed by the wheelchair. The trained trajectories are shown as solid lines. The dots mark the positions where permanently stored images have been taken at. Based on these images, the wheelchair has reproduced the trajectories that are displayed as broken lines.

that it has taken. As the positions where each image has been taken have been recorded, it has been possible to determine the desired values for ω and δ afterwards for analysis purposes. These values have been compared with the actually calculated ones. The results of this comparison are the error graphs shown in the right column of figure 3.18.

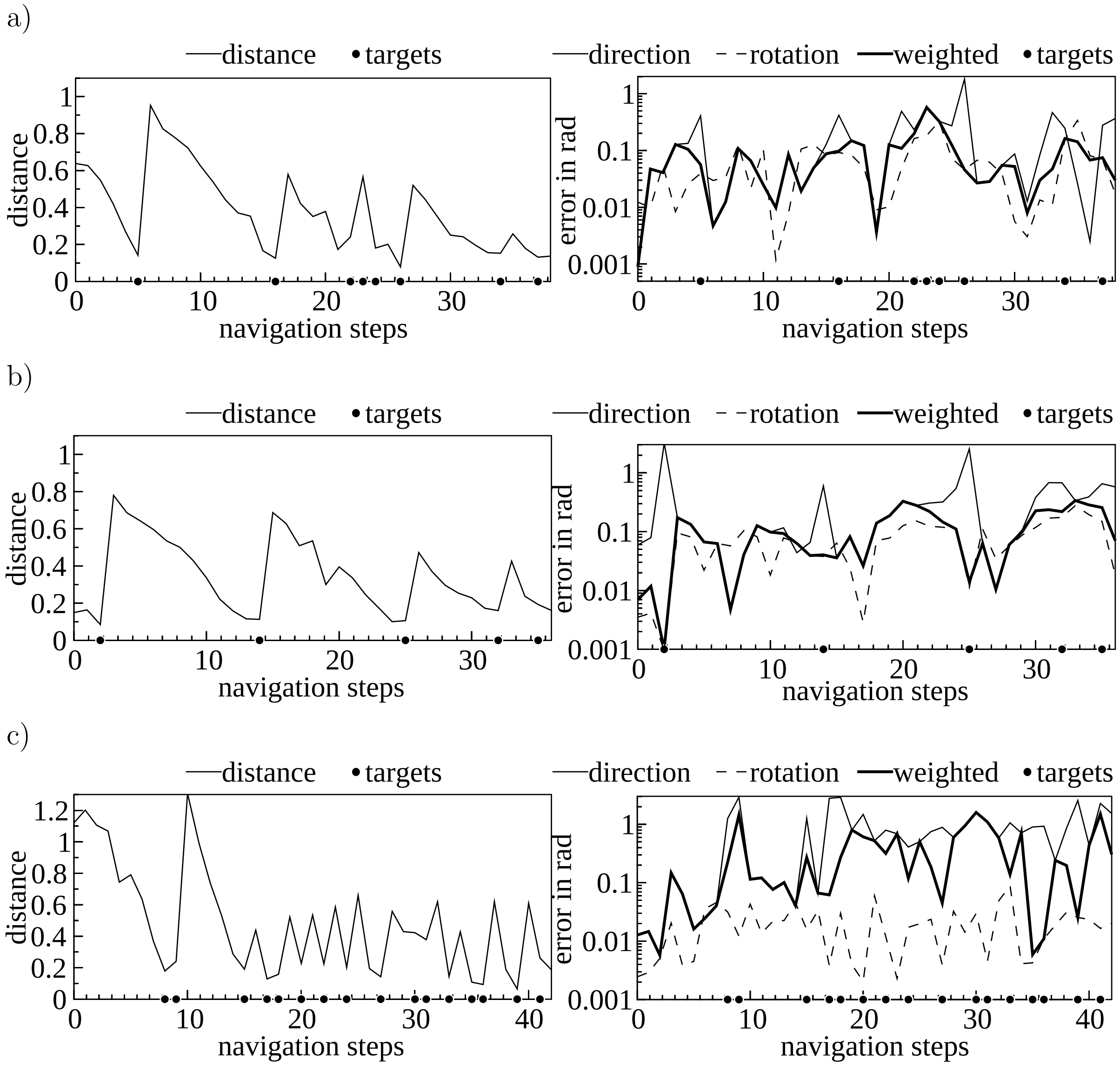

Figure 3.18: On the left, the determined distance φ during the autonomous drive is presented for each of the three trajectories shown in figure 3.17. On the right, the errors in the calculation of the rotation, the direction, and a weighted combination of both values are displayed.

In the left column of figure 3.18, the course of the calculated distance φ during the autonomous drive is shown. It is clearly visible that the distance decreases while the wheelchair got increasingly closer to an intermediate goal position. On the right hand side, it is apparent that the precision of the bearing direction has been dependent on the proximity to the next target as well. The closer the wheelchair has come to an intermediate goal, the more inaccurate the calculation of the direction to this position has become. In contrast, the determination of the rotation has been independent of the closeness to the intermediate targets.

As the wheelchair has increasingly used the rotation ω for the determination of the driving direction when φ has become smaller, the large errors in the estimation of the direction have toned down. If both error graphs are weighted in a way that corresponds to their influence on the determination of the driving commands, this results in the bold graphs shown in figure 3.18.

In the experiments a) and b), the weighted error has mostly been smaller than 0.1, i.e. less than 5°. In the corridor experiment, the error has often been bigger. There are two reasons for this: on the one hand the walls have been much closer to the wheelchair and this has resulted in a larger optical flow, on the other hand one of the two walls has not provided any visual cues, and therefore it has impeded the matching process. The wheelchair still succeeded in following the trajectory because the learned positions have been so close that φ has never become large. Therefore, the wheelchair has only used the rotation for the navigation over wide ranges of the path. This shows that the method is able to cope with actually unsuitable situations.

The number of learned images, the maximum errors of the wheelchair's position, and the average errors in the determination of the rotation and the direction are summarized in table 3.1.

	learned images	maximum errors in position	average errors		
			direction	rotation	weighted
trajectory a	8	30 cm	0.19	0.07	0.08
trajectory b	5	31 cm	0.36	0.08	0.11
trajectory c	16	22 cm	0.81	0.02	0.37

Table 3.1: The number of learned images, the maximum errors in the wheelchair's position and the average errors (in radians) in the determination of the direction, the rotation and a weighted combination of both values.

3.4 Summary

In this chapter, a navigation method for autonomous mobile systems has been presented that is based on the image processing algorithm *Panama* that has been introduced in the previous chapter. In this approach, a teacher trains a robot by controlling it along certain trajectories. The robot takes panoramic images of its surroundings with an optical sensor and uses these pictures to describe the trained path in a way that enables it to follow the path autonomously. It chooses exactly the sequence of images that it needs, considering its capabilities to drive along the trajectory. Thus, it autonomously generates *routemarks* that it can use when it has to follow this particular path.

The chapter has been structured into three major sections:

1. The first section has presented the general part of the navigation method. This part has been independent of the kinematics of the utilized autonomous mobile system. In this part, it has been described how the images are selected that are used to represent the trained trajectory. Besides, it has been discussed how single trajectories can be combined to a topological map. This map consists only of image sequences and named positions. Nevertheless, it contains all information a robot needs to move from any starting place to any other place stored in the map.

2. In the second section, the implementation of the navigation method on a simulated, holonomic mobile robot has been presented. Thereby, the navigation method has been enhanced by obstacle avoidance. For evaluation, 42 trajectories were trained in a simulated flat. On average, each trajectory was represented by only 1.8 images. During the autonomous repetition of the learned paths by the robot, the largest deviation between the trained trajectories and the reproduced ones was less than 10 cm. In conjunction with obstacle avoidance, the accuracy was sufficient to enable the robot to navigate successfully in the whole scene.

3. In the last section, another implementation of the navigation method has been presented, this time on a real, kinematically restricted wheelchair. The realization was tested by the training of three trajectories in two different surroundings. In a nearly quadratic room, the mean error in the calculation of the parameters important for the navigation was approximately 5°. In a long narrow corridor, it was larger by factor of three. Nevertheless, the wheelchair was able to follow all trained trajectories with a maximum positioning error of 31 cm.

The experiments performed have shown that the navigation method introduced is applicable—at least is indoor environments. In the presented form, the method requires a constant and uniform illumination. These limitations are due to the used sensor that takes images with high noise and is not able to cope with large contrasts in the surroundings. While the high noise in the taken images can be reduced by utilizing a more powerful camera, the problems with large illumination differences can only be solved with a locally adaptive optical sensor. If such a sensor was available, the definition of the weight vectors in section 2.2.1 could be changed to one that can cope with changing illuminations.

Chapter 4

Self-Localization with Panama

In the previous chapter, a navigation approach has been presented that is based on the image processing algorithm *Panama*. The method is a complete solution for certain navigation tasks. In contrast to this, it is possible to integrate *Panama* into existing navigation methods as a means for a metrical self-localization. This approach as well as its suitability for applications in practice will be examined in the current chapter.

The approach of using *Panama* for a metrical self-localization is straightforward: *Panama* determines the rotation and the bearing direction from one position to another on the basis of panoramic images that have been taken at these two locations. Therefore, it is possible to take images at predefined positions and to calculate the current position by a triangulation between the bearing directions. To realize this, on the one hand, the metrical coordinates of the reference positions have to be known and on the other hand, a panoramic image must have been taken at each of these points of reference to allow *Panama* to compare it with the current camera image. The advantage of this solution in contrast to many other approaches for metrical self-localization, e.g. the approach of Pritschow *et al.* (1994), is that it is not necessary to place any artificial landmarks in the environment that enable a specialized sensor to detected them.

In the method presented here, every point of reference R_i consists of the coordinates x_i and y_i, the rotation ζ_i, and a panoramic picture I_i that was taken at this position:

$$R_i = \begin{pmatrix} c_i \\ \zeta_i \\ I_i \end{pmatrix} \quad \text{where} \quad c_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix} \tag{4.1}$$

4.1 Determination of the x/y-Position

If *Panama* is applied to the current camera image $I_{current}$ and one of the stored reference pictures I_i, it calculates a rotation ω_i and a bearing direction δ_i. With the help of a single reference image, the current rotation of the camera $\xi_{current}$ can already be determined. This rotation corresponds to the rotation against the point of reference that is calculated by *Panama* plus the rotation of the reference position itself:

$$\xi_{current} = \xi_a = \omega_a \hat{+} \zeta_a \tag{4.2}$$

In section 2.3.3 beginning on page 35, it has been defined that *Panama* determines the bearing direction relatively to the camera's system of coordinates. Therefore, it is necessary to transform the bearing directions δ_i into the global system of coordinates before they can be used for triangulation. The rotation of the camera has already been determined in equation (4.2). Hence, the bearing directions can be transformed into global coordinates by turning the δ_i around the value of the corresponding ξ_i. The operator $\hat{+}$ has been defined in equation (3.17) on page 76:

$$\delta_i' = \delta_i \hat{+} \xi_i \tag{4.3}$$

For the determination of the camera's position, two points of reference R_a and R_b are required. The current location of the camera $\rho_{current}$ can be calculated from their coordinates and their bearings from the position of the camera:

$$\begin{aligned} \rho_{current} = \rho_{a,b} = c_b + b \begin{pmatrix} u_b \\ v_b \end{pmatrix} & \\ \text{where} \quad u_a &= \cos(\delta_a') \\ v_a &= \sin(\delta_a') \\ u_b &= \cos(\delta_b') \\ v_a &= \sin(\delta_b') \\ b &= \frac{y_b - y_a + v_a \frac{x_a - x_b}{u_a}}{v_a \frac{u_b}{u_a} - v_b} \end{aligned} \tag{4.4}$$

Two problems arise from the application of equation (4.4):

1. With this equation, it is only possible to determine positions that do not lie on the common straight line of the two points of reference (cf. figure 4.1). To demonstrate this fact, an experiment was conducted in a simulated room. The room was already used in section 2.5 beginning on page 44 and is again shown in figure 4.2. It has a size of 7×7 m^2. All positions of the camera that were determined have lain in the inner $\frac{1}{9}$ of the room. This inner square was approximately 2.8×2.8 m^2 in size. The

two reference positions were in opposite corners of the inner area. The results of this experiment are shown in figure 4.3. It is obvious that the error in the determination of the x/y-position was not only large on the connecting line between the two points of reference but in addition, it was also large close to this straight line. The precision of the determination of the position depends on the angle between the two bearing directions δ'_a and δ'_b, i.e., it depends on the angle $\delta''_{a,b}$:

$$\delta''_{a,b} = \delta'_a \hat{-} \delta'_b \tag{4.5}$$

The self-localization is imprecise if either $\delta''_{a,b}$ is very large or very small, i.e. if it heavily differs from $\pm\frac{\pi}{2}$. In these ranges, small errors in the determination of the angles δ'_a and δ'_b cause large differences in the calculated position $\rho_{current}$, and therefore they generate large errors.

2. In section 3.3.5 beginning on page 97, it has been shown that *Panama* determines the bearing direction δ only imprecisely if the two positions where the panoramic images have been taken are located very close to each other. Therefore, the calculated position is incorrect in the vicinity of each of the points of reference. For the results presented in figure 4.3, this is not important because the positions close to the points

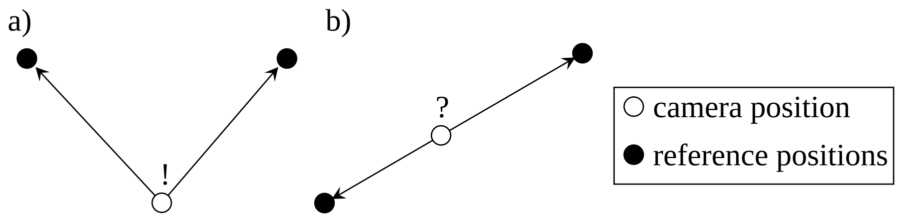

Figure 4.1: Illustration of the limitations of a determination of the position that is only based on two points of reference. a) A unique camera position can be determined. b) No unique position can be calculated.

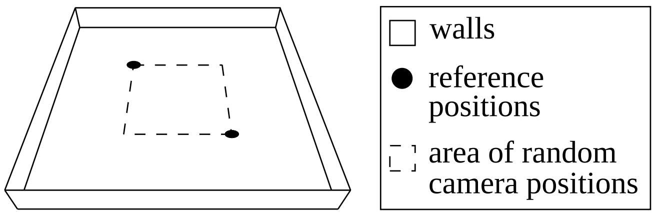

Figure 4.2: Illustration of the simulated test environment with two reference positions.

of reference are close to the connecting line between these points as well. Hence, the self-localization is faulty at these locations anyway.

To get around these weaknesses, it is necessary to use three points of reference instead of only two. Then, the determination of the camera's position is performed using the two points of reference that are farthest away from the camera. If the reference positions are far enough apart and if they are uniformly distributed, this avoids small distances between the camera and the two points of reference used for triangulation as well as between the camera and the connecting lines of these points. For clarification, figure 4.4 shows two examples for this selection process.

As it is the task of the presented method to determine the position of the camera, its proximity to the reference positions is unknown before the self-localization. Therefore, the two sources of information *rotation* and *bearing direction* are insufficient to select, which of the three points of reference is closest to the camera's position. Nevertheless, the proximity to the reference positions can be rated by the different sizes of the translational optical flow fields between the camera image and each of the corresponding reference images. This size of the translational flow is calculated by the image processing method

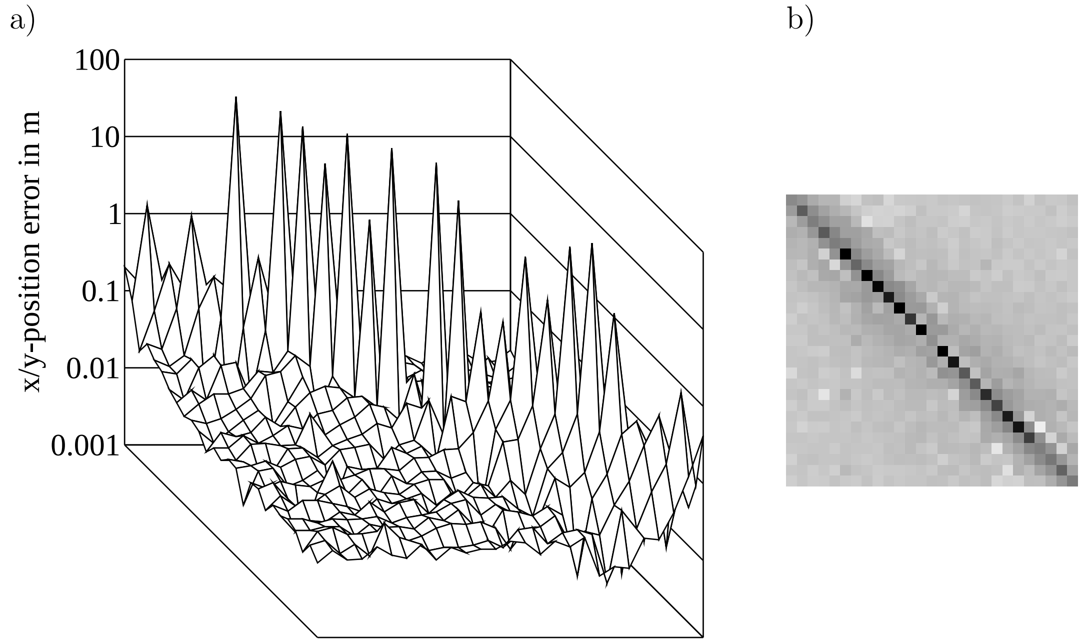

Figure 4.3: The results of the calculation of the position with two points of reference. The experiments have been performed in the scene that is shown in figure 4.2. a) The errors in a three-dimensional plot. b) The errors in a gray-scale plot (darker areas visualize larger errors).

Panama, too. The calculation of this value φ has already been described in section 2.3.2 on page 35. It can be used to select the two out of three bearing directions that point to the reference positions that are further away than the third one. To enable the algorithm to perform this selection, it is necessary to apply *Panama* to the current camera image and every reference image, i.e., it has to be utilized three times. Then, equation (4.4) can be used with the two chosen points of reference to determine the location of the camera:

$$\rho_{current} = \begin{cases} \rho_{a,b} & \text{if } \varphi_c < \varphi_a \wedge \varphi_c < \varphi_b \\ \rho_{a,c} & \text{if } \varphi_b < \varphi_a \wedge \varphi_b < \varphi_c \\ \rho_{b,c} & \text{if } \varphi_a < \varphi_b \wedge \varphi_a < \varphi_c \end{cases} \tag{4.6}$$

For comparison, the same experiment was repeated with three reference positions in the room that is depicted in figure 4.5a. The results plotted in figure 4.6 show that the error in calculating the x/y-position was uniformly distributed over the whole experimental area in this setup. The error has only been approximately 1 cm, i.e., it is as large as

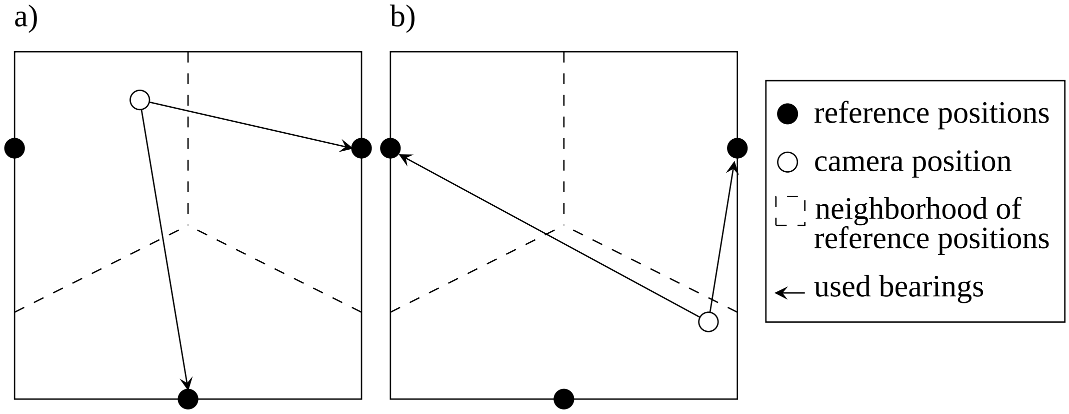

Figure 4.4: Illustration of the selection of two out of three points of reference for the determination of the x/y-position. a+b) Two examples.

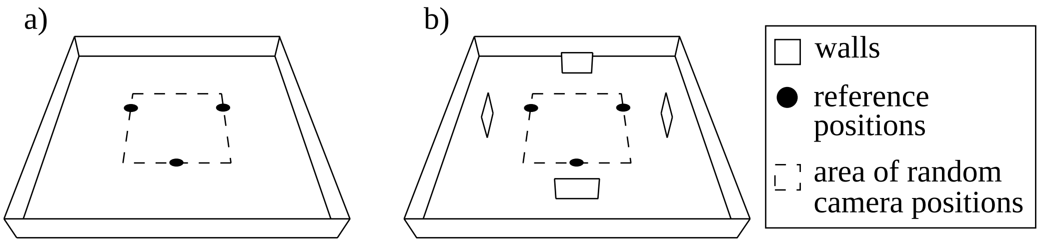

Figure 4.5: Illustration of the simulated test environment with three reference positions. a) Without obstacles. b) With four obstacles.

the smallest errors in the first experiment shown in figure 4.3. Therefore, the use of three reference positions significantly improves the precision of the metrical self-localization.

4.2 Calculation of the Rotation

As *Panama* has to be applied to the current camera image and three reference images, the question is how the three rotations ω_i determined by these three calculations should be integrated into the single rotation of the camera $\xi_{current}$. Three approaches have been taken from a variety of possibilities:

1. The first variant is to determine the average of the three calculated rotations and to use it as the rotation of the camera $\xi_{current}$:

$$\xi_{current} = \frac{\xi_a + \xi_b + \xi_c}{3} \tag{4.7}$$

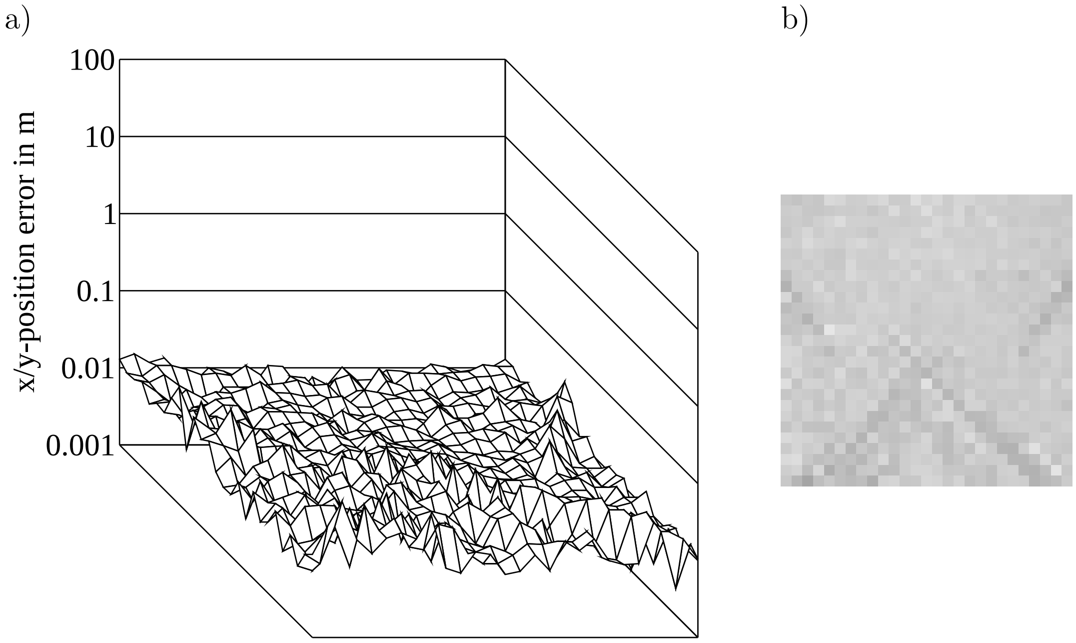

Figure 4.6: The results of the calculation of the position with three points of reference. The experiments have been performed in the scene that is shown in figure 4.5a. a) The errors in a three-dimensional plot. b) The errors in a gray-scale plot (darker areas visualize larger errors).

This approach is motivated by the assumption that the possible errors in the determination of a single ω_i can be compensated by the corresponding other two rotations.

2. As an alternative, one of the three rotations can be used as the camera's orientation. For example, the rotation to the closest point of reference can be selected, i.e., the rotation to the position with the smallest translational flow field to the camera image:

$$\xi_{current} = \begin{cases} \xi_a & \text{if } \varphi_a < \varphi_b \wedge \varphi_a < \varphi_c \\ \xi_b & \text{if } \varphi_b < \varphi_a \wedge \varphi_b < \varphi_c \\ \xi_c & \text{if } \varphi_c < \varphi_a \wedge \varphi_c < \varphi_b \end{cases} \tag{4.8}$$

   This approach is based on the supposition that larger translations impede the determination of the rotation, and therefore they should be avoided in the calculation of the camera's rotation.

3. In contrast to the previous solution, the rotation to the reference position can be used that has the largest distance to the camera's location and therefore the largest translational flow field to the camera image:

$$\xi_{current} = \begin{cases} \xi_c & \text{if } \varphi_a > \varphi_b \wedge \varphi_a > \varphi_c \\ \xi_b & \text{if } \varphi_b > \varphi_a \wedge \varphi_b > \varphi_c \\ \xi_c & \text{if } \varphi_c > \varphi_a \wedge \varphi_c > \varphi_b \end{cases} \tag{4.9}$$

4.3 Experiments and Results

Analogously to the analysis of the *Panama*-algorithm conducted in section 2.5, the performance of the approach presented in this chapter was examined. As in section 2.5, the experiments were conducted in two different simulated rooms. These rooms were identical to the rooms used in section 2.5 but in addition, three points of reference were placed into them. The two experimental environments are shown in figure 4.5.

The precision of the metrical self-localization was analyzed with different ratios of noise in the taken images. Altogether, 15 different levels of noise from 2^{-14} to 2^0 in relation to the pixel's possible maximum intensity were examined. The utilized model of noise has already been discussed in section 2.5. For each level of noise, 100 positions were determined. The maximum error as well as the average error in the calculation of the x/y-position were recorded. In addition, for each of the three solutions for the calculation of the rotation, the maximum and average errors were registered, too. The results of the experiments performed in the simulated room A are depicted in figure 4.7. The outcomes of the tests in room B are shown in figure 4.8.

The results of the experiments in the two environments differed only slightly. The errors in the room B always were larger than in room A but the differences were very small. Hence, it is not necessary to differentiate between the two scenarios when discussing the results.

The experiments clearly showed that the second variant for the determination of the rotation is superior. The rotation is calculated more precisely if the translational flow field is small. This property was not yet visible in the results of the studies presented in section 3.3.5 beginning on page 97. The determination of the rotation by calculating the average only was the second best solution. Consequently, the use of the rotation against the furthest point of reference was the most unfavourable. Although the noise in the images impeded the calculation of the camera's orientation, the rotation is not a limiting factor in the method's applicability to real world tasks. Instead, the calculation of the x/y-position defines the breakpoint, where the precision falls below a level that prevents the method from being used for a particular task.

With increasing noise ratio, the errors in the determination of the x/y-position increase steadily. The average error in the x/y-self-localization remained below 20 cm up to a noise

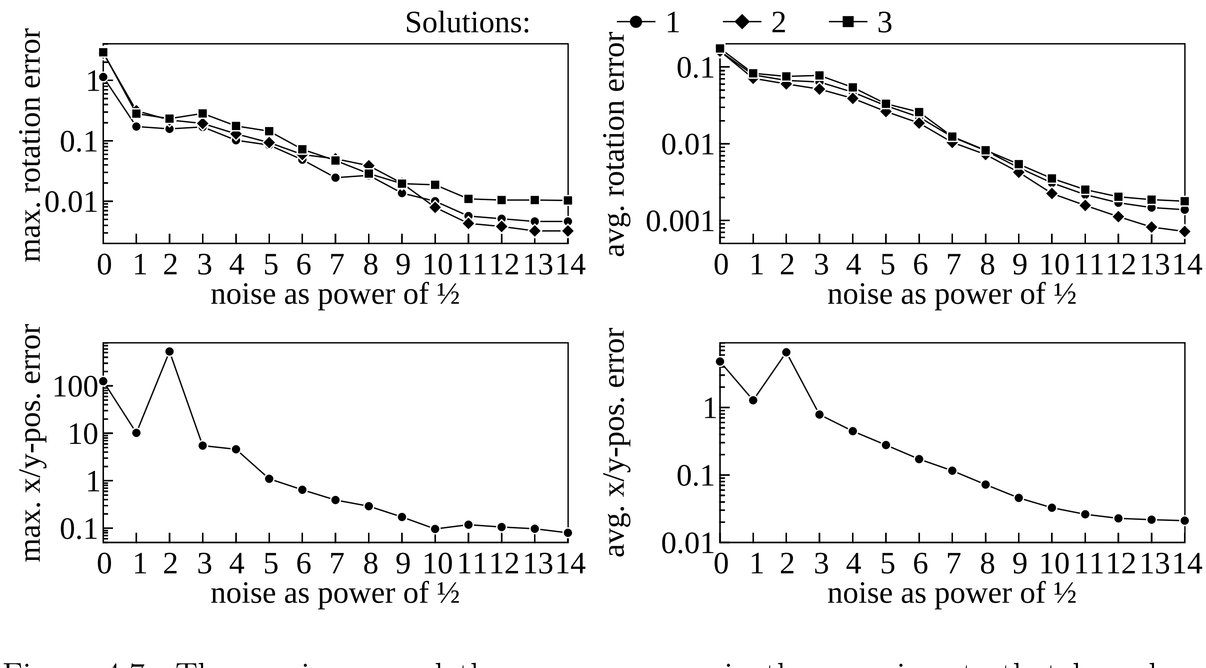

Figure 4.7: The maximum and the average errors in the experiments that have been conducted in the simulated room A that is shown in figure 4.5a. The rotation errors are shown in radians. The errors in the calculation of the x/y-position are plotted in m.

ratio of 2^{-6}, i.e. below 5% of the length of the experimental area's diagonal whereas the maximum error already exceeded this limit at a noise level of 2^{-8} (with one exception in room B). For most applications, the use of *Panama* for metrical self-localization will only be possible if the noise ratio of the taken images remains below 2^{-7}, i.e. approximately 1%. In both experimental environments, the average error was around 10 cm and the maximum error was around 40 cm at this noise level.

4.4 Summary

In this chapter, an approach for a metrical self-localization has been presented that is based on the image processing method *Panama*. As the only preparation of a room for a determination of the camera's position, it is necessary to take a panoramic image at each of three metrically surveyed positions. Then, the bearings of these points of reference can be taken from an arbitrary camera location in the same room. Out of these three bearing directions, those two are selected that point to the farthest reference positions. From these two directions, the current position of the camera is calculated by triangulation.

The simulated experiments showed that the mean error in the calculation of the position was 2.5% of the experimental area's diameter at a noise ratio of 2^{-7}. The maximum

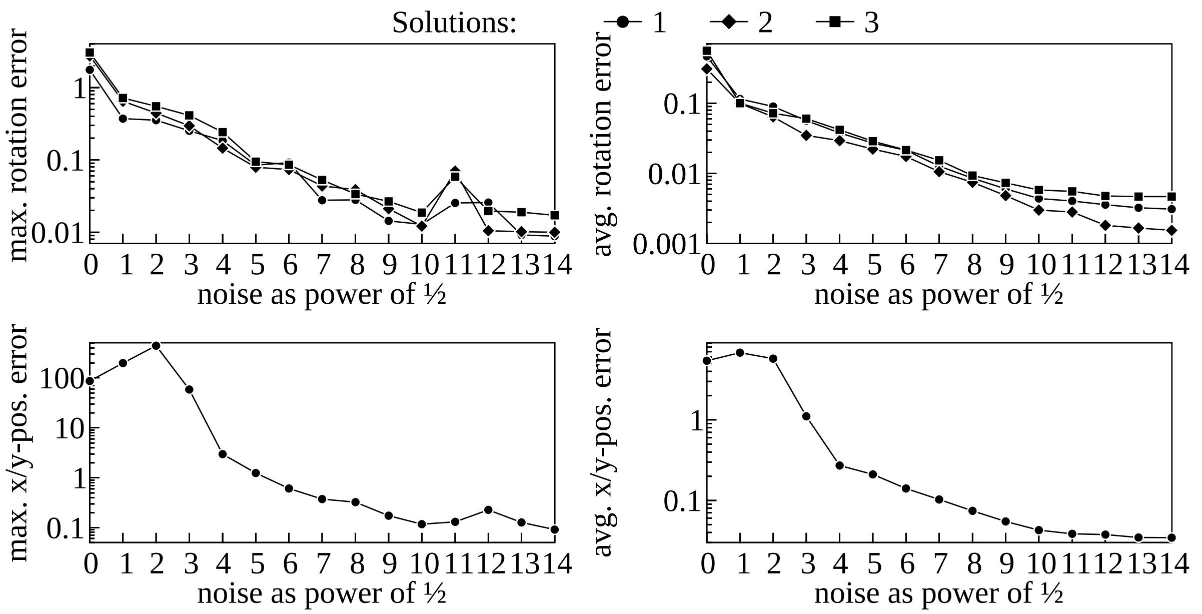

Figure 4.8: The maximum and the average errors in the experiments that have been conducted in the simulated room B that is shown in figure 4.5b. The rotation errors are shown in radians. The errors in the calculation of the x/y-position are plotted in m.

error was 10% of this length. The errors in the determination of the rotation were ignorable because the maximum error at this noise ratio was less than 1.5°.

Chapter 5

Navigation with Artificial Routemarks

In this chapter, another navigation approach will be presented that has been developed in the context of the priority program "Spatial Cognition". This program is funded by

Figure 5.1: The Wheelchair.

the Deutsche Forschungsgemeinschaft and facilitates inter-disciplinary collaboration on the acquisition and representation of spatial knowledge. In contrast to the work that has been presented in this thesis so far, the navigation approach that will be introduced in this chapter is psychologically motivated. It follows the *Taxonomy of Navigation* (cf. section 1.1.3) that has been discussed with the psychological working groups in the priority program. The navigation approach implemented is a route-following tactic based on several *basic behaviors* and the detection of *routemark constellations* that function as *location designators* (Krieg-Brückner *et al.* 1998). It has been realized on the wheelchair that was also used for the experiments described in section 3.3.

This chapter will first present the three *sensor control subsystems* that—together with a *local obstacle map*—are the foundation for the implementation of the *basic behaviors*. Then, as the second basis for the presented navigation approach, a simple method for the detection of artificial routemarks and their combination to *routemark constellations* will be introduced. Afterwards, the route-following approach will be presented, followed by a discussion about possible errors that may occur during the autonomous following of a route. A backtracking method will be proposed as a method to recover from these errors.

5.1 Sensor Control Subsystems

The sensory equipment of the wheelchair has already been presented in section 3.3.1. In the work presented in this chapter, the sensors are assigned to four control subsystems. Three of these systems are illustrated in figure 5.2:

Collision Detection. The wheelchair uses all tactile sensors, all infrared sensors and the wide-angle ultrasonic sensors to detect collisions with the environment. As long as an obstacle is perceived by the infrared or ultrasonic sensors, the wheelchair is able to stop before physical contact is made.

Steering Restriction. As the wheelchair is steering with its back wheels, its rear swings out very heavily. To prevent it from colliding with obstacles on the side during driving maneuvers, the distance to the closest obstacle is measured and the steering angle is reduced as much as necessary to avoid a collision.

Navigation. Six of the narrow-angle ultrasonic sensors and the six infrared sensors are employed for navigation purposes. They have been chosen because their measurements do not only reflect a certain distance to an obstacle but also determine it in a definite direction. In contrast, the wide-angle ultrasonic sensors would not allow a precise localization. Nevertheless, two wide-angle sensors to the side support the navigation subsystem at the moment.

Landmark or Routemark Detection. In contrast to the approach presented in section 3.3, the camera is used to take normal flat images. It is mounted on a pan-tilt-head. Therefore, it can watch the environment independently from the current orientation of the wheelchair. This allows the camera to scan the surroundings for the routemarks required for the route-following navigation.

5.2 Local Obstacle Map

Ultrasonic sensors have several weaknesses (Lee 1996, Jörg *et al.* 1993). The signal that has been sent out by a sensor may not return to the same sensor if it hits a smooth surface diagonally (specularity) or is caught by another sensor (cross-talk). As the sensors of the wheelchair's navigation subsystem do not seem to produce cross-talks at all in their current arrangement but often miss smooth objects, it is not possible to implement the basic navigation by a straight sensorimotor-linkage. Instead, the measured distances are recorded in a local obstacle map (cf. figure 5.3). This map plots the local environment around the wheelchair and represents an area of 4×4 m^2. When the wheelchair drives on, the measurements in the map are shifted in the same way as the environment passes along the moving wheelchair. Everything that is scrolled out of the map is forgotten, as are

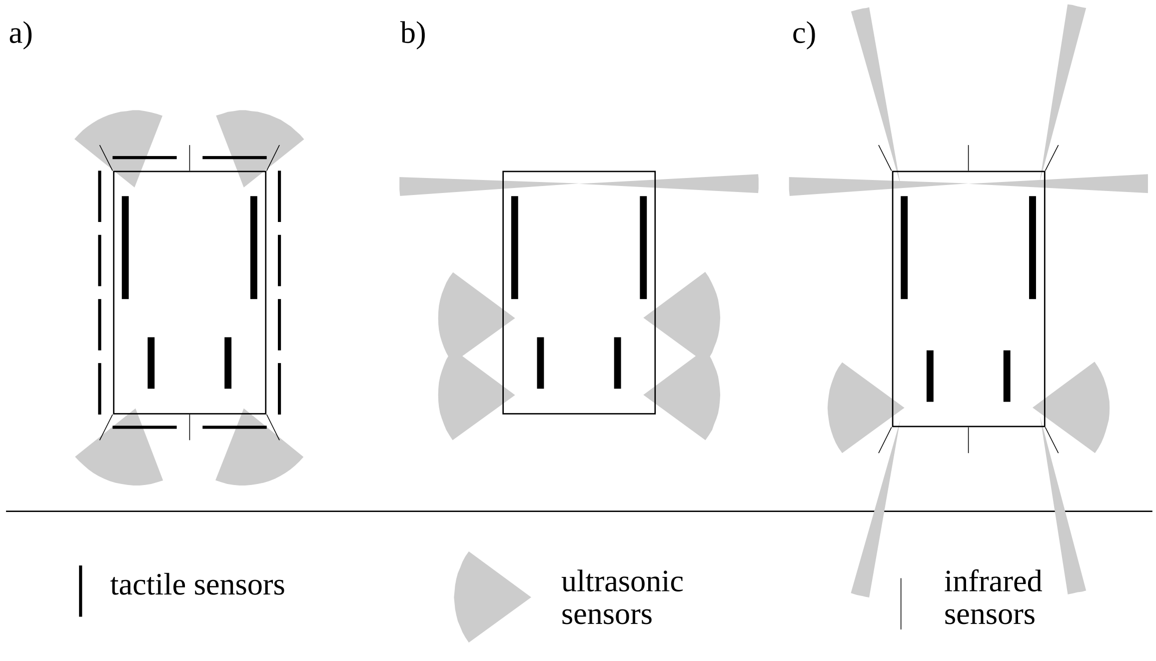

Figure 5.2: The sensor control subsystems. a) Collision detection. b) Steering restriction. c) Navigation.

measurements that are older than 30 seconds. This allows the wheelchair to cope with dynamic obstacles. Thus the local obstacle map corresponds to a short term memory. In contrast to approaches of Matthies and Elfes (1988), Elfes (1991), and Burgard *et al.* (1997), the obstacle map is only used for local navigation maneuvers. Hence, the measurements are simply stored in the map and no probabilistic sensor model is used.

The information in the obstacle map is abstracted by four *virtual sensors* in the map; their measurements are used for navigation. The virtual sensors work like ultrasonic sensors: they determine the distance to the closest obstacle in their measuring range. In contrast to real sensors, the virtual ones only measure distances to objects already represented in the obstacle map. Thus, they can exploit the sensory data collected in the past 30 seconds; they may even detect an obstacle that the real sensors currently overlook. Two virtual sensors in front scan the map from a position that, in reality, is 10 cm in front of the wheelchair; the other two scan the map 10 cm behind the vehicle. In each pair, one is oriented towards the left side; the other towards the right side (cf. figure 5.3).

5.3 Basic Behaviors

Several basic behaviors, e.g., wall-following and turning-into-door, form the basis of the presented navigation method. They enable the wheelchair to move in corridors, and to enter and exit rooms. They are fairly robust against changes in the environment because

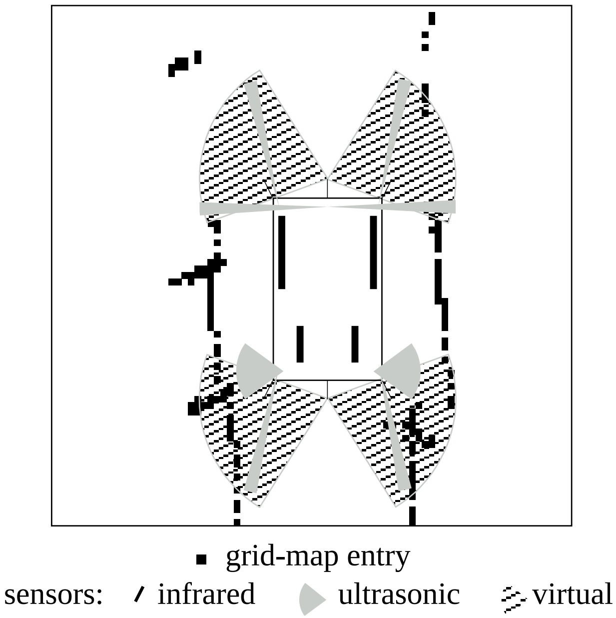

Figure 5.3: The local obstacle map.

they hardly ever assume fixed compositions of the surroundings. Their implementation is based on the aforementioned virtual sensors.

Eight basic behaviors have been implemented (cf. figure 5.4):

5.3.1 Center between Walls

If this behavior is selected, the wheelchair will try to measure the same distance with both of the front virtual sensors. To achieve this, it always steers in the direction of the larger measurement. First, the difference between both sensor readings has to be calculated:

$$\Delta = v_{left} - v_{right} \tag{5.1}$$

Then this difference is transformed into a *steering radius*. The steering radius is the radius of the arc that describes the wheelchair's trajectory. A negative radius describes a curve to the left, whereas a positive radius represents a curve to the right. As a special case, a steering radius of zero stands for a movement straight ahead:

$$r = \begin{cases} 0 & \text{if } \Delta = 0 \\ \operatorname{sgn}(\Delta) \frac{w}{\Delta^2} & \text{otherwise} \end{cases} \tag{5.2}$$

While performing this behavior, the wheelchair is driving forwards. As it is possible that an obstacle may have been overlooked, the collision detection subsystem perceives it and stops the wheelchair. In this case, the wheelchair performs the same behavior in backward direction for a distance of 50 cm and then returns to driving forwards. Normally, the infrared sensors have detected the missed obstacle before the collision. As their measurements have been recorded in the obstacle map, the wheelchair is aware of this obstacle during further actions. To perform the same behavior while driving backwards, the difference between the two virtual sensor readings simply has to be inverted.

5.3.2 Follow Left/Right Wall

Wall-following is realized by a slightly modified wall-centering behavior. The only difference is that the measurements of the virtual sensor opposite to the wall are limited to a maximum distance of v_{max}. Therefore, the wheelchair assumes that there is another wall within a distance of v_{max} and centers itself between this virtual wall on one side and the real one on the other side. If the real corridor is narrower than this virtual one, there is no change to the wall-centering behavior. To follow a left wall, e.g., the difference between the virtual sensor readings has to be calculated as

$$\Delta_{left} = v_{left} - \min(v_{right}, v_{max}) \tag{5.3}$$

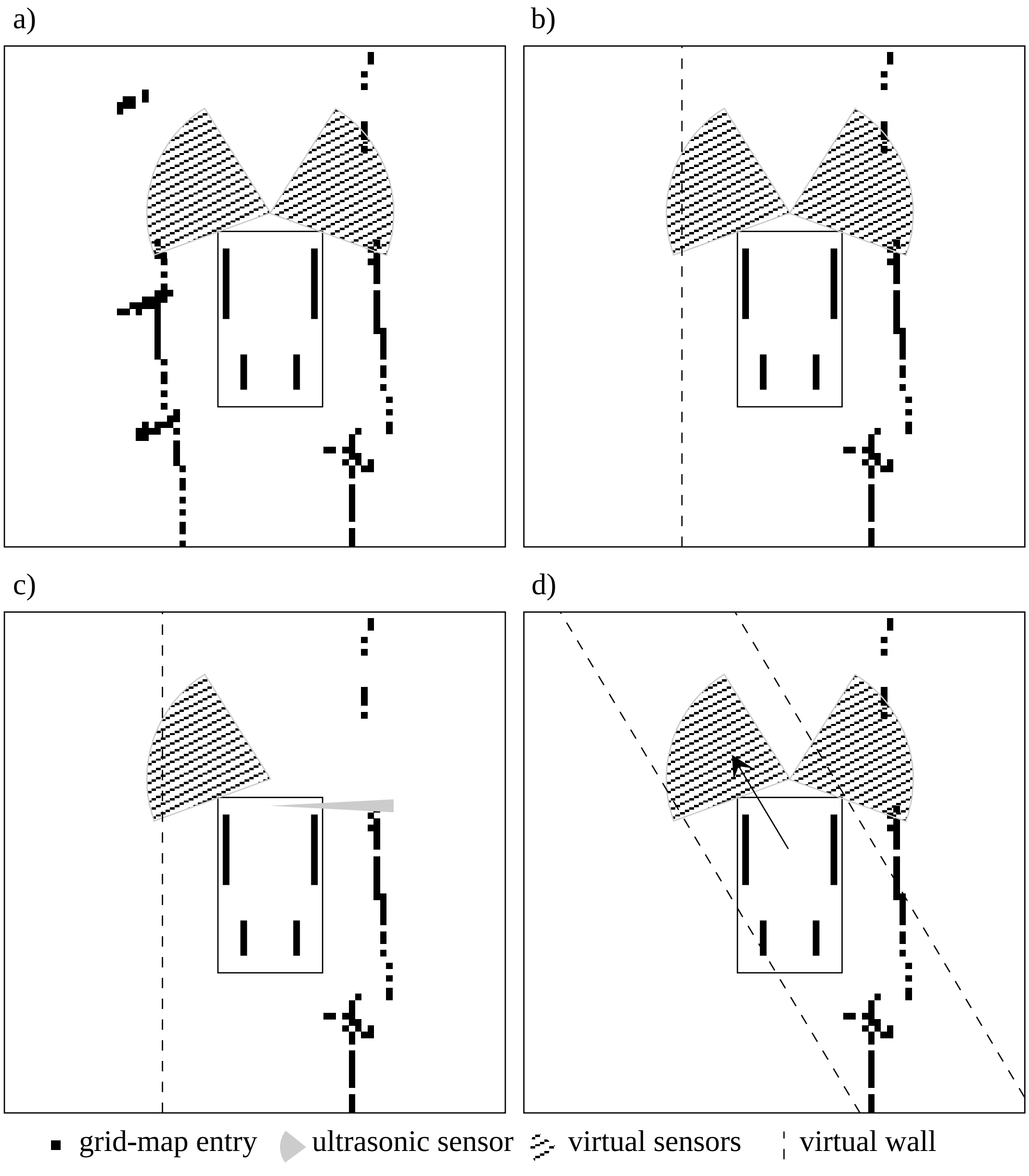

Figure 5.4: The implementation of the basic behaviors. a) Center between walls. b) Follow right wall. c) Turn into right door. d) Follow a direction while driving forwards.

5.3.3 Turn into Left/Right Door

These two behaviors enable the wheelchair to turn into a door that is either in the left or the right wall. They are quite similar to the wall-following behaviors. The only difference is that on the side of the door, no virtual sensor is used. Instead, the measurement of the real narrow-angle ultrasonic sensor is used that is oriented toward this side. In this way, the hole between the door-jambs can be determined as soon as possible. If the wheelchair has turned more than 60°, it automatically switches to the wall-centering behavior that will then drive the wheelchair through the door-frame.

5.3.4 Follow a Direction while Driving Forwards/Backwards

This behavior controls the wheelchair in a certain direction. As it is not equipped with a compass, the odometry is used as a reference system for the specification of the direction. The behavior can be performed driving forwards or driving backwards. Depending on the driving direction, either the front or the rear virtual sensors are used. The two behaviors are based on the centering between walls—as all behaviors that have been presented so far. As a result, the wheelchair is still able to avoid obstacles while it is driving in a certain direction. Similarly to wall-following, a virtual corridor is constructed from the given direction; the wheelchair centers itself in this virtual corridor. If the real corridor is narrower than this virtual one, there is no difference to the normal wall-centering behavior.

The virtual corridor is generated by calculating the difference α between the wheelchair's orientation and the desired direction. This difference is limited to a range of $[-\alpha_{max} \ldots \alpha_{max}]$. α_{max} determines how "aggressive" the wheelchair tries to follow the given direction. Then, the position in the direction of α_{max} is calculated that is on the same level as the virtual sensors. The values of the two virtual sensors are modified as if the wheelchair would drive through a corridor with walls in a certain distance s to the left and to the right of this position. In the case of driving forwards, e.g., Δ is calculated as

$$\Delta_{forward} = \min\left(v_{left}, s + o_{forward} \tan\alpha\right) - \min\left(v_{right}, s - o_{forward} \tan\alpha\right) \tag{5.4}$$

In this equation, $o_{forward}$ is the distance between the wheelchair's driving axle and the position of the virtual sensors.

5.3.5 Stop

As it is always the goal of the wheelchair to reach a certain position, it has to stop when it has arrived at the target position. Hence, stopping is also a basic behavior.

5.4 Routemark Detection

The basic behaviors enable the wheelchair to *move* in an office environment. In order to also allow it to *navigate*, it must be able to localize itself in the environment. Therefore, it has to be capable of recognizing points of reference in the surroundings, e.g. to switch to the next behavior. In the navigation approach that is presented in this chapter, these reference points are called *routemarks* because they are used to localize the wheelchair's position along a certain route.

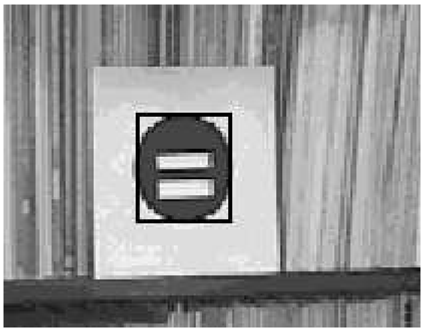

Figure 5.5: View of an artificial routemark. The square indicates recognition.

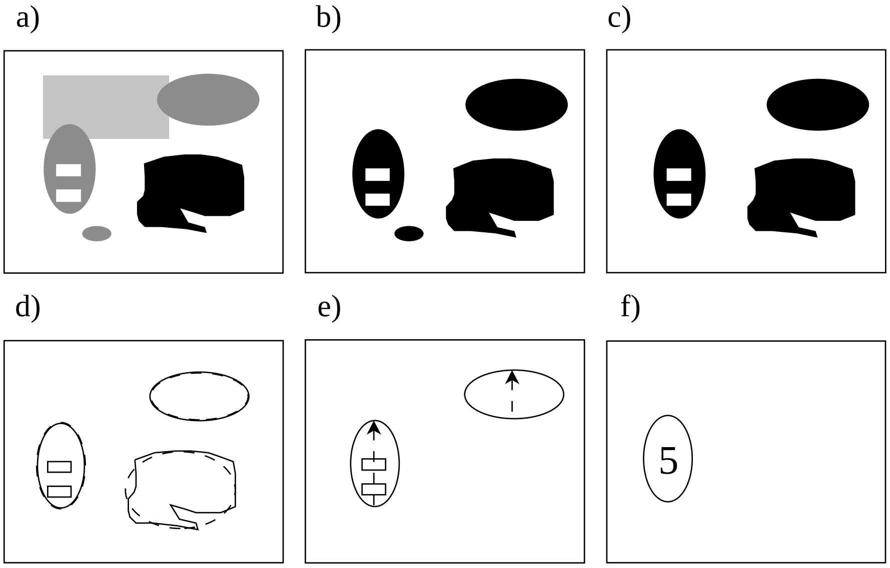

Figure 5.6: The recognition of routemarks. a) Original image. b) Extracting dark areas. c) Ignoring small areas. d) Comparing outlines. e) Extracting scan-codes. f) Resulting routemark with identification number.

At the moment, only artificial 2D-marks are employed, determined by an image processing algorithm. These marks consist of a black circle on a white background (cf. figure 5.5). In the circle, there are up to four white, horizontal stripes interpreted as a scan-code.

The recognition of the routemarks has been inspired by the work of Behrens and Kubon (1998) and is performed in four steps (cf. figure 5.6):

1. First, the image is scanned for pixels that are darker than a certain threshold φ_{max}. If such a pixel p_i has been found, a continuous region of pixels darker than 110% of p_i is determined, starting from the position of the pixel found.

2. In a second step, the width w and the height h of the region are calculated. If either the width is smaller than a predefined threshold w_{min} or the height is smaller than h_{min}, the extracted region is not used as a candidate for being a routemark.

3. Based on the values of w and h, 32 prototypical points of an ellipse with these dimensions are calculated. If the distance of at least one of these points to the region's border is larger than a predefined threshold d_{max}, the region is not assumed to be a valid mark.

4. Based on the region's height, the routemark's horizontal center is analyzed in a vertical direction to extract the embedded scan-code. Again, if there is not at least one white stripe on the mark, it is ignored. If the selected area has passed all these hurdles, it is assumed to be a valid routemark.

5.5 Route Navigation

On the level of tactical navigation (cf. section 1.1.3), the basic behaviors and the routemark recognition are combined. The environment is represented as a set of routes. A route is a static way from a starting location to a target location. The wheelchair can drive along such a route by a concatenation of different basic behaviors while routemarks are used to trigger the starting and changing of these behaviors. They are the points of reference along the route. Therefore, a route is represented as a sequence of basic behaviors and the routemarks that trigger these behaviors. This sequence can be learned by the wheelchair, e.g. when a teacher controls the vehicle along the route by switching between the available behaviors (cf. figure 5.7). Meanwhile, the camera scans the surroundings for routemarks and inserts them into the *routemark map*. When the teacher alters the wheelchair's behavior, it stores the view of the routemark map as a trigger for the new operation.

5.5.1 Routemark Map

As the sizes of the artificial routemarks are known, the distances to them can be determined from their vertical extent in the camera image. Therefore, the position of a mark relative to the wheelchair can be calculated from its distance and its bearing. This allows the creation of another map: the routemark map. This map is similar to the obstacle map that has already been presented. When the wheelchair moves, the positions of the marks in the map are adjusted on the basis of the vehicle's dead reckoning system. Similarly to the obstacle map, the routemark map is local to the wheelchair, i.e. it only maps a region of a radius of 5 m around the vehicle. All routemarks that move out of this radius are forgotten. Through the routemark map, the wheelchair knows the positions of all marks in its vicinity, even if its camera does not currently see them. Hence, the map represents a *view*, i.e. the current routemark constellation of a certain location.

5.5.2 Multiple Routemarks

When the wheelchair drives along the learned route autonomously, it executes each recorded behavior until it recognizes a part of the routemark constellation that it has stored during the training to trigger the change to the next behavior. As it cannot be assumed that the wheelchair reaches exactly the same positions during the training and the autonomous repetition, it cannot compare the stored routemark constellation with the current one directly. Instead, the intersection of the stored routemark constellation and the current state of the routemark map is determined. If at least one common routemark

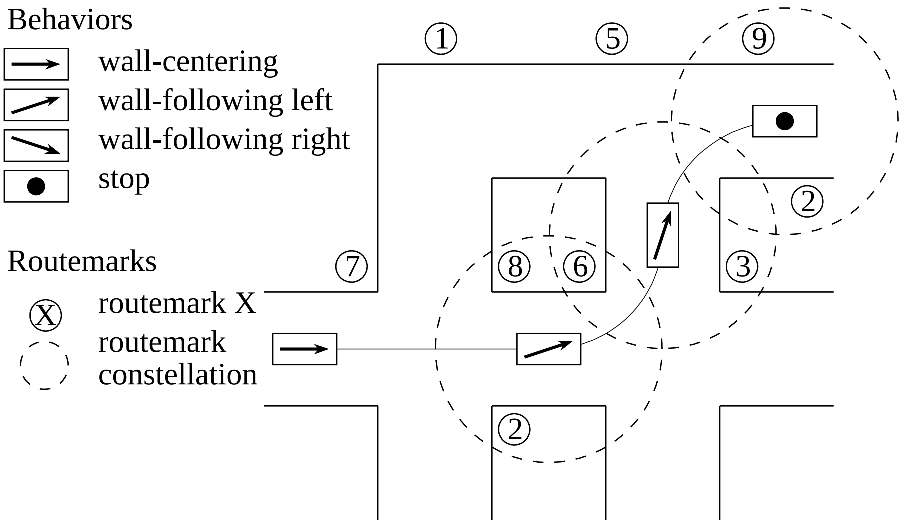

Figure 5.7: Teaching of routes.

is found, it is checked whether the wheelchair has passed all these common routemarks. It has "passed" a routemark, if the current bearing of a mark is further back than the bearing of the stored mark (cf. figure 5.8). As soon as all common routemarks have been passed, the wheelchair switches to the next behavior of the route. This solution is robust against the absence of single marks. Nevertheless, it tries to reproduce the moment of switching the behaviors as precisely as possible by using multiple marks.

As a single recognized routemark is sufficient for switching from one behavior to the next, behaviors might be switched too early if all routemarks of a certain location are stored because it is possible that the same marks have already been present earlier in this route segment. Since the decision when to switch to the next behavior is based on the bearings of the marks, it is necessary to use only those marks that have their furthest back bearing at the end of the route segment.

5.5.3 Errors

In spite of the use of multiple routemarks for the switching of behaviors, errors during the autonomous drive of the wheelchair cannot be ruled out. Two types of errors are possible (cf. figure 5.9):

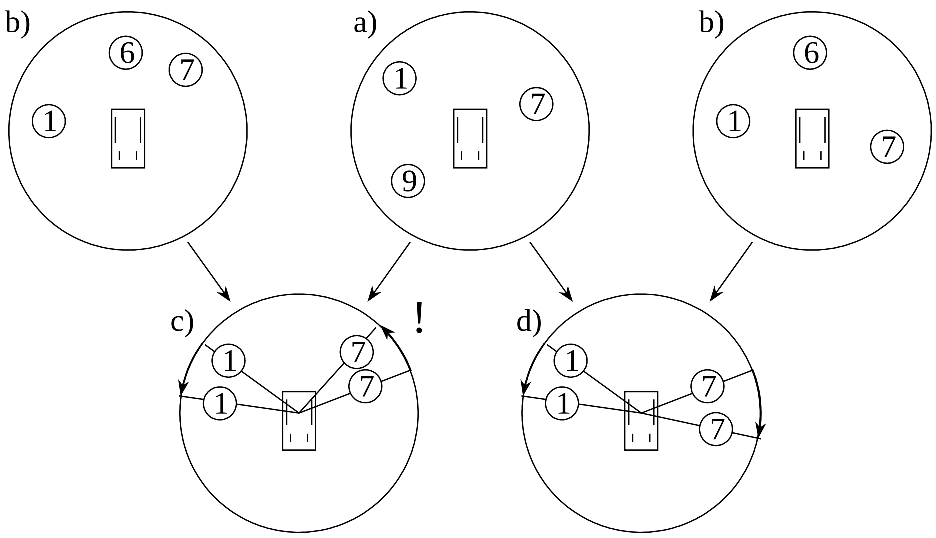

Figure 5.8: Switching of behaviors. a) Stored routemark constellation. b) Current routemark constellations. c) Some routemarks not passed. d) All routemarks passed; switching to next behavior.

No Routemarks. All routemarks for switching a behavior have been missed. This can happen, e.g. if a routemark is hidden by a person (cf. figure 5.9b).

Erroneous Behaviors. During the autonomous drive, a behavior is performed that is different from the version that has been trained. If, e.g., the wheelchair should follow the left wall and misses a turn-off because a person blocks the passage, the vehicle drives straight on where it should have turned left (cf. figure 5.9c).

In both cases, two possibilities exist to terminate the erroneous behavior:

Timeout. The execution of a behavior lasts more than 50% longer than during the training. To be able to determine this fact, the wheelchair measures the duration of behaviors. If such an overflow has been detected, it is assumed that there was an error in the reproduction of the learned route.

Premature Switching. The wheelchair finds different routemarks that match the expected constellation and switches to the next behavior. In this case, the erroneous execution is not detected immediately but it can be expected that one of the subsequent behaviors cannot be completed and therefore will fail.

So, an error can only be determined if a behavior lasts too long, but the cause of this error can also be a defective execution of a previous behavior. Thus, a strategy for compensating for errors must be able to also correct errors in previous behaviors. If the wheelchair has missed some routemarks, it can track back the route segment and search again. If it has made an error in carrying out a behavior, it has no other possibility than to drive back to a position before the error and to try again. As it does not know where the error occurred, it drives back the last route segment and searches for the missing routemarks; then it repeats this segment. If this again results in an error, it drives back the last two segments and repeats them etc. If it has repeated the complete

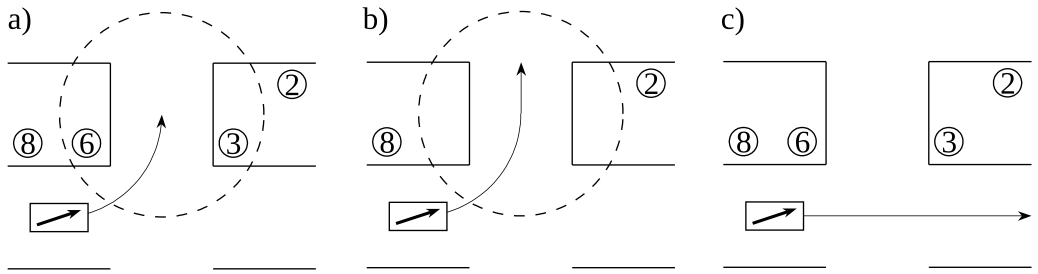

Figure 5.9: Possible errors. a) Learned route segment. b) All routemarks missing. c) Erroneous carrying out of behavior.

previous route without being successful, the execution of the route fails. Such failures can only be recovered from on a higher level in the control structure. Following the models proposed in the *Taxonomy of Navigation* (cf. section 1.1.3), this would be the level of survey knowledge. Possible solutions for a recovery from the complete failure of following a certain route may be, e.g., to also backtrack the previous route or to use an alternative.

5.5.4 Backtracking

A straightforward approach for the implementation of backtracking along a route segment would be the use of "inverse behaviors", i.e. the same behavior but performed in backward direction. The problem of such an approach is that a behavior can be carried out incorrectly in forward direction but its inverse behavior may be executed accurately. In this case, the inverse behavior would not take the wheelchair back to its starting-point, and therefore it would not compensate for the forward behavior. To really cancel a behavior, the wheelchair uses a threading tactic: it records the positions of its dead reckoning system during the execution of the behaviors and then uses the "follow a direction while driving backwards"-behavior to drive back the segment position by position. This allows it to cancel any behavior, even if the behavior was performed incorrectly. Currently, all odometry positions of a particular route are permanently stored until the same route is used again. Then, the "thread" of positions is replaced by the one recorded during the drive along this route.

5.6 Results

Although the wheelchair is 72 cm wide and 134 cm long, the basic behaviors enabled it to drive through 94 cm wide door-frames and to turn into doors in a 176 cm wide corridor (Röfer 1997b). It was able to learn and to repeat routes consisting of several basic behaviors. For example, figure 5.10 shows a 60 m long trajectory that was generated by concatenating four behaviors. The trajectory was recorded by the wheelchair's onboard odometry on the second floor of the computer science department's building. During the experiment, there were several obstacles on the floor that are not shown in the plan, e.g. tables, cupboards, and open doors, that the wheelchair bypassed.

The mobile system was able to recover from both types of possible errors, i.e. from missing routemarks and from the erroneous performance of behaviors. Both errors were forced by hiding routemarks and blocking passages. An important result for future work with natural routemarks is that it is necessary to always have a wide selection of routemarks because the worst thing that can happen is that the behavior should be changed and no appropriate routemark to trigger this change is available.

5.7 Summary

This chapter has presented the concept of a psychologically inspired navigation approach. It follows the *Taxonomy of Navigation* that has been discussed in section 1.1.3. The method realized is a route-following tactic that is based on several basic behaviors and the detection of location designators that are—in this implementation—combinations of artificial routemarks. The chapter has first described the implementation of the basic behaviors. All these behaviors are based on some kind of "center between all obstacles" approach. As a result, they all include an obstacle avoidance. Then, a method for the detection of artificial routemarks and their combination to routemark constellations has been presented. With the help of these two techniques, the route-following approach has been implemented. Routes are described as sequences of basic behaviors and routemark constellations that trigger the switching between these behaviors. Hence, the service robot, i.e. a wheelchair in the presented implementation, performs a certain behavior until it detects a particular routemark constellation, and then it continues with the successive behavior etc. until it reaches the end of the route. Possible errors during the autonomous

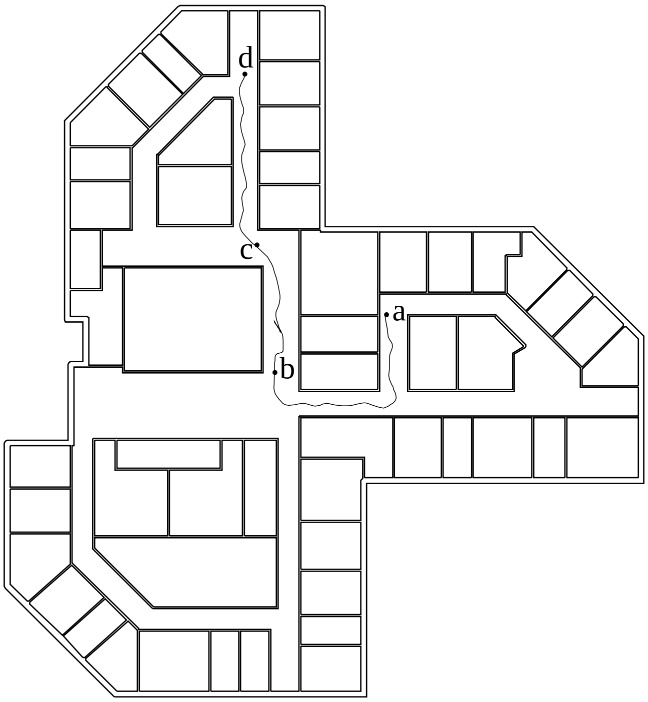

Figure 5.10: An autonomous reproduction of a learned trajectory that has been generated by the combination of four basic behaviors. a) Follow right wall. b) Follow left wall. c) Follow right wall. d) Stop.

following of routes have been identified and the possibilities to recover from these errors have been discussed. A backtracking approach that uses a threading tactic has been proposed as a suitable method for error recovery.

Chapter 6

Conclusions

6.1 Summary

This thesis has presented a novel method to determine the optical flow field from one-dimensional panoramic color images. In addition, it has shown how such a motion flow field can be used to calculate the motion parameters *rotation* and *translation direction*. The image processing approach has been embedded in a navigation method and a metrical self-localization mechanism. In addition, a method for route-based navigation has been presented that—in contrast to the other parts of this thesis—does not use panoramic pictures.

This thesis has addressed the following five topics:

Motion Field Estimation. An image processing algorithm called *Panama* that determines the motion field from one-dimensional panoramic color images has been presented. The method calculates the flow field between two images in an iterative adaptation process by finding corresponding pixels in both images. The most important characteristic of this approach is the strict *neighborhood preservation* during the matching process that is based on the constancy in the identity and the order of the depicted objects in the panoramic images. This mechanism stabilizes the matching process, and in addition, it provides for a *continuous* motion flow field, i.e. there are no crossing flow vectors.

Calculation of the Motion Parameters. From a one-dimensional flow field, two motion parameters can be determined: *rotation* and *translation direction*. Both can be in the range from $-180°$ to $180°$. As the motion field estimation calculates a continuous flow field, a new method can be applied to separate rotational flow from translational flow: by simply sorting the flow vectors, the rotational component can be found. Two methods have shown advantages in the calculation of the translation direction from the remaining translational flow field. The first one is superior at higher noise level and was also used by Hong *et al.* (1991) but only for very sparse flow fields. The other method reaches a sub-pixel precision but only under low noise conditions.

The complete algorithm consisting in flow field calculation and motion parameter determination has been called *Panama* (*Pan*oram*a* *Ma*tching).

Navigation with Panama. In contrast to other flow field approaches, e.g. by Herwig (1996), *Panama* has been used as a guidance technique instead of determining proper motion. This guidance method has been embedded in a topological route-following navigation approach. The required *route knowledge* is acquired by *teaching*. A teacher controls the agent, i.e. a robot, along a certain trajectory. The robot records a sequence of panoramic images of the surroundings, and at the end of the trajectory, it determines exactly these images of the recorded sequence that it needs to follow the trained path autonomously. Hence, a trajectory is described as a sequence of panoramic pictures. The route descriptions have been combined to a topological map. This map consists only of names that describe places and image sequences that represent the routes between these places.

The navigation approach was implemented on a simulated holonomic robot as well as on a real kinematically restricted wheelchair. Hence, it was shown that the approach is capable of solving real world navigation tasks.

Self-Localization with Panama. As second application of *Panama*, a metrical self-localization method has been presented. This approach is based on the use of panoramic pictures for self-localization, and hence it does not need, e.g., artificial landmarks for this purpose. In section 6.2, it will be shown that this method is superior to another localization technique that is also based on panoramic images.

Route Navigation. As a second navigation approach, a route-following technique has been presented that is based on the detection of artificial routemarks instead of the use of panoramic images. The approach follows the *Taxonomy of Navigation* (cf. section 1.1.3). The realized method is based on several basic behaviors and the detection of routemark constellations that function as location designators. Routes are described as sequences of basic behaviors and routemark constellations that trigger the switching between these behaviors. Hence, the robot performs a certain behavior until it detects a particular routemark constellation, and then it continues with the successive behavior etc. until it reaches the end of the route. As errors cannot be ruled out, a backtracking approach that uses a threading tactic has been proposed as suitable method for recovering from navigational errors. This method also was implemented on the kinematically restricted wheelchair.

6.2 Comparison to Related Work

6.2.1 Scene-Based Homing

Franz *et al.* (1997b) called the area of robot-homing and the use of panoramic images *scene-based homing*. There are several aspects of approaches in this field that distinguish the different methods:

- Which kind of images has been used as input.
- Which assumptions are made to be able to determine the optical flow field and determine the motion parameters.
- The method that determines the flow field.
- The approach for the determination of the motion parameters.
- Whether the robot has to stay in a constant orientation or not.
- Whether real images have been used or not.

Methods with Known or Constant Orientation. Cartwright and Collet (1983) analyzed the homing behavior of honeybees and developed five approaches to model this behavior. All these models use binary 360° snapshots. In two of the models the correspondence between two images is determined by matching edges, the other three match areas. The matching is performed by assigning each edge or area in one image to the closest edge or area in the other image. Obviously, this will only work if a few areas or edges have to be matched. The only method that works with more than three landmarks requires a known orientation. Cartwright and Collet (1983) tested their method only in simulations. In addition, they ignored the problem of converting real world images into the binary ones that are required by their method.

In the approach of Hong *et al.* (1991), correspondences between images were determined by selecting 15 characteristic points from one image and searching them in the second image. The search range in the second image was only 8° around the characteristic point's corresponding position in the first image. Hence, only small deviations between the images could be determined. The result of this correspondence search was a sparse flow field with only 15 vectors. With such few vectors, an exact estimation of the translation is impossible. Therefore, they determined several translation directions with slightly different methods and calculated the average of these directions. In addition, they expected only rotations up to three degrees. Their basic approach for the translation determination—without any averaging—has also been used in this thesis as solution 1 in section 2.3.3. As the *Panama* algorithm uses a much larger number of flow vectors, the translation direction can be determined very precisely.

In the simulations of Wittmann (1996), the deviations between stored target image and current image were determined by a correlation method in combination with a resolution pyramid. As this approach was based on the findings of Cartwright and Collet (1983), the orientation has to be known.

Methods that Determine Rotation and Translation. All the approaches discussed so far are not able to determine the rotation. In contrast, Nelson and Aloimonos (1988) analyzed the calculation of rotation *and* translation direction from spherical motion fields. They ignored the acquisition of the motion field but they showed how such a field can be employed for the determination of translation direction as well as rotation. They proposed a very straightforward method for the calculation of these parameters: they tried all parameter combinations and assessed their effects on the motion flow field. The combination with the smallest error is expected to be the correct one. They analyzed this approach under several conditions, e.g. different ratios between rotation and translation and different levels of noise. However, their results cannot be compared to the analysis of *Panama* because they induced noise into the motion flow field and not—as has been done in section 2.5—into the original images. Their straightforward approach is computationally costly: if the motion field contains n vectors and both parameters should be calculated with the same precision as the flow field's resolution, the computational complexity is $O(n^3)$. In contrast, the approach presented in this thesis only has a complexity of $O(n \log n)$ if the motion flow field is already known (cf. table 2.1 on page 43).

It is interesting to note that Nelson and Aloimonos (1988)—as any other author reviewed here—did not specify whether the motion fields used contain flow vectors larger than $\pm\pi$ which is—at least for the work presented in this thesis—an important aspect (cf. section 2.2.3).

Chahl and Srinivasan (1997) developed the *image interpolation algorithm*. In this algorithm, two panoramic images are taken at different reference positions. Then, a third one is taken between these references. The image interpolation algorithm is able to determine where the third image has been taken between the two reference positions. On a real robot, Chahl and Srinivasan (1997) have only tested either to determine translation or rotation; in a simulation, their algorithm has calculated both at once. In the real environment, all images involved in the matching process were taken at close positions. Hence, the deviations between these images were only small. In their simulation, Chahl and Srinivasan (1997) addressed the task of self-localization. This can be compared to the approach presented in chapter 4 of this thesis. Without any perturbation of the used images, they measured positional errors of up to 45% of the size of their test area. In the localization method presented in this thesis, such errors only occur at a minimum noise level of 2^{-5}. In addition, in the experiments of Chahl and Srinivasan (1997), the allowed maximum rotation was only 10°. Even under these limitations, the maximum error was 2.7°, i.e. 27% of the possible angles. Although the localization method presented in this thesis always determines rotations in the range from -180° to 180°, such large errors are only produced at a noise level of 100%.

The approach that comes closest to the performance of the image processing method that has been presented in this thesis, is the one of Franz *et al.* (1997b) inspired by preliminary work of this thesis (Röfer 1995a). In their method, it is assumed that the distances between the camera and all objects in the surroundings are the same. Based on this assumption, they determined rotation and translation direction by minimizing an error function. As they used a gradient descent method, their approach is very fast. If the same number of pixels is used, their method would run four times faster on an SGI-Indy workstation than the one presented in this thesis on the wheelchair's Pentium PC (both 100 MHz). However, as the distances to the surrounding objects are not equal in general, this method does not always calculate the correct direction to the target position. Franz *et al.* (1997b) showed that their robot would normally reach the goal position but not on a straight line. Hence, the method is suitable for navigation in free space but unsuitable for indoor environments. In contrast, the method presented in this thesis relies on a constant order of the surrounding objects that are depicted in the used images and not on equal distances. As this "constant sequence" assumption is not true for all situations either, the navigation approach based on the image processing method compensates for this weakness. As a result, the complete algorithm generates precise estimates of rotation and translation direction.

Homing with Flat 2-D Images. An approach that also belongs to the area of scene-based homing although it does not use 360° views for navigation is the *elastic template matching* of Balkenius and Kopp (1997). In their method, a robot is trained by a teacher to follow a certain trajectory. Afterwards, it is able to follow the trained path autonomously. During training, *templates* are learned as spatial arrangements of small features. These features consist of 8×8 edge-filtered pixels. During the autonomous drive, the robot tries to find the learned templates again. This is done by searching for the stored features in the learned spatial arrangement. If some of the searched features are found, their deviations are used to determine the robot's movement. The distances between the features are *elastic*, and hence a template can be found in images taken at different locations. However, the features are of constant size. So, the templates can only be matched with images in a distance of up to 60 cm from the original recording position. Nevertheless, as this approach uses a flat camera image, it can only compensate for rotations up to 15°. Balkenius and Kopp (1997) did not mention how their robot decides when to learn a new template on a trajectory but it seems that they did not use an adaptive method as the one presented in chapter 3 of this thesis.

Another homing method based on the use of flat 2-D images was developed by Basri and Rivlin (1995). They searched for large edges in images taken at different positions to acquire a model of the surroundings. Then, during homing, an attempt is made to interpolate the edges in the current image as a linear combination of the models. In certain respects, this is a 2-D variant of the *image interpolation algorithm* of Chahl and Srinivasan (1997). However, the image interpolation algorithm works on unpreprocessed images. In contrast to the method of Franz *et al.* (1997b) and the one presented in this thesis, the approach on Basri and Rivlin (1995) requires the existence of large edges in the surroundings. As all approaches are based on the use of flat 2-D images, only small rotations can be compensated for.

authors	basic assumptions	correspondence	input	const. orient.	real images
Cartwright & Collet 1983	continuous binary image areas, few areas	region matching	binary 1-D 360°	yes	no
Nelson & Aloimonos 1988		preexisting motion field	motion field 3 × 1-D 360°	no	no
Hong *et al.* 1991	characteristic points, only small deviations	feature matching	gray value 1-D 360°	yes	yes
Basri & Rivli 1995	weak perspective, large edges	linear combination of model images	gray value 2-D flat	no	yes
Wittmann 1996	constant brightness	correlation method & resolution pyramid	gray value 1-D 360°	yes	no
Balkenius & Kopp 1997	salient small image areas with high contrast	elastic template matching	gray value 2-D flat	no	yes
Chahl & Srinivasan 1997	constant brightness, large continuous image areas	linear combination of model images	gray value 1-D 360°	yes and no	yes and no
Franz *et al.* 1997b	equal distances to surroundings	parameterized disparity fields	gray value 1-D 360°	no	yes
Röfer 1997a	constant ordering of surroundings	adaptation by moving pixel regions	color value 1-D 360°	no	yes

Table 6.1: Comparison to related work.

The approaches discussed in this section used different information for the determination of the correspondence between image regions. Some used original gray values, some used edges, and Cartwright and Collet (1983) used continuous binary areas. The *Panama* algorithm presented in this thesis combines some of these methods and improves them: the *weight vectors* utilized contain original pixel information as well as derivatives and—as a new approach—color information. And—maybe together with the work of Franz *et al.* (1997b)—it is the only method that is able to determine any rotation.

Table 6.1 summarizes the different aspects of the related work.

6.2.2 Route Navigation with Error Recovery

There have been several approaches to implement a topological navigation method and all these approaches contain navigation along routes, e.g. by Nehmzow and Smithers (1991, 1992), Giralt *et al.* (1984), Schölkopf and Mallot (1995), and Franz *et al.* (1997b). However, these approaches have not been able to recover from navigational errors. The only method that combines route navigation with error recovery has been developed by Kuipers and Byun (1988). Their simulated system was able to identify distinctive places, and thus it was able to re-localize itself after it lost its way. This was implemented by Lee (1996) on a real robot. A similar re-localization technique was realized by Duckett and Nehmzow (1997), but so far, it has not been combined with route navigation. The error recovery of these approaches is based on the assumption that the lost robot will reach a position that it already knows. Hence, it is necessary that the robot has at least a topological representation of the environment. The recovery approach presented in chapter 5 requires less information because it works on the level of route navigation, i.e., following Kuipers (1996), the causal level of navigation. Hence, it does not require map knowledge. A major enhancement in robustness might be achieved if both approaches were combined.

6.3 Future Perspectives

Navigation. In the last year while working on this thesis, the author was partially employed in the priority program "Spatial Cognition". Chapter 5 has presented first results of this work. There are two possible ways how the two navigation approaches that have been discussed in chapter 3 and chapter 5 can be combined. On the one hand, the routemark constellations in the route navigation can be replaced by panoramic images, eliminating the need for the currently used artificial routemarks. On the other hand, the technique for recovering from errors can be integrated into the navigation approach that has been presented in chapter 3, because currently, the wheelchair is lost if it leaves the trained trajectory. In addition, guidance mechanisms such as *Panama* would not normally be used to control the *local* navigation behaviors. Instead, they would only take on the *global* navigation. Hence, the local navigation presented in chapter 3 could be replaced by some of the basic behaviors described in chapter 5. Together, these two enhancements would remarkably improve the reliability of the navigation method presented in chapter 3. The result would be a route navigation method that uses natural routemarks, i.e. panoramic views, and that is able to recover from navigational errors.

3-D Routemarks. In the "Spatial Cognition" project, it is planned to replace the identification of artificial routemarks by the recognition of natural 3D-marks. To achieve this, a structure from motion algorithm will be applied to the camera images. The proposed method will be based on the work of Herwig (1996). It should generate relative depth images segmented into areas of nearly equal relative motion. In each such area, discontinuities of depth, and discontinuities and extrema of depth derivatives should be recognized. Significant depth discontinuities or extrema should be automatically selected and utilized as 3-D marks. Early results in this field indicate that the algorithm is capable of estimating depth in real time, with high spatial resolution and robust against changes in illumination, orientation, and noise (Krieg-Brückner *et al.* 1998, Röfer and Müller 1998).

This will also change the way routemarks are organized. When the recognition of routemarks is not only regarded as a binary decision but instead as a process with a particular uncertainty, the representation of knowledge becomes more complex. To develop a solution for this problem, several psychological findings could be employed, e.g. expected routemarks should be detected with a higher probability than unexpected ones. Marks in the same constellation could support each other in the recognition process.

Panoramic Image Processing. Another enhancement would be the use of *Panama* for navigation and localization in 3-D space. As Nelson and Aloimonos (1988) pointed out, methods that determine motion parameters from one-dimensional 360° flow fields can directly be used to estimate motion in 3-D space. However, they ignored the question of the calculation of a spherical motion field. It is conceivable that a spherical version of *Panama* can also determine such flow fields, but the neighborhood preservation on a sphere's surface would be more complicated. The computational complexity of such an approach would remain the same as in the one-dimensional case, i.e. $O(t_{max}n)$. However, as a spherical image would have significantly more pixels than a circular one, n and t_{max} would be very large.

An essential improvement would be the use of a color sensor that takes images with lower noise ratio. This would allow to determine the weight vectors (cf. section 2.2.1) in a way that is—to a certain extent—independent of the ambient illumination. A possible approach would be to rely on hue and saturation information and only on the derivatives of the intensity. However, whether such a method would function under extreme lighting conditions, e.g. the illumination during sunsets, is unclear, but it may work under both daylight and artificial lighting conditions.

Appendix A

Architecture of the Wheelchair

This appendix will briefly describe the hardware and software architecture of the wheelchair used. It is the first prototype of the *Bremen Autonomous Wheelchair* developed in the student project SAUS partially directed by the author of this thesis. The control software of the wheelchair was implemented by the author, based on the preliminary work in the student project (Maedl *et al.* 1997, Bühlmeier *et al.* 1998). It consists of 5300 lines of C++ code, plus 16300 lines of code for the robotics simulator package used, i.e. *SimRobot* (Siems *et al.* 1994) that was also partially developed by the author and ported to the *Microsoft Windows 95* operating system.

The wheelchair and its sensory equipment have already been presented in section 3.3.1. As has also been mentioned, the vehicle is controlled by five micro-controllers and a PC. These components are connected by a CAN-BUS.

A.1 CAN-BUS Nodes

The micro-controllers are called *alpha node*, *beta node* etc. Each of these five nodes has certain functions (cf. figure A.1):

Alpha Node. This node has four different functions:

1. It controls the driving speed. For security reasons, it limits the maximum speed of the wheelchair to approximately 30 cm/s.
2. It controls the driving direction, i.e. forwards/backwards, by switching a relay on the wheelchair.
3. It determines which tactile sensors have contact to an obstacle.
4. It generates an oscillating signal that loads a capacitor. If this capacitor is not loaded, the power of the wheelchair's motor is switched off.

Beta Node. This node measures the rotations of the two front wheels of the wheelchair with a precision of 3°. This information is provided in the form of two odometer counters.

Gamma Node. This node has three different functions:

1. It controls the steering angle.
2. It determines the states of the infrared sensors.
3. It controls the pan-tilt-head the camera is mounted on.

Delta Node. This node determines the measurements of the eight wide-angle ultrasonic sensors.

Epsilon Node. This node determines the measurements of the eight narrow-angle ultrasonic sensors.

A.2 Protocol

The five nodes communicate with the PC using a sequential protocol via the CAN-BUS. The PC sends a message to the first node, waits for the answer, communicates with the next node etc. Each message contains all information that the sender can provide for the receiver. Hence, in one communication round, all information is exchanged between

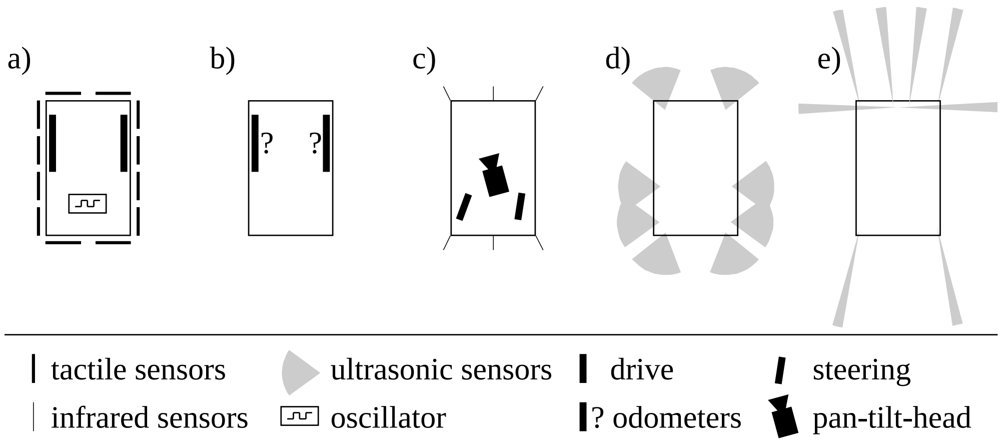

Figure A.1: The functions of the CAN-BUS nodes. a) Alpha node. b) Beta node. c) Gamma node. d) Delta node. e) Epsilon node.

the PC and the micro-controllers. On the one hand, this provides for a collision-free communication, on the other hand, the protocoll is very prone to lost messages.

A.3 Safety

The safety concept of the wheelchair is only weak. There are only a few safety techniques that have been implemented in the architecture:

- If one of the tactile sensors measures a contact, the power of the drive motor will be switched off. This is realized in an electric circuit.

- If the *alpha node* does not load the capacitor, the power of the drive motor will also be switched off. Hence, if the software on the *alpha node* does not work, the wheelchair will stop.

- The *alpha node* measures the time between the communications with the PC. If it has to wait too long for a message, it will switch off the drive motor. Hence if any message on the CAN-BUS is lost or any of the micro-controllers or the PC crashes, this timeout detection will stop the wheelchair. When the next message arrives, the wheelchair will continue to drive.

- There is a large red safety power-off switch fitted to the wheelchair. It allows to manually stop the vehicle. However, this switch causes a problem: it resets the state of the relay that determines the drive direction. Thus if the wheelchair is driving backwards when the power-off switch is pressed, it will always start driving in a forward direction after the switch has been released. As it is impossible to determine the state of the relay, nor the state of the power-off switch, this misbehavior cannot be compensated for. However, it does not influence the low level safety mechanisms described in this section.

A.4 Software on the PC

A.4.1 Operating System

As it was required to select the hardware (sensors and computers) and the software in a short time, *Microsoft Windows 95* was chosen as operating system because nearly any hardware manufacturer provides drivers for this system or at least for *Microsoft DOS* such that drivers can also be used under *Windows 95*. The second prototype of the *Bremen Autonomous Wheelchair* (Lankenau and Meyer 1997) will be based on the real-time operating system *QNX* that, on the one hand, is more suitably for robot control, but on the other hand, supports only a limited variety of hardware equipment.

A.4.2 Architecture

The software on the PC has been implemented in the object oriented language C++. The most powerful technique in object oriented programming is derivation. Thus, the software components have been structured this way. The software has been organized as a derivation hierarchy. On the lowest level, a class implements the communication with the hardware. On top of it, class by class, additional functionality is realized (cf. figure A.2). The basic idea is that higher classes can change the functionality of lower ones in order to add a special property, e.g. to implement a collision detection. Thus, derived classes can add new member functions and replace others. If other classes do not use any of its new member functions, a class can be inserted in the derivation hierarchy without changing any other class.

Windows 95 provides parallel execution of programs by a multi-threading scheduler. Thus, in one process, several program threads can be executed in a semi-parallel fashion. The wheelchair control software uses multiple threads, e.g. one each for communication with the micro-controllers. These threads are synchronized on the level of objects, i.e., each object synchronizes the access to its member variables. As the member variables are completely encapsulated, no unsynchronized access is possible. The functions that control multi-threading and synchronization have also been encapsulated in classes. These platform-specific classes must be adapted if the software should be ported to another operating system that of course must provide multi-threading, too.

The software interface uses hardware-independent units of measurement for all information. Distances are always specified in cm and angles always in radians, e.g., the pan or tilt of the camera head is specified in radians instead of using stepper motor coordinates.

A.4.3 Classes

This section presents the classes that together form the control software of the wheelchair on the PC. The classes specified here have been used to implement the route navigation presented in chapter 5. However, most of them can be used for different purposes, e.g., the classes *Wheelchair*, *Odometry*, and *StopSensor* have also been used for the experiments described in section 3.3. The class hierarchy is depicted in figure A.2.

CanBus. The class *CanBus* provides a simple interface to the communication via the CAN-BUS. Its functionality was developed in the student project SAUS. It was converted to a class by the author of this thesis.

Wheelchair. The class *Wheelchair* performs the communication with the micro-controllers, and hence with the hardware of the wheelchair. It starts an own thread that handles the communication with the micro-controllers via its base class *CanBus*. A data exchange between the PC and the micro-controllers is performed about every 150 ms. If the PC has to wait longer than 500 ms for the answer of a micro-controller, it stops the communication thread. A function provides the information which CAN-BUS node has generated this timeout. Hence, an application can warn the user. As the alpha node also determines timeouts, the wheelchair will stop in such a case.

The class *Wheelchair* provides functions for setting the wheelchair's speed, driving direction, and steering radius. It allows to read the states of the tactile, infrared, and ultrasonic sensors. In addition, it provides the states of the two odometer counters and several functions that allow to control the pan-tilt-head.

The class *Wheelchair* provides a function that is called after each message that has been sent to a micro-controller. This function can be overridden by derived classes to handle real-time events that should immediately be performed after new information has been received from a micro-controller. As there are five CAN-BUS nodes, five slots exist that can be used by derived classes to perform their calculations. They should use the slots in a way that minimizes reaction time, e.g., a wide-angle sonar-based safety stop should be executed after the new ultrasonic measurements have been received from the *delta node* but before the new driving speed is sent to the *alpha node*.

SimChair. This class links *Wheelchair* into the robotics simulator package *SimRobot* (Siems *et al.* 1994). It is derived from *SimRobot*'s *CONTROLLER* class and exists in two versions by defining or undefining the preprocessor symbol *REALITY*. If this symbol is defined, *SimChair* is derived from *Wheelchair*, if not, *Wheelchair* is simulated by *SimChair*

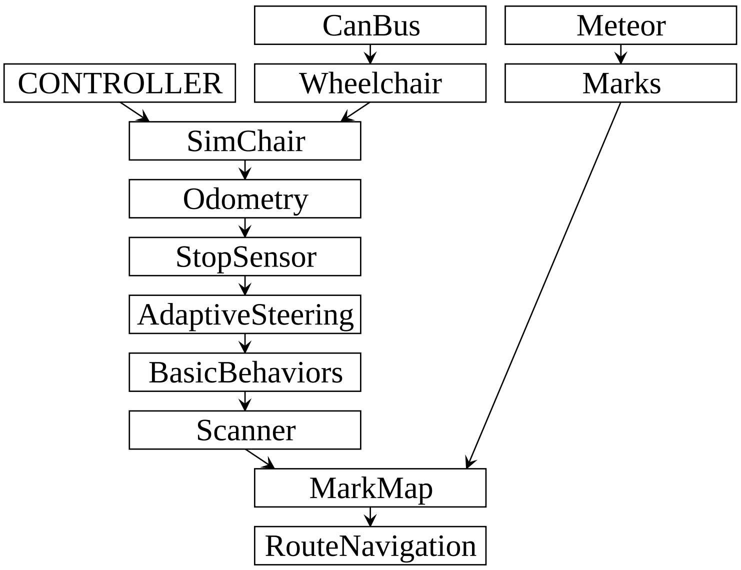

Figure A.2: The class hierarchy.

with the help of *SimRobot*. *SimChair* provides the same interface as *Wheelchair* but enhances it by some additional functions. Hence, any class that can be derived from *Wheelchair* can also be derived from *SimChair*. This allows to switch between simulation and reality by setting or removing a single preprocessor symbol. However, it requires that the wheelchair and its environment have to be correctly modeled in a *SimRobot* scene definition file. Currently, as the only exception, the pan-tilt-head is not simulated by *SimChair* although this would be possible.

As *SimChair* runs asynchronously to the control program, the simulation cannot use simulation steps with a constant duration. Instead, the duration of the previous step is measured and the following is calculated under the assumption that it will last the same period of time. Thus, the simulated wheelchair continues to move even if the control program performs lengthy operations—similar to the real wheelchair that will not wait for the program either.

As additional functionality, *SimChair* implements simple keyboard, mouse, and joystick interfaces. It also provides information about the arrangement of the sensors on the wheelchair because this information can be extracted from the simulation scene.

SimChair launches a separate thread for the robot controller. The controller itself has to be defined in a derived class. On the real wheelchair, *SimRobot*, the communication thread in *Wheelchair*, and the controller run in parallel. In the simulated version, *SimRobot* imitates the communication thread, i.e., only *SimRobot* and the controller run in parallel. The controller can use any function provided by *SimRobot*, but it has to be taken into account that *SimRobot* is not a multi-threaded program, and hence it does not provide any synchronization mechanisms.

Odometry. The class *Odometry* must at least be derived from *Wheelchair*. It enhances the interface by an odometry calculation that is based on the two odometer counters provided by *Wheelchair*. *Odometry* allows to read and change the current dead reckoning position. Together with this class, the class *Position* is introduced that describes positions in 2-D space, i.e. x/y coordinates and rotation. The class comes with a complete arithmetic for addition and subtraction of such positions, including the required coordinate transformations.

StopSensor. This class must at least be derived from *Wheelchair*. It introduces a safety stop that is based on the readings of the tactile, infrared, and wide-angle sonar sensors. This safety stop is dependent on the current driving direction and the steering radius. The class provides the information whether it has stopped the wheelchair. Single sensory categories can be excluded from this stop system. In addition, the minimum distance can be specified that the sonar sensors are allowed to measure without stopping the wheelchair.

The safety stop fails if either the sensors overlook an obstacle or if the wheelchair moves in a different direction than it is assumed to (cf. section A.3).

AdaptiveSteering. This class must at least be derived from *Odometry* and *SimChair*. It realizes an adaptation of the steering radius to avoid obstacles that are measured towards the sides of the wheelchair. It uses a local obstacle map (cf. section 5.2) with a size of 1.5×1.5 m^2 and a precision of 2 cm for this purpose.

BasicBehaviors. This class must at least be derived from *Odometry* and *SimChair*. It implements the basic behaviors that have been presented in section 5.3. It uses a local obstacle map (cf. section 5.2) with a size of 4×4 m^2 and a precision of 5 cm.

Meteor. This class is the base class for the image processing branch of the wheelchair software. It is similar to the class *Wheelchair* because it also launches a communication thread that, in this case, reads images from the framegrabber card (a *Matrox Meteor*). In this thread, it also defines a function that is called when a new image has been grabbed. Hence, all image processing is performed in parallel to the rest of the program.

Marks. This class must at least be derived from *Meteor* or another class that provides the same interface for a framegrabber card. It detects the artificial routemarks that have been described in section 5.4. The direction to these marks is determined in the camera's system of coordinates—as well as their vertical extend.

Scanner. This class must at least be derived from *Wheelchair*. It enables the pan-tilt-head to continuously turn up and down or from left to right and vice versa. The scan range can be specified.

MarkMap. This class must at least be derived from *Scanner* and *Marks*. It generates the routemark map that has been described in section 5.5.1. Thus, it converts the bearing directions of the routemarks detemined by *Marks* via the pan direction of the pan-tilt-head and the current dead reckoning position into the odometry's system of coordinates. The distances to the routemarks are determined from their vertical extend.

RouteNavigation. This class must be derived from *MarkMap*. It implements the route navigation method presented in section 5.5, provides a keyboard interface to the user, and allows to load and save route descriptions. Another class called *Route* is used to represent these routes.

A.4.4 Debugging

During the development of a program, errors cannot be ruled out (unless formal development is used). Thus, during the software-development, debuggers are used. One important property of such debuggers is that they can completely stop a program so that the programmer can inspect all variables to find the error. On a mobile system, a debugger would not normally function this way because the system would continue to drive even if the program is stopped. Two mechanisms have been introduced to stop the real as well as the simulated wheelchair when the program is halted. The real wheelchair stops because the *alpha node* will switch off the motor if the PC does not answer. The problem with the simulated wheelchair is that its movement is based on the duration of the simulation steps. If the program is stopped, the ongoing step will last very long. Thus, the next step will perform a very large movement. The reason for this behavior is that the system clock is not halted by the debugger. Hence, to overcome this problem, the time will be determined by a thread that counts in milliseconds. This is less precise than the system clock but this timer will also be halted by the debugger. As a result, the debugger can stop the real as well as the simulated wheelchair.

Appendix B

Notation

In some equations, a notation of *lists* is employed that follows notations used in functional programming languages, e.g. by Bird and Wadler (1988). Below, this notation is briefly explained.

A list containing the elements a_0 to a_{n-1} is written as $[a_0, a_1, \ldots, a_{n-1}]$ and an empty list is represented by $[]$.

Two lists can be concatenated by the $+$ operator:

$$[a_0, a_1, \ldots, a_{n-1}] + [b_0, b_1, \ldots, b_{m-1}] = [a_0, a_1, \ldots, a_{n-1}, b_0, b_1, \ldots, b_{m-1}] \tag{B.1}$$

The $\#$ operator returns the number of entries in a list:

$$\# [a_0, a_1, \ldots, a_{n-1}] = n \tag{B.2}$$

List comprehension allows to generate lists that only contain entries that satisfy a number of conditions. The list entries generated are specified on the left side of the vertical bar; the conditions are defined on the right side. The conditions can be separated into two groups: on the one hand, there are expressions that generate values, on the other hand, some predicative expressions p_a function as filters that limit the entries in the resulting list. If at least one of the generators is a list, it is assumed that the entries in the resulting list are ordered in the same way as in this generator list. In contrast, if the generator is a set, the order of the list is arbitrary.

$$[g_a \mid a \in [a_0, a_1, \ldots, a_{n-1}], \ldots, p_a, \ldots] = [\ldots, g_{a_i}, \ldots] \tag{B.3}$$

The function *hd* returns the first element of a list:

$$\mathrm{hd}_{[a_0, a_1, \ldots, a_{n-1}]} = a_0 \tag{B.4}$$

The function *tl* returns a list without its first element:

$$\mathrm{tl}_{[a_0,a_1,\ldots,a_{n-1}]} = [a_1,\ldots,a_{n-1}] \tag{B.5}$$

The function *last* returns the last element of a list:

$$\mathrm{last}_{[a_0,a_1,\ldots,a_{n-1}]} = a_{n-1} \tag{B.6}$$

The function *init* returns a list without its last element:

$$\mathrm{init}_{[a_0,\ldots,a_{n-2},a_{n-1}]} = [a_1,\ldots,a_{n-2}] \tag{B.7}$$

The function *take* returns the first m elements of a list:

$$\mathrm{take}_m^{[a_0,\ldots,a_{m-1},\ldots,a_{n-1}]} = [a_0,\ldots,a_{m-1}] \tag{B.8}$$

The function *drop* returns a list without its first m elements:

$$\mathrm{drop}_m^{[a_0,\ldots,a_m,\ldots,a_{n-1}]} = [a_m,\ldots,a_{n-1}] \tag{B.9}$$

The function *concat* concatenates a list of lists:

$$\mathrm{concat}_{[[a_0,\ldots,a_{n-1}],[b_0,,\ldots,b_{m-1}],\ldots,[c_0,\ldots,c_{l-1}]]} = [a_0,\ldots,a_{n-1},b_0,\ldots,b_{m-1},\ldots,c_0,\ldots,c_{l-1}] \tag{B.10}$$

The function *countlast* returns the number of elements at the end of a list that satisfy a certain condition:

$$\begin{aligned} countlast_c^L &= \begin{cases} 0 & \text{if } L = [] \\ 1 + countlast_c^T & \text{if } c_h \\ 0 & \text{otherwise} \end{cases} \\ &\text{where} \quad \begin{aligned} h &= last_L \\ T &= init_L \end{aligned} \end{aligned} \tag{B.11}$$

Bibliography

J. K. Aggarwal & N. Nandhakumar (1988). On the computation of motion from sequences of images – a review. In: *Proc. of the IEEE* **76**. 917–935.

M. Asade (1991). Map Building for a Mobile Robot from Sensory Data. In: S. S. Iyengar & A. Elfes (Eds.): *Autonomous Mobile Robots* **1**. IEEE Computer Society Press. Los Alamitos, California. 312–322.

Ch. Balkenius & L. Kopp (1997). Elastic Template Matching as a Basis for Visual Landmark Recognition and Spatial Navigation. In: *Spatial Reasoning in Mobile Robots and Animals*. AISB-97 Workshop. Manchester University. 15–24.

J. Bares, M. Hebert, T. Kanade, E. Krotkov, T. Mitchell, R. Simmons, & W. Whittaker (1989). Ambler – An autonomous rover for planetary exploration. In: *Computer* (6). 18–25.

T. Barkowsky, Ch. Freksa, B. Berendt, & S. Kelter (1997). Aspektkarten – Integriert räumlich-symbolische Repräsentationsstrukturen. In: C. Umbach, M. Grabski, & R. Hörnig (Eds.): *Perspektive in Sprache und Raum*. Deutscher Universitätsverlag. Wiesbaden (*to appear*).

S. T. Barnard (1991). Biological and Computational Stereo Vision. In: J.-A. Meyer & S. W. Wilson (Eds.): *Meyer J-A, Wilson S W*. MIT Press. Cambridge, Massachusetts. 127–134.

J. Barron (1984). A survey of approaches for determining optic flow, environmental layout and egomotion. In: *Tech. Report RBCV-TR-84-5*. Dept. of Computer Science. University of Toronto.

A. G. Barto, R. S. Sutton, & C. W. Anderson (1983). Neuronlike adaptive elements that can solve difficult learning control problems. In: *IEEE Transactions on Systems, Man, and Cybernetics* **13**. 834–846.

R. Basri & E. Rivlin (1995). Localization and Homing Using Combinations of Model Views. In: *Artificial Intelligence* **78**. Elsevier. 327–354.

A. Behrens & S. Kubon (1998). Objektverfolgung durch aktive Konturen und Objekterkennung. In: A. Bühlmeier, J. Kollmann, B. Krieg-Brückner, & Th. Röfer (Eds.): *Sensomotorik autonomer Systeme*. Informatik Bericht. ISSN 0722–8996. Universität Bremen (*to appear*).

K. Berns & M. Fiegert (1991). Laufmaschinen – von der Fiktion zur Realität - eine Übersicht. Studie. FZI Karlsruhe.

R. Bird & P. Wadler (1988). Introduction to Functional Programming. Series in Computer Science. Prentice Hall.

R. A. Brooks (1987). A robot that walks; emergent behaviours from a carefully evolved network. In: *A. I. Memo* **2** (1091). MIT.

R. A. Brooks (1990). Elephants Don't Play Chess. In: *Journal of Robotics and Autonomous Systems* **6** (1&2). North-Holland. 3–15.

W. Burgard, D. Fox, & D. Henning (1997). Fast Grid-Based Position Tracking for Mobile Robots. In: G. Brewka, Ch. Habel, & B. Nebel (Eds.): *KI-97: Advances in Artificial Intelligence*. Lecture Notes in Artificial Intelligence. Springer. Berlin, Heidelberg, New York. 289–300.

D. Burschka, Ch. Eberst, A. Hauck, & N. O. Stöffler (1995). Hierarchische Umgebungsmodellierung für Lokalisation, Exploration und Objektidentifikation. In: R. Dillmann, U. Rembold, & T. Lüth (Eds.): *Autonome Mobile Systeme* **11**. Informatik aktuell. Springer. Berlin, Heidelberg, New York. 132–141.

A. Bühlmeier, J. Kollmann, B. Krieg-Brückner, & Th. Röfer (Eds.) (1998). Sensomotorik autonomer Systeme. Informatik Bericht. ISSN 0722–8996. Universität Bremen (*to appear*).

B. A. Cartwright & T. S. Collet (1982). How honey bees use landmarks to guide their return to a food source. In: *Nature* **295**. 560–564.

B. A. Cartwright & T. S. Collet (1983). Landmark Learning in Bees. In: *Journal of Comparative Physiology A* **151**. 521–543.

B. A. Cartwright & T. S. Collet (1987). Landmark Maps for Honeybees. In: *Biological Cybernetics* **57**. 85–93.

J. S. Chahl & M. V. Srinivasan (1997). Navigation, Path-Planning and Homing for Autonomous Mobile Robots Using Panoramic Visual Sensors. In: *Spatial Reasoning in Mobile Robots and Animals*. AISB-97 Workshop. Manchester University. 46–55.

L. Chittka & K. Geiger (1995). Can honeybees count landmarks? In: *Animal Behavior* **49**. 159–164.

T. S. Collet & J. Baron (1994). Biological compasses and the coordinate frame of landmark memories in honeybees. In: *Nature* **368**. 137–140.

T. S. Collet & A. Kelber (1988). The retrieval of visuo-spatial memories in honeybees. In: *Journal of Comparative Physiology A* **163**. 145–150.

T. S. Collet, B. A. Cartwright, & B. A. Smith (1986). Landmarks learning and visuo-spatial memories in gerbils. In: *Journal of Comparative Physiology A* **158**. 835–851.

S. Cordes, K. Berns, & R. Dillmann (1993). Steuerungsarchitektur der sechsbeinigen Laufmaschine LAURON. In: G. Schmidt (Ed.): *Autonome Mobile Systeme* **9**. Technische Universität München. 205–213.

Ch. H. Cotter (1993). Navigation. In: *The New Grolier Multimedia Encyclopedia* **6**. Grolier Electronic Publishing, Inc.

H. Cruse, C. Bartling, G. Cymbalyuk, J. Dean, & M. Dreifert (1994). A Neural Net Controller for a Six-Legged Walking System. In: P. Gaussier & J.-D. Nicoud (Eds.): *Proc. From Perception to Action Conf.*. IEEE Computer Society Press. Los Alamitas, California. 55–65.

H. Cruse (1991). Cordination of Leg Movement in Walking Animals. In: J.-A. Meyer & S. W. Wilson (Eds.): *From animals to animats*. MIT Press. Cambridge, Massachusetts. 105–119.

U. R. Dhond & J. K. Aggarwal (1989). Structure from Stereo – A Review. In: *IEEE Transactions on Systems, Man, and Cybernetics* **19** (6). 1489–1510.

T. Duckett & U. Nehmzow (1997). Experiments in Evidence Based Localisation for a Mobile Robot. In: *Spatial Reasoning in Mobile Robots and Animals*. AISB-97 Workshop. Manchester University. 25–34.

G. Dudek, M. Jenkin, E. Milios, & D. Wilkes (1997). On Building and Navigation with a Globally Topological but Locally Geometrical Map. In: *Proc. Third ECPD International Conference on Advanced Robotics, Intelligent Automation and Active Systems*. Bremen. 132–136.

D. Dunekamp & O. Meyer (1998). Hierarchisches Belohnungslernen. In: A. Bühlmeier, J. Kollmann, B. Krieg-Brückner, & Th. Röfer (Eds.): *Sensomotorik autonomer Systeme*. Informatik Bericht. ISSN 0722–8996. Universität Bremen (*to appear*).

F. C. Dyer (1991). Bees acquire route-based memories but not cognitive maps in familiar landscape. In: *Animal Behavior* **41**. 239–246.

Th. Edlinger & G. Weiss (1995). Exploration, Navigation and Self-Localization in an Autonomous Mobile Robot. In: R. Dillmann, U. Rembold, & T. Lüth (Eds.): *Autonome Mobile Systeme* **11**. Informatik aktuell. Springer. Berlin, Heidelberg, New York. 142–151.

A. Elfes (1991). Occupancy Grids: A Stochastic Spatial Representation for Active Robot Perception. In: S. S. Iyengar & A. Elfes (Eds.): *Autonomous Mobile Robots* **1**. IEEE Computer Society Press. Los Alamitos, California. 60–70.

J. Eltze, H.-J. Weidemann, & F. Pfeiffer (1993). Sechsbeiniges technisches Laufen unter Berücksichtigung biologischer Prinzipien. In: G. Schmidt (Ed.): *Autonome Mobile Systeme* **9**. Technische Universität München. 193–204.

J. D. Foley, A. v. Dam, S. Feiner, & J. Hughes (1990). Computer Graphics – Principles and Practice. Addison-Wesley. Reading, Massachusetts.

M. O. Franz, B. Schölkopf, P. Georg, H. A. Mallot, & H. H. Bülthoff (1997a). Learning View Graphs for Robot Navigation. In: W. L. Johnson (Ed.): *Proc. 1st Int. Conf. on Autonomous Agents*. ACM Press. New York. 138–147.

M. O. Franz, B. Schölkopf, & H. H. Bülthoff (1997b). Homing by Parameterized Scene Matching. In: *Proc. 4th Europ. Conf. on Artificial Life* (*to appear*).

K. v. Frisch (1967). The dance language and orientation of bees. Oxford University Press. London.

P. Fäustle, W. Daxwanger, & G. Schmidt (1994). Steuerung lokaler Fahrmanöver durch direkte Kopplung abbildender Sensorik an ein künstliches neuronales Netz. In: P. Levi & Th. Bräunl (Eds.): *Autonome Mobile Systeme* **10**. Informatik aktuell. Springer. Berlin, Heidelberg, New York. 214–225.

C. R. Gallistel (1990). The organization of learning. MIT Press. Cambridge, Massachusetts.

J. J. Gibson, P. Olum, & F. Rosenblatt (1955). Parallax and Perspective During Aircraft Landings. In: *Am. J. Psychology* **68**. 372–385.

J. J. Gibson (1950). The Perception of the Visual World. Houghton Mifflin. Boston.

J. J. Gibson (1966). The Senses Considered as Perceptual Systems. Houghton Mifflin. Boston.

J. J. Gibson (1979). The Ecological Approach to Visual Perception. Houghton Mifflin. Boston.

G. Giralt, R. Chatila, & M. Vaisset (1984). An Integrated Navigation and Motion Control System for Autonomous Mobile Robots. In: *International Symposium on Robotics Research*. MIT Press. Cambridge, Massachusetts. 191–214.

J. L. Gould (1986). The locale map of honeybees: Do insects have a cognitive map? In: *Science* **234**. 861–863.

L. C. Graue (1963). The effect of phase shifts in the day-night cycle on pigeon homing at distances of less than one mile. In: *Ohio Journal of Science* **63**. 214–217.

E. Grosso, G. Sandini, & M. Tistarelli (1991). 3-D Object Reconstruction Using Stereo and Motion. In: S. S. Iyengar & A. Elfes (Eds.): *Autonomous Mobile Robots* **1**. IEEE Computer Society Press. Los Alamitos, California. 116–127.

J.-S. Gutmann & B. Nebel (1997). Navigation mobiler Roboter mit Laserscans. In: *Autonome Mobile Systeme* **13**. Informatik aktuell. Springer. Berlin, Heidelberg New York (*to appear*).

P. E. Hart, N. J. Nilsson, & B. Raphael (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. In: *IEEE Transactions on Systems, Man, and Cybernetics* **SSC-4** (2). Institute of Electrical and Electronics Engineers, Inc. 100–107.

M. Hebert, T. Kanade, & I. Kweon (1988). 3-D Vision Techniques for Autonomous Vehicles. In: *NSF Range Image Understanding Workshop*. 273–337.

D. J. Heeger & A. J. Jepson (1992). Subspace Methods for Recovering Rigid Motion I: Algorithm and Implementation. In: *Int. J. of Computer Vision* **7** (2). 95–117.

M. Heger (1996). Risk-Averse Reinforcement Learning. Ph. D. Thesis. In: B. Krieg-Brückner, G. Roth, & H. Schwegler (Eds.): *ZKW-Bericht* (5/96). ISSN 0947–0204. Zentrum für Kognitionswissenschaften. Universität Bremen.

H. v. Helmholtz (1909). Handbuch der Physiologischen Optik. 3rd edn. Verlag von Leopold Voss. Hamburg.

T. Herrmann, K. Schweizer, G. Janzen, & S. Katz (1997). Routen und Überblickswissen – konzeptuelle Überlegungen. In: *Arbeiten des Mannheimer Teilprojekts im DFG Schwerpunktprogramm "Raumkognition"* (1). Universität Mannheim.

Ch. Herwig (1996). Visual Motion Processing for Active Observers. Ph. D. Thesis. In: B. Krieg-Brückner, G. Roth, & H. Schwegler (Eds.): *ZKW-Bericht* (1/96). ISSN 0947–0204. Zentrum für Kognitionswissenschaften. Universität Bremen.

J. Hong, X. Tan, B. Pinette, R. Weiss, & E. M. Riseman (1991). Image-based Homing. In: *Proc. of the IEEE International Conference on Robotics and Automation*. 620–625.

H. Hoyer & R. Hoelper (1993). Intelligentes Steuerungssystem für einen omnidirektionalen Rollstuhl mit flexibel konfigurierbarer Funktionalität. In: G. Schmidt (Ed.): *Autonome Mobile Systeme* **9**. Technische Universität München. 3–16.

H. Hoyer, R. Hoelper, & U. Pabst (1994). Benutzerspezifische Wegplanung für omnidirektionale und kinematisch beschränkte Rollstühle. In: P. Levi & Th. Bräunl (Eds.): *Autonome Mobile Systeme* **10**. Informatik aktuell. Springer. Berlin, Heidelberg, New York. 274–284.

A. Hörmann & U. Rembold (1991). Development of an Advanced Robot for Autonomous Assembly. In: *IEEE Int. Conf. on Robotics and Automation*. Sacramento, California. 2452–2457.

S. Ishikawa, H. Kuwamoto, & S. Ozawa (1988). Visual Navigation of an Autonomous Vehicle Using White Line Recognition. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* **10** (5). 743–749.

O. Janda (1993). Simulation eines sensorbasierten Lokalisationssystems für automatisch geführte Fahrzeuge im Bergbau. In: G. Schmidt (Ed.): *Autonome Mobile Systeme* **9**. Technische Universität München. 385–396.

K.-W. Jörg, E. v. Puttkamer, & H.-J. Richstein (1993). Integration und Fusion heterogener Multisensorinformation zur geometrischen Weltmodellierung für einen Autonomen Mobilen Roboter. In: G. Schmidt (Ed.): *Autonome Mobile Systeme* **9**. Technische Universität München. 287–298.

B. W. Kernighan & D. M. Ritchie (1977). The C Programming Language. Prentice-Hall Inc.

A. Kirchner (1997). Mapping with Laser Rangefinder and a Grid Based Transformation Estimation. In: *Proc. Third ECPD International Conference on Advanced Robotics, Intelligent Automation and Active Systems*. Bremen. 320–325.

T. Kohonen (1982). Self-organized Formation of Topologically Correct Feature Maps. In: *Biological Cybernetics* **43**. 59–69.

J. Kollmann, A. Lankenau, A. Bühlmeier, B. Krieg-Brückner, & Th. Röfer (1997). Navigation of a Kinematically Restricted Wheelchair by the Parti-Game Algorithm. In: *Spatial Reasoning in Mobile Robots and Animals*. AISB-97 Workshop. Manchester University. 35–44.

B. Krieg-Brückner, Th. Röfer, H.-O. Carmesin, & R. Müller (1998). A Taxonomy of Spatial Knowledge and its Application to the Bremen Autonomous Wheelchair. In: Ch. Freksa, Ch. Habel, & K. F. Wender (Eds.): *Spatial Cognition*. Lecture Notes in Artificial Intelligence **1404**. Springer. Berlin, Heidelberg, New York. 373–397.

E. Krotkov, J. Bares, T. Kanade, T. Mitchell, R. Simmons, & R. Whittaker (1991). Ambler: A Six-Legged Planetary Rover. In: *Int. Conf. on Advanced Robotics*. Pisa. 717–722.

B. J. Kuipers & Y.-T. Byun (1988). A Robust, Qualitative Approach to a Spatial Learning Mobile Robot. In: *Sensor Fusion: Spatial Reasoning and Scene Interpretation* **1003**. International Society for Optical Engineering.

B. J. Kuipers & Y.-T. Byun (1991). A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. In: *Journal of Robotics and Autonomous Systems* **8**. 47–63.

B. Kuipers, R. Froom, Y. W. Lee, & D. Pierce (1993). The Semantic Hierarchy in Robot Learning. In: J. Connell & S. Mahadevan (Eds.): *Robot Learning*. Kluwer Academic Publishers. 141–170.

B. Kuipers (1996). A Hierarchy of Qualitative Representation for Space. In: *Working Papers of the Tenth International Workshop on Qulitative Reasoning (QR-96)*. Fallen Leaf Lake, California.

A. Lankenau & O. Meyer (1997). Der autonome Rollstuhl als sicheres eingebettetes System. Diplomarbeit. Universität Bremen.

S. Lanser, O. Munkelt, & Ch. Zierl (1994). Robuste videobasierte Identifizierung von Hindernissen und Werkstücken sowie die Bestimmung ihrer räumlichen Lage. In: P. Levi & Th. Bräunl (Eds.): *Autonome Mobile Systeme* **10**. Informatik aktuell. Springer. Berlin, Heidelberg, New York. 95–106.

W. Y. Lee (1996). Spatial Semantic Hierarchy for a Physical Robot. Ph. D. Thesis. University of Texas at Austin.

M. Lehrer (1994). Spatial Vision in the Honeybee: the Use of Different Cues in Different Tasks. In: *Vision Res.* **34** (18). Elsevier Science Ltd. Great Britain. 2363–2385.

B. Leonard & B. L. McNaughton (1990). Rat: conceptual, behavioral, and neurophysical perspectives. In: R. P. Kesner & D. S. Olton (Eds.): *Neurobiology of Comparitive Cognition*. Lawrence Erlbaum Associates. New York.

I. S. Lin & R. Dillmann (1995). Interaktive Umweltmodellierung für halbautonome mobile Roboter. In: R. Dillmann, U. Rembold, & T. Lüth (Eds.): *Autonome Mobile Systeme* **11**. Informatik aktuell. Springer. Berlin, Heidelberg, New York. 110–119.

L.-J. Lin (1993). Reinforcement Learning for Robots Using Neural Networks. Ph. D. Thesis. Carnegie Mellon University.

R. A. Liston & R. S. Mosher (1968). A versatile walking truck. In: *Proceedings of the Transportation Engeneering Conference*. Institution of Civil Engeneers. London.

A. Maedl, H. Tietgens, & J. Kollmann (Eds.) (1997). Abschlußbericht des studentischen Projekts SAUS. FB3 Informatik. Universität Bremen.

P. Maes (1991). A Bottom-Up Mechanism for Behavior Selection in an Artificial Creature. In: J.-A. Meyer & S. W. Wilson (Eds.): *From animals to animats*. MIT Press. Cambridge, Massachusetts. 238–246.

P. Maes (1992). Learning Behavior Networks from Experience. In: F. J. Varela & P. Bourgine (Eds.): *Toward a Practice of Autonomous Systems*. MIT Press. Cambridge, Massachusetts. 48–57.

M. J. Mataric (1991). Navigating with a Rat Brain: A Neurobiologically-Inspired Model for Robot Spatial Representation. In: J.-A. Meyer & S. W. Wilson (Eds.): *From animals to animats*. MIT Press. Cambridge, Massachusetts. 169–175.

L. Matthies & A. Elfes (1988). Integration of Sonar Data and Stereo Range Data Using a Grid-Based Representation. In: *Proc. IEEE Int. Conf. on Robotics and Automation*. 727–733.

S. Mecklenbräuker, W. Wippich, M. Wagener, & J. E. Saathoff (1997). The Interaction of Spatial Information and Actions in Mental Represention. In: Ch. Freksa, Ch. Habel, & K. F. Wender (Eds.): *Spatial Cognition*. Lecture Notes in Artificial Intelligence. Springer (*to appear*).

M. Mergel, C. Koburg, & U. Vogel (1994). Automatisches Einparken. In: P. Levi & Th. Bräunl (Eds.): *Autonome Mobile Systeme* **10**. Informatik aktuell. Springer. Berlin, Heidelberg, New York. 226–237.

M. L. Minsky (1961). Step Toward Artificial Intelligence. In: *Proc. of the Institute of Radio Engineers* **49**. 8–30.

H. Mittelstaedt & M.-L. Mittelstaedt (1982). Homing by Path Integration. In: F. Papi & H. G. Wallraff (Eds.): *Avian Navigation*. 290–297.

A. Moore & C. Atkeson (1995). The Parti-Game Algorithm for Variable Resolution Reinforcement Learning in Multidimensional State-Spaces. In: *Machine Learning* **21**. Kluwer Academic Publishers. Boston. 199–233.

A. Moore (1993). The Parti-Game Algorithm for Variable Resolution Reinforcement Learning in Multidimensional State-Spaces. In: *Neural Information Processing Systems*.

O. Munkelt, P. Levi, B. Radig, M. Rozmann, & J. Detlefsen (1993). Integration eines hochauflösenden Radarsensors in ein videobasiertes Objekterkennungssystem. In: G. Schmidt (Ed.): *Autonome Mobile Systeme* **9**. Technische Universität München. 299–312.

M. Müller & R. Wehner (1988). Path integration in desert ants, Cataglyphis fortis. In: *Proc. Natl. Acad. Sci. USA* **85**. 5287–5290.

U. Nehmzow & T. Smithers (1991). Mapbuilding Using Self-Organising Networks in "Really Useful Robots". In: J.-A. Meyer & S. W. Wilson (Eds.): *From animals to animats*. 152–159.

U. Nehmzow & T. Smithers (1992). Using Motor Actions for Location Recognition. In: J. F. Varela & P. Bourgine (Eds.): *Toward a Practice of Autonomous Mobile Systems*. MIT Press. Cambridge, Massackusetts. 96–104.

U. Nehmzow (1992). Experiments in Competence Aquisition for Autonomous Mobile Robots. Ph. D. Thesis. University of Edinburgh.

U. Nehmzow (1995). Animal and Robot Navigation. In: *Robotics and Autonomous Systems* **15** (1–2). 71–81.

R. C. Nelson & J. Aloimonos (1988). Finding Motion Parameters from Spherical Motion Fields (Or the Advantage of Having Eyes in the Back of Your Head). In: *Biological Cybernetics* **58**. Springer. 261–273.

W. Neubauer (1994). Rohrkletterer mit acht Beinen. In: P. Levi & Th. Bräunl (Eds.): *Autonome Mobile Systeme* **10**. Informatik aktuell. Springer. Berlin, Heidelberg, New York. 1–7.

J. O'Keefe & D. H. Conway (1978). Hippocampal place cells in the freely moving rat: why they fire where they fire. In: *Expl. Brain Res.* **31**. 573–590.

J. O'KEEFE & J. DOSTROVSKY (1971). The hippocampus as a spatial map. Preliminary evidence from unit activity in the freely moving rat. In: *Brain Res.* **34**. 171–175.

J. O'KEEFE (1979). A review of hippocampal place cells. In: *Progr. Neurobiology* **13**. 419–439.

M. OBERDORFER, A. ZELL, & P. LEVI (1995). NEUROPA: Ein paralleles System zum neuronalen Einparken eines Autonomen Mobilen Roboters. In: R. DILLMANN, U. REMBOLD, & T. LÜTH (Eds.): *Autonome Mobile Systeme* **11**. Informatik aktuell. Springer. Berlin, Heidelberg, New York. 182–192.

C. OWEN & U. NEHMZOW (1997). Middle Scale Navigation – A Case Study. In: *Spatial Reasoning in Mobile Robots and Animals.* AISB-97 Workshop. Manchester University. 104–112.

B. POUCET (1993). Spatial cognitive maps in animals: New hypotheses on their structure and neural mechanisms. In: *Psychological Review* **100**. 163–182.

T. J. PRESCOTT & C. IBBOTSON (1997). The early evolution of spatial behaviour: robot models of trace fossils. In: *Spatial Reasoning in Mobile Robots and Animals.* AISB-97 Workshop. Manchester University. 86–93.

A. J. PRESCOTT (1994). Explorations in reinforcement and model-based learning. Ph. D. Thesis. Dept. of Psychology. University of Sheffield. U.K.

G. PRITSCHOW, M. DALACKER, & J. KURZ (1994). Automatisierte Mauerwerksfertigung mit mobilen Robotern: Neue Herausforderungen für die Steuerungs- und Sensortechnik. In: P. LEVI & TH. BRÄUNL (Eds.): *Autonome Mobile Systeme* **10**. Informatik aktuell. Springer. Berlin, Heidelberg, New York. 326–337.

M. H. RAIBERT (1991). Legged robots. In: *Research Directions in Computer Science.* MIT Press. Cambridge Massachusetts. 371–406.

D. M. RAUP & A. SEILACHER (1969). Fossil foraging behaviour: computer simulation. In: *Science* **166**. 994–995.

U. REMBOLD (1988). The Karlsruhe Autonomous Mobile Assembly Robot. In: *Proc. IEEE Int. Conf. Robotics and Automation.* 598–603.

S. ROSSEL & R. WEHNER (1986). Polarization vision in bees. In: *Nature* **323**. 128–131.

M. ROZMANN & J. DETLEFSEN (1995). Radarsensorbasierte Kartenerstellung in Innenräumen für Navigationsaufgaben. In: R. DILLMANN, U. REMBOLD, & T. LÜTH (Eds.): *Autonome Mobile Systeme* **11**. Informatik aktuell. Springer. Berlin, Heidelberg, New York.

TH. ROSSMANN & F. PFEIFFER (1995). Simulation und Regelung eines Rohrkrabblers. In: R. DILLMANN, U. REMBOLD, & T. LÜTH (Eds.): *Autonome Mobile Systeme* **10**. Informatik aktuell. Springer. Berlin, Heidelberg, New York. 80–89.

TH. RÖFER & R. MÜLLER (1998). Navigation and Routemark Detection of the Bremen Autonomous Wheelchair. In: T. LÜTH, R. DILLMANN, P. DARIO, & H. WÖRN (Eds.): *Distributed Autonomous Robotics Systems.* Springer. Berlin, Heidelberg, New York. 183–192.

TH. RÖFER (1994a). VierLING – Quadruped with Integrated Neural Balance Control. In: M. MARINARO & P. MORASSO (Eds.): *Proc. Int. Conf. Artificial Neural Networks.* Springer. London, Berlin, Heidelberg. 1311–1314.

TH. RÖFER (1994b). VierLING – Vierbeinige Laufmaschine mit Integrierter Neuronaler Gleichgewichtssteuerung. In: B. KRIEG-BRÜCKNER, G. ROTH, & H. SCHWEGLER (Eds.): *ZKW-Bericht* (3/94). ISSN 0947–0204. Zentrum für Kognitionswissenschaften. Universität Bremen.

TH. RÖFER (1995a). Controlling a robot with image based homing. In: B. KRIEG-BRÜCKNER & CH. HERWIG (Eds.): *Tagungsband des Workshops "Kognitive Robotik", ZKW Bericht* (3/95). ISSN 0947–0204. Zentrum für Kognitionswissenschaften. Universität Bremen.

TH. RÖFER (1995b). Image based homing using a self-organizing feature map. In: F. FOGELMAN-SOULIE & P. GALLINARI (Eds.): *Proc. Int. Conf. Artificial Neural Networks* **1**. EC2 & Cie. 475–480.

TH. RÖFER (1995c). Navigation mit eindimensionalen 360° Bildern. In: R. DILLMANN, U. REMBOLD, & T. LÜTH (Eds.): *Autonome Mobile Systeme* **10**. Informatik aktuell. Springer. Berlin, Heidelberg, New York. 193–202.

TH. RÖFER (1997a). Controlling a Wheelchair with Image-based Homing. In: *Spatial Reasoning in Mobile Robots and Animals.* AISB-97 Workshop. Manchester University. 66–75.

TH. RÖFER (1997b). Routemark-Based Navigation of a Wheelchair. In: *Proc. Third ECPD International Conference on Advanced Robotics, Intelligent Automation and Active Systems*. Bremen. 333–338.

U. SCHMID, D. HERNÁNDEZ, B. KRIEG-BRÜCKNER, A. EISENKOLB, A. MUSTO, K. SCHILL, S. WIEBROCK, F. WYSOTZKI, C. SCHLIEDER, J. RENZ, & B. NEBEL (1997). Process Control and Planning for Movement in Space: Qualitative and Metric Approaches to Spatial Inference and the Analysis of Movements. Technical Report. Technische Universität Berlin (*Also: http://ki.cs.tu-berlin.de/˜sppraum/juni-paper4.ps*).

M. J. Scholl (1987). Cognitive Maps as Orienting Schemata. In: *Journal of Experimental Psychology* **13** (4). 615–628.

K. Schweizer & G. Janzen (1996). Zum Einfluß der Erwerbssituation auf die Raumkognition: Mentale Repräsentation der Blickpunktsequenz bei räumlichen Anordnungen. In: *Sprache und Kognition* **15** (4). 217–233.

B. Schölkopf & H. A. Mallot (1995). View-based cognitive mapping and planning. In: *Adaptive Behavior* **3**. 311–348.

U. K. Sharma & L. S. Davis (1988). Road Boundary Detection in Range Imagery for an Autonomous Robot. In: *IEEE Journal of Robotics and Automation* **4** (5). 515–523.

U. Siems, Ch. Herwig, & Th. Röfer (1994). SimRobot – Ein Programm zur Simulation sensorbestückter Agenten in einer dreidimensionalen Umwelt. In: B. Krieg-Brückner, G. Roth, & H. Schwegler (Eds.): *ZKW-Bericht* (1/94). ISSN 0947–0204. Zentrum für Kognitionswissenschaften. Universität Bremen (*Also: http://www.tzi.org/˜simrobot*).

M. Snaith & O. Holland (1992). The application of Temporal Difference Learning to the neural control of quadruped locomotion. In: *Towards a Practise of autonomous systems*. MIT Press. Cambridge Massachusetts.

R. S. Sutton (1991). Reinforcement Learning Architectures for Animats. In: J.-A. Meyer & S. W. Wilson (Eds.): *From animals to animats*. MIT Press. Cambridge, Massachusetts.

C. Thorpe, M. H. Hebert, T. Kanade, & S. A. Shafer (1988). Vision and Navigation for the Carnegie-Mellon Navlab. In: S. S. Iyengar & A. Elfes (Eds.): *Autonomous Mobile Robots* **2**. IEEE Computer Society Press. Los Alamitos, California. 401–412.

E. C. Tolman (1948). Cognitive maps in rats and men. In: *Psychological Review* **55**. 189–208.

O. Trullier, S. I. Wiener, A. Bertholz, & J.-A. Meyer (1997). Biogically Based Artificial Navigation Systems: Review and Prospects. In: *Progress in Neurobiology* **51**. Pergamon. 483–544.

M. A. Turk, D. G. Morgenthaler, K. D. Gremban, & M. Marra (1988). VITS – A Vision System for Autonomous Land Vehicle Navigation. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* **10** (3). 342–361.

K. J. Waldron, V. J. Vohnout, A. Pery, & R. B. McGhee (1984). Configuration design of the adaptive suspension vehicle. In: *Int. Jour. of Robotics Research* **3**. 37–48.

C. J. C. H. Watkins (1989). Learning with Delayed Rewards. Ph. D. Thesis. Dept. Psychology. Cambridge University.

R. Wehner & R. Menzel (1990). Do insects have cognitive maps? In: *Ann. Rev. Neurosci.* **13**. 403–414.

R. Wehner (1983). Celestial and Terrestial Navigation: Human Strategies – Insect Strategies. In: F. Huber & H. Markl (Eds.): *Neuroethology and Behavioural Physiology*. Springer. Berlin, Heidelberg, New York. 366–381.

C. R. Weisbin, G. d. Saussure, J. R. Einstein, & F. G. Pin (1989). Autonomous Mobile Robot Navigation and Learning. In: *IEEE Computer Magazine* (6). Institute of Electrical and Electronics Engineers, Inc. 29–35.

S. Werner, B. Krieg-Brückner, H. A. Mallot, K. Schweizer, & Ch. Freksa (1997). Spatial Cognition: The Role of Landmark, Route, and Survey Knowledge in Human and Robot Navigation. In: *Informatik '97 – Informatik als Innovationsmotor*. Informatik aktuell. Springer. Berlin, Heidelberg, New York.

G. v. Wichert (1994). ATHENA: Mobile Roboter für Mülldeponien – Bewegungsplanung für nicht holonome mobile Roboter. In: P. Levi & Th. Bräunl (Eds.): *Autonome Mobile Systeme* **10**. Informatik aktuell. Springer. Berlin, Heidelberg, New York. 316–325.

B. H. Wilcox, D. B. Gennery, A. H. Mishkin, & B. K. Cooper (1987). A Vision System for a Mars Rover. In: *SPIE Mobile Robots II* **852**. 172–179.

R. Wiltschko & W. Wiltschko (1995). Magnetic Orientation in Animals. Springer. Berlin, Heidelberg, New York.

R. Wiltschko (1997). The Navigation System of Birds. In: *Spatial Reasoning in Mobile Robots and Animals*. AISB-97 Workshop. Manchester University. 5–14.

Th. Wittmann (1995). Insektennavigation: Modelle und Simulationen. Teil 1: Wegintegration. In: B. Krieg-Brückner, G. Roth, & H. Schwegler (Eds.): *ZKW-Bericht* (2/95). ISSN 0947–0204. Zentrum für Kognitionswissenschaften. Universität Bremen.

Th. Wittmann (1996). Insektennavigation: Modelle und Simulationen. Ph. D. Thesis, Universität Bremen. Berichte aus der Biologie. Shaker. Aachen.