Lehrstuhl für Steuerungs- und Regelungstechnik
Technische Universität München

Aspects of Cognitive Understanding of the Environment by Vision-Based Semantic Mapping

Quirin A. Mühlbauer

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr.-Ing. habil. Gerhard Rigoll

Prüfer der Dissertation:

1. TUM Junior Fellow Dr.-Ing. Kolja Kühnlenz
2. Univ.-Prof. Dr.-Ing. Eckehard Steinbach

Die Dissertation wurde am 16.06.2010 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 12.10.2010 angenommen.

Foreword

This thesis summarizes the results of three years research at the Institute of Automatic Control Engineering, Technische Universität München, where I stayed from 2007 till now and would certainly not have been possible without the guidance, companionship, and friendship of many different persons.
I thank my "Doktorvater" Dr. Kolja Kühnlenz for providing the topic, guidance, and encouragement throughout the thesis and Prof. Martin Buss for giving me the opportunity to work at his institute, where I enjoyed the inspiring working environment.
In addition, I thank the ACE-Team, Andrea Bauer, Klaas Klasing, Georgios Lidoris, Florian Rohrmüller, Stefan Sosnowski, Tingting Xu, and Tianguang Zhang for fruitful discussions, always giving a helping hand, and the great time we had building and testing the robot. Also, I thank Daniela Feth, Kwang-Kyu Lee, and Bernhard Weber for being great office mates and for all the entertainment and distractions. Further thank goes to Michelle Karg, David Miller, Thomas Schauß, and to everyone else who was proofreading. Thanks to all other LSR-collegues, for always having an open ear, playing kicker after mensa, etc.
I want to thank all students who contributed to this work, especially Sebastian Hilsenbeck for the implementation of the median fusion algorithm and Christoph Vogl for the implementation of the stereo module and the ego-motion algorithm.
In deep gratitude I thank my family for their constant support, encouragement, and words of confidence. Thank you Anette for filling my life besides thesis writing.

Munich, June 2010 Quirin Mühlbauer

to my parents

Abstract

The role of mobile robots in our daily life has been increasing rapidly during the last decades, resulting in a greater need for autonomous behavior. Similar to the human sensory system, vision is one of the most important senses for mobile robotics. This thesis gives an insight into the cognitive abilities of current mobile robots. The presented research discusses the development and implementation of a cognitive architecture, formed by perception and its complement, cognition. Algorithms for object detection and mapping compose the perception system, while cognition is defined by a suitable knowledge representation and techniques for decision making.

Methods for both two- and three-dimensional object detection are introduced. A novel two-dimensional object detection algorithm is based on a cascade of three different histograms, namely color histograms, histograms of oriented gradients, and color co-occurrence histograms. This cascade is real-time capable, robust to occlusions, and requires few training images. Additionally, this thesis proposes an algorithm for the estimation of human body poses. This algorithm is using three-dimensional point clouds as input and can easily be extended to detect other skeleton-based objects. Mapping is another significant task for a mobile robot with cognitive abilities. Two different approaches for vision-based mapping are presented, one based on two-dimensional images and the other based on three-dimensional point clouds. The two-dimensional mapping algorithm extends the state-of-the-art with a memory, allowing the robot to remember old terrain. Moreover, a sophisticated real-time stereo reconstruction algorithm with integrated ego-motion estimation and a novel genetic iterative closest point algorithm for sensor fusion is presented. The cognition system is marked by semantic maps, a knowledge representation combining semantic networks and metric maps. A sound mathematical description of semantic maps is introduced, which is used to derive methods for adding new knowledge and for action planning. Both simulations and experiments have been conducted to verify the presented cognitive architecture. By introducing new algorithms for perception and cognition and by enhancing known algorithms, this thesis contributes to advance the cognitive abilities in mobile robotics.

Zusammenfassung

In den letzten Jahren haben sich mobile Roboter zu einem immer wichtiger werdenden Bestandteil unseres täglichen Lebens entwickelt, wobei autonomes Verhalten immer mehr in den Vordergrund rückt. Ähnlich wie beim Menschen ist das Sehen der wichtigste Sinn eines mobilen Roboters. Diese Dissertation gibt daher einen Einblick in die kognitiven Fähigkeiten bereits existierender Roboter und stellt neue Forschungsergebnisse in diesem Bereich vor. Kognitive Fähigkeiten setzen sich aus der Perzeption sowie deren Gegenstück, der Kognition zusammen. Algorithmen für das Erstellen von Karten und das Erkennen von Objekten bilden die Perzeption, während die Kognition durch eine geeignete Wissensdarstellung und Methoden zur Entscheidungsfindung geprägt wird.
Diese Arbeit präsentiert zwei Methoden zur Objekterkennung, die auf zwei- beziehungsweise dreidimensionalen Bildern basieren. Ein neuartiger Algorithmus zur Erkennung der Körperhaltung verwendet dreidimensionale Punktwolken, während zur allgemeinen Objekterkennung zweidimensional Bilder verwendet werden. Der präsentierte Algorithmus basiert auf einer Kaskade aus drei verschiedenen Histogrammen und verwendet sowohl farb- als auch räumliche Informationen, ist echtzeitfähig, erkennt Objekte trotz Verdeckungen und benötigt wenig Trainingsbilder. Für das Erstellen von Karten werden ebenfalls zwei Algorithmen präsentiert, die analog zur Objekterkennung auf zwei- beziehungsweise dreidimensionalen Darstellungen basieren. Der zweidimensionale Algorithmus verwendet einen Speicher, um alte Bodentexturen wiederzuerkennen. Weiterhin wird ein fortschrittlicher Stereoalgorithmus mit integrierter Schätzung der Kamerabewegung vorgestellt, sowie eine neue Methode zur Fusionierung verschiedener Sensordaten. Das vorgestellte kognitive System basiert auf semantischen Karten, einer Kombination aus semantischen Netzwerken und metrischen Karten. Eine mathematische Beschreibung für semantische Karten wurde eingeführt, von der Methoden zum Hinzufügen neuen Wissens sowie zum Planen von Aktionen abgeleitet wurden. Abschließend wird die vorgestellte kognitive Architektur durch Simulationen und Experimente validiert.
Indem neue Methoden der Perzeption und Kognition entwickelt und bereits bekannte Algorithmen verbessert werden, trägt diese Arbeit wesentlich zur Weiterentwicklung der kognitiven Fähigkeiten mobiler Roboter bei.

Contents

1 Introduction **1**
- 1.1 State-of-the-Art . . . 3
- 1.2 Challenges . . . 8
- 1.3 Main Contributions and Outline of Thesis . . . 8

2 Robot System Development **11**
- 2.1 The Autonomous City Explorer . . . 11
- 2.2 Software Architecture of a Semantic Mapping System . . . 13
  - 2.2.1 Object Detection . . . 15
  - 2.2.2 Vision-Based Mapping . . . 16
  - 2.2.3 Semantic Mapping . . . 17

3 Object Detection **19**
- 3.1 Overview of different Object Detection Algorithms . . . 20
- 3.2 Stereo Image Processing . . . 22
  - 3.2.1 Computation of Matching Costs . . . 23
  - 3.2.2 Aggregation of Costs . . . 24
  - 3.2.3 Computation of the Disparity . . . 26
- 3.3 Three-Dimensional Object Detection . . . 27
  - 3.3.1 Pre-Processing . . . 27
  - 3.3.2 Segmentation of Three-Dimensional Point Clouds . . . 29
  - 3.3.3 Object Detection using Local Attributes . . . 30
  - 3.3.4 Human Body Pose Estimation . . . 32
- 3.4 Two-Dimensional Object Detection . . . 38
  - 3.4.1 Conventional Algorithms . . . 38
  - 3.4.2 Histogram-based Object Detection Cascade . . . 39
  - 3.4.3 Different Types of Histograms . . . 40
  - 3.4.4 Intersection of Histograms . . . 43
  - 3.4.5 Implementation Details . . . 44
- 3.5 Experimental Results . . . 46
  - 3.5.1 Experimental Setup . . . 46
  - 3.5.2 Stereo Image Processing . . . 46
  - 3.5.3 Human Body Pose Estimation . . . 48
  - 3.5.4 Detection of Traffic Signs using a Cascade of Haar-Like Features . . . 50
  - 3.5.5 Object Detection using a Cascade of Histograms . . . 51
- 3.6 Discussion . . . 54

4 Vision-Based Mapping **55**
4.1 Overview of current Vision-Based Mapping Systems . . . 56
4.2 Two-Dimensional Mapping . . . 58
4.2.1 Detection of the Ground . . . 59
4.2.2 Computation of the Reference Texture . . . 61
4.2.3 Back-Projection . . . 61
4.3 Three-Dimensional Mapping . . . 65
4.3.1 System Architecture . . . 65
4.3.2 Stereo Vision-Based Ego-Motion Estimation . . . 67
4.3.3 Median Fusion Algorithm . . . 68
4.3.4 Obstacle Detection in Three-Dimensional Maps . . . 70
4.4 Mapping with Different Sensor Types . . . 72
4.4.1 Requirements . . . 72
4.4.2 Calibration of the Laser Rangefinders . . . 74
4.4.3 Registration of Vision and Laser Data . . . 75
4.5 Experimental Results . . . 79
4.5.1 Experimental Setup . . . 79
4.5.2 Two-Dimensional Mapping . . . 79
4.5.3 Three-Dimensional Mapping . . . 81
4.5.4 Sensor Fusion . . . 84
4.6 Discussion . . . 86

5 Semantic Maps **89**
5.1 Introduction to Semantic Networks . . . 89
5.1.1 Basic Types of Semantic Networks . . . 90
5.1.2 Overview of current Semantic Knowledge Representations . . . 91
5.2 Semantic Maps Approach . . . 92
5.2.1 Mathematical Representation of Semantic Maps using Set Theory . . . 92
5.2.2 Introducing States for Objects . . . 94
5.2.3 Similarity Probability of Objects . . . 96
5.2.4 Action Planning using Object States . . . 97
5.3 Implementation . . . 100
5.3.1 Simulator . . . 101
5.3.2 Integration of Semantic Maps with Mapping and Object Detection . . . 101
5.3.3 Semantic Robot Client . . . 102
5.4 Simulation Results . . . 103
5.5 Experimental Results . . . 105
5.6 Discussion . . . 106

6 Conclusion **109**
6.1 Concluding Remarks . . . 109
6.2 Suggestions for Future Work . . . 111

A System Overview of ACE **113**
A.1 Hardware . . . 113

A.2 Software Architecture 113

B Introduction to CUDA **117**
B.1 Overview of CUDA 117
B.2 Integration of a CUDA Application 118

C Computation of the Camera Motion **119**

Notations

Abbreviations

ACE	Autonomous city explorer
CAD	Computer-aided design
CCD	Charge-coupled device
CCH	Color co-occurrence histogram
CH	Color histogram
CUDA	Compute unified device architecture
CPU	Central processing unit
DSI	Disparity-space image
DOF	Degree of freedom
FPS	Frames per second
FSM	Finite state machine
GPU	Graphics processing unit
GUI	Graphical User Interface
HOG	Histogram of oriented gradients
HSV	Hue, saturation and value color space
ICP	Iterative closest point algorithm
LRF	Laser rangefinder
RGB	Red, green and blue color space
ROI	Region of interest
RT	Real-time
SIFT	Scale invariant feature transformation
SLAM	Simultaneous localization and mapping
SVD	Singular value decomposition
SVM	Support vector machines
UDP	User datagram protocol
USB	Universal serial bus

Conventions

Scalars, Vectors, and Matrices

Scalars are denoted by upper and lower case letters in italic type. *Vectors* are denoted by bold lower case letters. The vector $\mathbf{x}$ is composed of elements x_i. A cartesian point $\mathbf{p}$ is composed of elements p_x, p_y and p_z. *Matrices* are denoted by upper case letters in bold type. The matrix $\mathbf{M}$ is composed of elements m_{ij} (i^{th} row, j^{th} column).

x or X	Scalar
$\mathbf{x}$	Vector
$\mathbf{X}$	Matrix
$\mathbf{X}^T$	Transposed of $\mathbf{X}$
$\mathbf{X}^{-1}$	Inverse of $\mathbf{X}$
$\mathbf{X}^+$	Pseudoinverse of $\mathbf{X}$
$f(\cdot)$	Scalar function
$\mathbf{f}(\cdot)$	Vector function

General Symbols

$r_{i,j}$	Red component of a pixel (RGB color space)
$g_{i,j}$	Green component of a pixel (RGB color space)
$b_{i,j}$	Blue component of a pixel (RGB color space)
$h_{i,j}$	Hue component of a pixel (HSV color space)
$s_{i,j}$	Saturation component of a pixel (HSV color space)
$v_{i,j}$	Value component of a pixel (HSV color space)
$\mathbb{C}$	Point cloud
i	Index variable
N	Number of points in a point cloud $\mathbb{C}$
$\mathbf{p}$	Point of a three- or two-dimensional space
$\mathbf{R}$	Rotation matrix
${}^A\mathbf{T}_B$	Transformation matrix
$\mathbf{t}$	Translation vector

Extrinsic and Intrinsic Camera Parameters

γ	Field of view
Θ	Pitch angle
Φ	Roll angle
Ψ	Yaw angle

B	Stereo basis
f	Focal length
$l_x \times l_y$	Chipsize $x-$ and $y-$direction
$p_x \times p_y$	Number of pixels in $x-$ and $y-$direction
$\mathbf{s_p}$	Position of the camera with respect to S_0

Object Detection

α	Weighting factor
ζ	Confidence
ζ_{Min}	Threshold for the confidence
$\theta_{i,j}$	Orientation at pixel i, j
a_i	Penalty factor for link i
$\mathbf{B}^R$	Reference set of vectors of attributes
$\mathbf{b}$	Vector of attributes
$\mathbf{b}_i^R$	Vector of reference attributes
b_i	Attribute
$C_0(i, j, d)$	Element of DSI
$C_A(i, j, d)$	Element of the updated DSI
$C_I(i, j, d)$	Intensity cost of element i, j, d
$C_G(i, j, d)$	Gradient cost in of element i, j, d
C_{Max}	Maximal cost
d	Disparity
$d_{i,j}$	Length of gradient at pixel i, j
$e(H, H_r)$	Intersection error between a histogram H and a reference histogram H_r
e_i	Error metric for link i
f	Scale factor
H	Histogram
H_r	Reference Histogram
$h(x, y, z)$	Three-dimensional gaussian distribution
I	Input image
I^L	Left input image
I^R	Right input image
$I(i, j)$	Pixel of input image
I_{cch}	Input image for color co-occurrence histogram
I_{ch}	Input image for color histogram
I_{hog}	Input image for histogram of oriented gradients
$I_{\text{ref, i}}$	Reference image with index i
$\mathbf{K}$	Kernel of a filter
m_{cch}	Error map for color co-occurrence histograms
m_{ch}	Error map for color histograms
m_{hog}	Error map for histograms of oriented gradients
$N \times N$	Filter window size

N_c	Number of points used for computation
$\mathbf{n}$	Point of a segment
n_{cch}	Number of colors in color co-occurrence histogram
n_{ch}	Number of colors in color histogram
n_{dist}	Number of distances in color co-occurrence histogram
n_{hog}	Number of orientations in histogram of oriented gradients
$n_x \times x_y$	Number of subregions in $x-$ and $y-$direction
$\mathbf{p}_L$	Point of left image
$\mathbf{p}_R$	Point of right image
$\mathbf{p}_r$	End point of a reference link
$\mathbf{p}_s$	End point of a detected link
t^i_{cch}	Threshold for color co-occurrence histograms and reference image $I_{\text{ref, i}}$
t^i_{ch}	Threshold for color histograms and reference image $I_{ref,i}$
t^i_{hog}	Threshold for histograms of oriented gradients and reference image $I_{\text{ref, i}}$
w	Weighting factor
$x_i \times y_i$	Size of input image I
$x_r \times y_r$	Size of reference image I_{ref}
$x_s \times y_s$	Size of subregion

Vision-Based Mapping

α	Rotation of the robot
γ	Field of view
σ	Standard deviation
$C(i,k,k,l)$	Cost value
$\tilde{C}_j$	Point cloud with points used for computation
$d(i,j)$	Disparity of a pixel
$d(\mathbf{p_1}, \mathbf{p_2})$	Distance between the two points $\mathbf{p_1}$ and $\mathbf{p_2}$
d_i	Distance measurement for iteration i
d_x	Real distance to a point x
d'_x	Distance in an image to a point x
d''_x	Distance in a virtual image to a point x
$f_j(d_j)$	Probability function for a distance measurement
$\mathbf{I_v}$	Individual of the genetic algorithm
i	Index variable of the current position in the ring buffer
j	Index variable of a pixel in an image
k	Index variable of a histogram in the ring buffer
m^k_j	Weighting factor
N_b	Size of the ring buffer
N_{Ego}	Window size
N_{Search}	Window size
$\mathbf{p_i}^j$	Point with index i from point cloud C_j
$\tilde{\mathbf{p}}^j_{\mathbf{i}}$	Point used for computation

S_0	Robot frame
S_c	Camera frame
s_{Max}	Maximal saturation component of a pixel
s_{Min}	Minimal saturation component of a pixel
${}_c t_x$	Movement of the robot in x-direction
${}_c t_y$	Movement of the robot in y-direction
t_n	Threshold
t_w	Threshold
$\mathbf{V}_{\text{Current}}(i,j)$	Corner of the current image
$\mathbf{V}_{\text{Previous}}(i,j)$	Corner of the previous image
v_{Max}	Maximal value component of a pixel
v_{Min}	Minimal value component of a pixel

Semantic Mapping

α_k	Weighting factor
$\Lambda_i(\cdot)$	Update function
$\Lambda_i^{-1}(\cdot)$	Inverse update function
σ	Costs for performing an action
$\Phi(\cdot)$	Search function
$\chi_{\mathcal{O}_j}^{\mathcal{A}_i}$	State variable of object j with respect to the attribute i
$\widehat{\chi}_{\mathcal{O}_j}^{\mathcal{A}_i}$	Abstract state variable for planning
$\Psi^{\boldsymbol{\mathcal{R}}}(\mathcal{R})$	Get-connection function
$\Omega^{\boldsymbol{\mathcal{R}}}(\mathcal{R})$	Search function
$\mathcal{A}$	Node of type *attribute*
$\boldsymbol{\mathcal{A}}$	Set of nodes of type *attribute*
$\mathbf{C}$	Action list
$\mathcal{C}$	Node of type *action*
$\boldsymbol{\mathcal{C}}$	Set of nodes of type *action*
$d(\chi_{\mathcal{O}_i}^{\mathcal{A}_k}, \chi_{\mathcal{O}_j}^{\mathcal{A}_k})$	Distance between two states
$p(\mathcal{O}_i, \mathcal{O}_j)$	Probability of similarity
$\mathcal{O}$	Node of type *object*
$\boldsymbol{\mathcal{O}}$	Set of nodes of type *object*
$\mathcal{R}$	Node of unspecified type
$\boldsymbol{\mathcal{R}}$	Set of nodes of unspecified type
$s(\mathcal{O}_\rangle, \mathcal{O}_\vert)$	Similarity between two *objects*
$\mathcal{T}$	Node of type *type*
$\boldsymbol{\mathcal{T}}$	Set of nodes of type *type*
$(\mathcal{R}_i, \mathcal{R}_j)$	Connection between two nodes
X_i	State space of node i
Y_i	Reduced state space of node i

> *Miss Glory, robots are not people. They are mechanically much better than we are, they have an amazing ability to understand things, but they don't have a soul.*
>
> Karel Čapek, 1921

1 Introduction

With the begin of industrialization the desire for a mechanical working man grew stronger and stronger. After the first appearance of the term *robot* in Karel Čapec's play *R.U.R (Rossum's Universal Robots)* [22] the desire was omnipresent in literature and later in moving pictures - it reflected the zeitgeist of the society, when almost everything seemed possible. The term robot was originally used to describe an artificial man and was later attributed by the media to the term *metal man*. Their increasing presence came with an increasing fear of their superiority. In R.U.R. robots were originally designed to perform minor or unpleasant work, but were equipped with sophisticated mechanical and cognitive abilities. Later they take over and eventually wipe out humankind. In the year 1950 Isaac Asimov wrote the novel *I, robot* [6] and came up with his famous three laws of robotics, stating a robot must never harm or kill a human and must obey all humans' orders. However, recent activities in the area of military research and the advances in the development of unmanned aerial vehicles (UAVs) [148] have shown that these laws play a subordinate role in the real world. This raises the question, if the fear of robots is legitimate and they will rule the world. Fortunately, this fear is unsubstantiated at their current state of development.

Before the cognitive abilities of mobile robots can be examined further, the terms perception and cognition have to be defined. Perception means the reception and collection of data, while cognition marks the complement to perception and is often referred to as the process of thought in order to plan the future based on past observations and experiences. Now, the cognitive abilities of robots can be examined further in detail. Early fictional art focuses on what seemed to be the main challenge - the mechanical development of robots - and neglects, how human-like cognitive abilities can be achieved. This reflects the current state of development, where complex bipedal walkers like Honda's ASIMO with sophisticated mechanics are constructed. Compared to the cognitive skills of fictional robots,

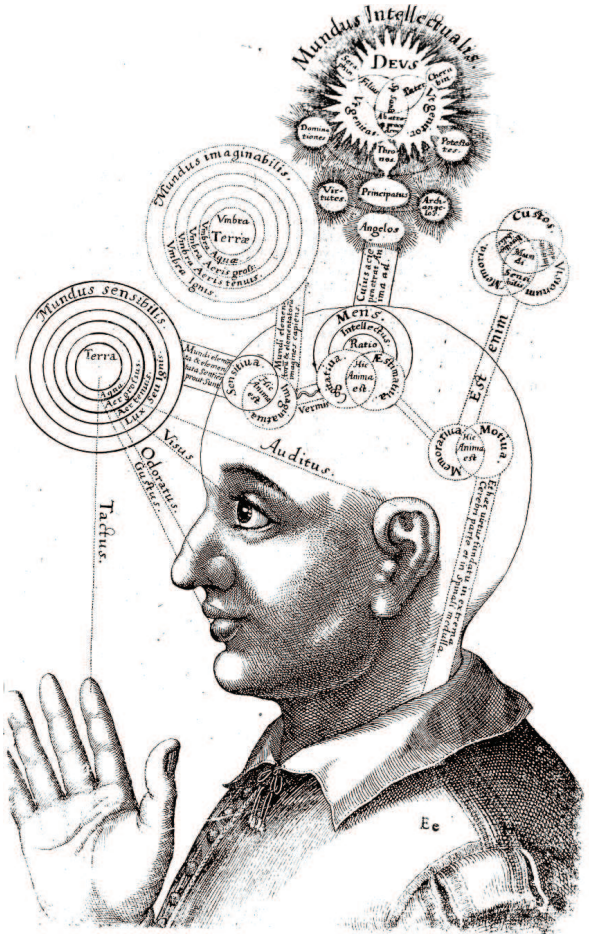


Fig. 1.1: Illustration of perception (Robert Fudd, 1619).

current developments are still way behind. The majority of the existing robots is placed on production lines and is manufacturing objects. Other applications include assisting humans in the household and performing work in hazardous or distant environments, e.g. mine-defusing. These mobile robots require a much higher degree of autonomy and thus more sophisticated perception and cognition systems. Despite the current achievements in the field of perception, robots are nothing more than tools functioning in controlled environments in a very specific manner. By integrating a vision system and actuators in one package, Sony's AIBO tries to fill this gap by providing simple cognitive behavior. The *robot hall of fame*[1] provides a comprehensive overview of both the most sophisticated and important fictional robots and milestones in the development of real robotic platforms.
Enhancing the cognitive abilities is a major challenge on the road to full autonomy. Most of the current cognitive architectures are inspired by the human brain as illustrated by Robert Fudd in Figure 1.1 and consist of three major parts: perception (mundus sensibilis), knowledge representation (mundus imaginabilis), and cognition (mundus intellectualis). Knowledge is mostly stored in a semantic manner, meaning by using words describing the objects and not by using an accurate model. Semantic representations allow much data

[1] `http://www.robothalloffame.org/`

to be stored, reflecting not only the type, position, and attributes of the object, but also its meaning. Of all our senses, vision is the most important one when it comes to understand the environment. Cameras provide an easy to use and cheap sensor for mobile robotics. Consequently, the vision system of a mobile robot should be its main source for the cognitive system. Other demands are also inspired by the human system and include that the cognitive architecture should be a prospective and anticipative system, not only planning the future and behaving accordingly [86], but also foreseeing the future in the same way humans do [150]. Learning from experience requires some sort of self-modification [151] and interpretation of the environment [45]. Summing up, a cognitive architecture in mobile robotics links perception and cognition with a suitable knowledge representation and provides methods for decision making and planning. As no low level planning of movements is performed, decision making and planning are regarded to be part of the cognition system. When it comes to mobile robotics, perception is formed by object detection and mapping. Using two-dimensional images is the most intuitive and easiest way to gather information. However, by ignoring the third dimension valuable information gets lost. Gathering three-dimensional images is inspired by the human vision system. By using two or more synchronized cameras with different viewpoints, objects will appear at different positions in the camera images. The different positions can be used to estimate the position of the object in three dimensions, a process called stereo-reconstruction. Both representations have advantages and disadvantages. Two-dimensional images are examined easily and three-dimensional images provide more accuracy, but require more complex algorithms and more computational power. Two-dimensional maps are easy to create and are sufficient for navigation. On the contrary, complex manipulation tasks can only be planned using a three-dimensional map. Detected objects and their attributes can be stored in a semantic network, which is a textual representation of knowledge and which can be used to link predefined knowledge with the objects. By connecting the semantic network with a metric map, a comprehensive knowledge base is created, which will be referred to as semantic map. In a last step, this knowledge base is used for decision making and planning. The work presented in this thesis investigates the three main aspects of cognitive architectures: object detection, mapping, and cognition. In the following section, previous research in the field of cognitive abilities for mobile robotics are presented and the main challenges are highlighted.

1.1 State-of-the-Art

Vision-based cognitive understanding of the environment requires sophisticated algorithms and methods from several different research areas. On one side, an advanced perception system is required, while the gained data has to be processed on the other side. Furthermore, the whole system has to form a suitable cognitive architecture. This section gives an overview of modern cognitive architectures and presents some robot systems, which are equipped with a cognitive architecture or at least with basic cognitive abilities. A detailed discussion of the state-of-the art of both the perception modules, namely object detection and vision-based mapping, and the cognitive processing module, the semantic mapping, can be found in the corresponding chapters.

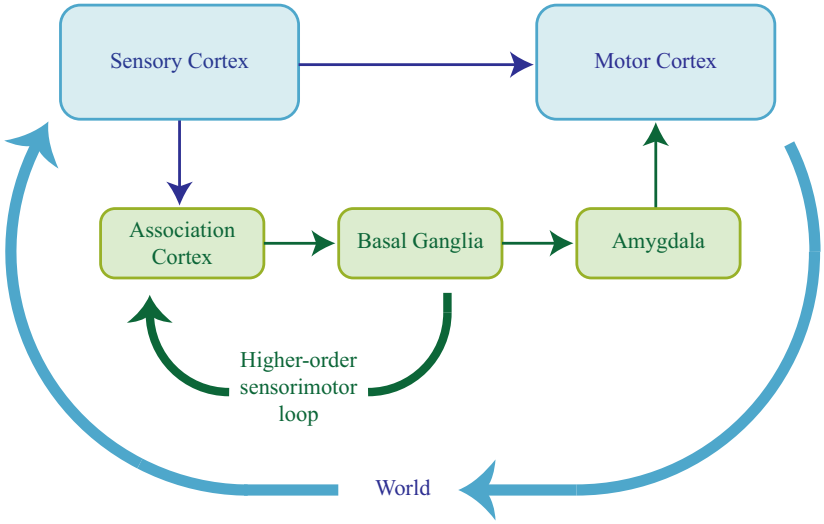


Fig. 1.2: Cognitive architecture simulating a biological neuronal network.

Knowledge Representations

A robot with cognitive abilities or a cognitive architecture requires a well suited and sophisticated knowledge representation. As wit the human memory, the information is broken to simple key points. This reduction brings several advantages, such as memory efficiency and possible generalization. Artificial neuronal networks [54] are inspired by the human brain and allow both storage of knowledge and the utilization of this knowledge, e.g. to recognize objects. They are composed of a multiplicity of simple units [9] like neurons. Expert systems [42] have been developed to assist in medical and scientific analysis by using a set of rules or predefined data to draw logic conclusions. The programming language PROLOG was developed to allow declarative programming. Its main data structures are called *facts* and *rules*. Other approaches try to link objects to certain attributes. Frame structures [63] are based on *frames*, containing a name and several attributes. However, there are no links between the frames. Semantic networks [49, 115, 135] are composed of nodes, representing objects or attributes and edges, representing relations between the nodes. Probabilistic approaches use probability functions to model occurring uncertainties.

Cognitive Architectures

During the last decades a variety of cognitive architectures has been developed. By the distinctive paradigms they are based on, most of them can be sorted into two main categories: **Emergent Systems**, which are mostly composed of artificial neuronal networks, and **Symbolic Processing Systems**, sometimes also called the cognivist approach.

Emergent systems have evolved from neuroscience and are based upon the imitation of the structure and aspects of biological neuronal networks. Likewise the biological archetype, a cognitive process is not separated into perception, recognition, classification,

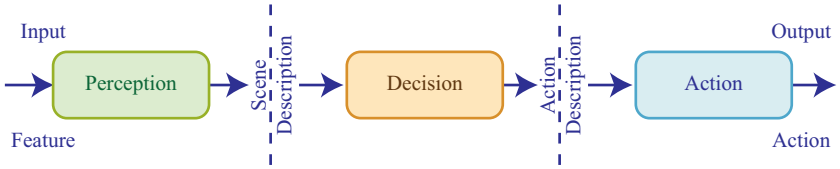


Fig. 1.3: Architecture of a symbolic processing system.

and action planning. As illustrated in Figure 1.2, the central processing of perception is conducted in the sensory cortex, while the association cortex combines the information with past experiences. Although there is currently no full understanding, the higher planning is assumed to be conducted by the basal ganglia. After the information has been associated with emotions in the amygdala, the actual movements are planned by the motor cortex. An input activates certain areas of the emergent systems, leading to a certain action. Sometimes, simulated biological architectures will lead to astonishing behavior when a robot is subject to deal with completely new situations. On the other hand, the system may be overstrained with relatively simple situations. Instead of using accurate representations like geometrical shapes or color histograms, emergent systems try to remember the resulting action of perceptions and try to guess which actions suite the current inputs best. Hence, the system uses no visual templates, but the sensorimotor's integration between the perception and the action [26]. Artificial neuronal networks [54] mark the most popular representation. However, emergent systems are not limited to artificial neuronal networks, they are decentralized complex systems from a multiplicity of simple units and interactions between the units. In contrast to neuronal networks, interaction is organized and is not limited to internal components. A perception is no longer the result of a single isolated loop and image processing systems are highly integrated into the entire emergent system.

In contrast to emergent systems, **symbolic processing systems** are created in a bottom-up manner with an explicit separation of the single steps. According to Vernon et al. [150, 151], a cognitive process can be separated into three different levels of abstraction: on the lowest level, *visual features* are abstracted by the perception system. Several features compose a *stimulus*, which lead to a final *action* as response to the original visual features. Figure 1.3 shows the different stages of a symbolic processing system. In the first step, a perception system processes all objects in the robots field of view and generates certain features, which serve as stimuli for the decision. Now, a description of the scene can be built, which serves as a basis for decision making, where finally actions are generated [45]. Consequently, perception and decision making can be considered as the most important parts of a symbolic processing system. Hence, most of the current symbolic processing systems vary in strategies for these two major parts. Different symbolic processing systems can be distinguished by the utilization of different probabilistic frameworks for decision making, such as Bayesian theory [34], which has won a wide range of interest.

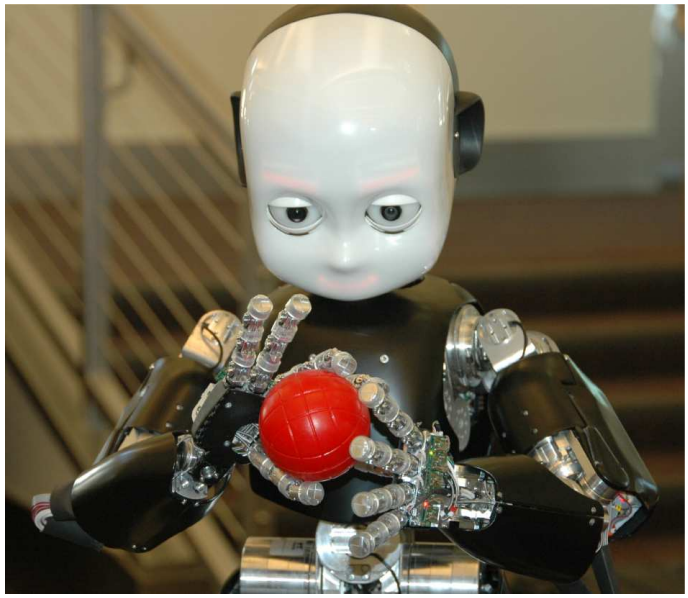


Fig. 1.4: The Robot iCub [119].

Current Implementations of Cognitive Architectures

The creation of cognitive architectures has been a large branch in computer science since the creation of the first neuronal network in the 1960's. Since then, several different approaches have evolved. In their current state of development, these architectures require vast amounts of computational power and memory or are not real-time capable and thus uneligible for the use on a mobile robot. However, the computational power of modern computers is increasing exponentially.
Cognitive architectures have been developed as artificial intelligence (AI), i.e. to model human behavior. In SOAR [73], solving problems is realized as a search in a problem space, while knowledge is represented by rules or objects. Adaptive control of thought-rational (ACT-R) [4] is composed of a set of predefined reproduction rules and a declarative memory containing simple knowledge items. A programming language like LISP can be used to predefine the rules. Other cognitive architectures try to model the biological cognition system from low level perception to high level reasoning (LIDA) [35] or use multi-agent systems like Cougaar (Cognitive Agent Architecture) [1]. Other widely known cognitive architectures include cognitive systems for cognitive assistants (CoSy) [26], cognitive systems that self-understand and self-extend (CogX) [111], and the cognitive robot companion (Cogniron) [149].

Robots with Cognitive Abilities

There exists a wide variety of mobile robots with more or less cognitive abilities. Some robot systems are equipped with a complex cognitive architecture, while others have been designed for other purposes and are only equipped with an image processing system and no cognitive architecture. Robots with cognitive architectures try to analyze and interpret

their environment by performing a semantic analysis [33, 102] and interpretation [21] to create a semantic network of the scene, containing information about all identified objects. The mobile robot can then use this semantic network to interact with the environment. Some approaches use vision data to recognize objects and compute their position [46] or use extensions of the constellation object model, which is popular in computer vision.
Robots serving as a museum guide are equipped with complex cognitive architectures [100, 125]. Besides robust localization and mapping, obstacle detection and object recognition, they have to be able to interact with humans and understand certain commands [134]. Other robots have been developed to assist humans in the household [117, 126]. As the main focus of application is the support of elderly people, a main focus of research is robustness. Impressive results have been shown in scope of the DARPA grand challenges in the years 2004, 2005, and 2007 [142], where autonomous cars drove across the desert in the first years. In the last grand challenge, these cars have proven to be able to drive within regular traffic obeying traffic rules.
Figure 1.4 shows the iCub robot [119, 132], one of the most popular examples for an emergent system. Its sophisticated hardware has a physical appearance similar to a 3.5-year old child and was designed as a humanoid platform to investigate the embodiment of cognition and the development of cognitive abilities. The main focus of research is self-development through learning from the environment, by interactive exploration, manipulation, imitation, and gestural communication. iCub's cognitive architecture consists of three levels: the perceptuo-motor skills, the action selection, and the internal action simulation.
Due to their modular nature and more stringent separation of perception and cognitive abilities and therefore better applicability, many robots have been developed based on symbolic processing architectures. Nüchter et al. presented the mobile robot Kurt3D [140], which uses a laser rangefinder to create a three-dimensional representation of the environment and a semantic processing system to analyze the scene. A semantic representation containing spatial information of different object types is predefined. By analyzing the geometric relations, the robot can detect floor, walls, ceiling, and other objects like doors.
A sophisticated technical approach of the symbolic processing paradigm is the explorer system [110, 137], which was designed for spatial exploration with the long-term aim to investigate artificial cognitive systems that are able to understand the environment. Special research interest was put in what, where, how, and why the robot can do things. Therefore a common understanding of the space between the robot and a human interaction partner has to be generated to achieve sufficient human-like behavior. A key issue was the semantic modeling of space, which was conducted on different levels of abstraction. A *metric map*, containing basic geometric features like lines, can be built using standard SIFT based SLAM techniques. This map can be abstracted into the *navigational map*, containing nodes and edges. Nodes represent possible positions of the robot and edges possible paths between the positions. As indicated by the name, this map is used for navigation. Several of these nodes can be combined to form a node of the *topological map*. Such a node can represent an area like a certain room. The highest level of abstraction is achieved by the *conceptual map*, linking the areas of the topological maps with each other and further semantic information.

1.2 Challenges

The development of robots with cognitive abilities comes with several challenges in the fields of robotics, computer vision, and cognitive architectures. These challenges can be related either to perception or to cognition. The main aspects targeted in this thesis are summarized below.
As shown in the state-of-the-art, the perception system is one of the key-issues. Some research of the most sophisticated cognitive architectures deals mainly with the cognitive abilities and assumes a perfect perception. Other cognitive architectures, like the iCub, are highly integrated with perception, making the exchange of modules difficult. As the perception system is formed by two different subsystems, object detection and mapping, different requirements arise for each subsystem. Two of the key requirements for the used algorithms are real-time capability and robustness. Consequently, not all algorithms are suitable for a cognitive architecture. An additional requirement for an object detection algorithm is the capability to detect attributes of an object. To allow online learning, few training images must be sufficient. When it comes to interaction with humans, specialized algorithms which are able to detect human body poses are eligible. Not only the type of an object is of great importance, but also its position. Two-dimensional maps are furthermore required for navigation, but they are unsuitable for complex manipulation tasks, requiring three-dimensional maps. As a laser rangefinder is unsuitable for some kinds of robots like biped walkers, these maps have to be created using only vision. Hence, particular interest has to be laid on the applicability of the developed algorithms.
Besides perception, cognition marks another key challenge for the development of robots with cognitive abilities. A suitable knowledge representation is required as a base for cognition. This knowledge representation should provide functions for adding new knowledge and for linking detected objects with prior knowledge. One of the most important aspect for the actual cognition is the integration of the different object detection and mapping algorithms with the knowledge representation. In addition, a cognitive architecture should be able to deal with uncertainties and provide functions for decision making based on the achieved knowledge.
As different areas of application come with different requirements, the cognitive architecture should be adaptable. This is achieved by creating a modular architecture, where modules can be replaced by other modules and new modules can be included easily. Thus another aspects arises, namely the thorough design of a convenient software architecture.

1.3 Main Contributions and Outline of Thesis

A mobile robot with cognitive abilities requires a sophisticated cognitive architecture with a perception system and a cognitive processing system. Considering todays robots with cognitive abilities, there is a lack of robots with comprehensive cognitive architectures, which can be adapted to the programmers needs. As the cognitive system developed in the scope of this thesis is modular, and the modules can be considered and even used independently, it allows manifold areas of application and can be extended easily. Due to the modularity, the cognitive system can be classified as a symbolic processing system. The

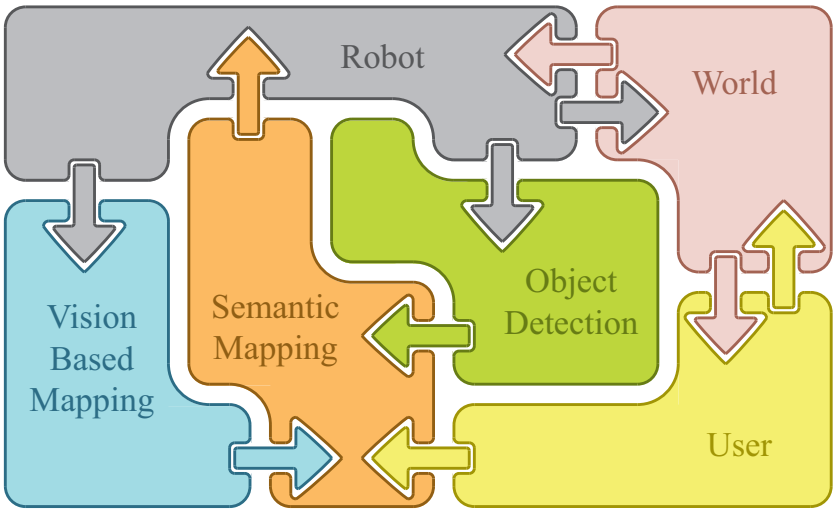


Fig. 1.5: Illustration of a symbolic processing system, integrating semantic mapping.

perception is split into two major parts: object detection and mapping. A guide through this thesis is given in the following section, summarizing the main contributions.
Figure 1.5 illustrates the semantic mapping system, a cognitive architecture developed in the scope of this thesis. The main aspects of the general system architecture are the perception system formed by the object detection and the mapping, and the cognitive processing, formed by the semantic mapping. A user can interact with the robot directly via the semantic mapping, or indirectly via interaction with the world.

System Architecture

Chapter 2 presents details about the software architecture and the autonomous city explorer (ACE), a mobile robot that was co-developed in the scope of this thesis and serves as hardware platform. ACE is equipped with a camera head, two stereo cameras and two PCs, one for navigation and interaction and one for vision processing. The proposed software system is highly modular and is distributed between different processes and threads on different machines. Designing clear and logical interfaces between the system, the subsystems, the modules, and the submodules is vital for a distributed system and was consequently of particular interest during the design process of the software architecture. Therefore, a stringent hierarchy with different communication protocols in each layer has been developed, allowing an almost arbitrary distribution of the subsystems on different machines.

Perception System

Sophisticated object detection and vision-based mapping form the perception system. Both provide methods and algorithms to process two-dimensional and three-dimensional data. Two-dimensional images can be obtained easily from a camera, while a three-dimensional representation has to be computed by using stereo image processing. Consequently, Chapter 3 starts with the introduction of a real-time stereo algorithm, which is able to process

images with high resolutions using CUDA, NVIDIA's stream computing architecture. The stereo algorithm computes a three-dimensional point cloud, which can be used by a novel algorithm for body pose estimation. These poses provide valuable and intuitive information for interaction with human operators. Furthermore, a real-time object detection algorithm based on two-dimensional images is proposed. This algorithm is based on a cascade of different types of histograms and is able to detect objects robustly, even if large parts of the objects are occluded. Compared to existing algorithms, few training images are required to achieve good results.
Like object detection, mapping can be performed with two and three dimensions. Chapter 4 focuses on methods for vision-based mapping. Robust mapping requires information about the robots position and thus the ego-motion. As it is strongly related to stereo reconstruction, the stereo module is extended with a fast ego-motion estimation. Detecting the ground by image analysis is the only possibility to build a map based on two-dimensional images. As searching for edges is only possible in some environments, a texture-based approach is developed. The algorithm is extended with a memory to remember valid types of the ground. For building three-dimensional maps, it is adequate to merge the different point clouds obtained by the stereo modules based on the robots ego-motion. Therefore, a modular approach to the median fusion algorithm is presented, allowing the use of different stereo modules and other types of sensors. Hence, the system can easily be adapted to different sensors or machines. Additionally, the thesis introduces a novel algorithm to merge the data obtained from laser rangefinders and cameras into one representation combining the advantages of both sensors - the laser rangefinder's accuracy and the camera's color information.

Cognitive Processing

As it is easy to access and integrate both perception modules, semantic mapping is a well suited representation for a cognitive architecture. Chapter 5 introduces a novel mathematical base for a semantic map. Such a semantic map is composed of a metric map and a semantic network, where cells from the metric map can link to nodes of the semantic network and vice versa. This mathematical base can be used for the decision making process and thus to plan actions and interact with the environment. Details about the integration of the cognitive architecture into a real-time capable system are presented together with a simulator, which is developed to test the cognitive architecture. Both simulations and real world experiments will be shown to validate the whole system.

This thesis presents a wide variety of aspects contributing to the development of sophisticated robots with cognitive abilities. Some of the aspects improve existing algorithms, while others mark new methods for perception and cognition. By covering and integrating the most important facets of perception and cognition, the presented thesis serves as a base for further research.

2 Robot System Development

A sophisticated mobile robot is essential in order to test the components of a cognitive architecture during the implementation process and to verify the whole system afterwards. As some of the proposed methods, like real-time stereo vision and object recognition, are computationally expensive, modern hardware is required on the robot. Due to the strong connection between the vision system and the navigation system and the small bandwidth of a wireless network, vision processing needs to be performed on board the robot. Offline computations would furthermore reduce the robots operating range and thus the autonomy. Equally important, the mobile robot should be capable of both indoor and outdoor scenarios. Indoor scenarios provide a structured environment and controllable weather and light conditions, while outdoor scenarios provide cluttered scenes with challenging light conditions. In general, indoor scenarios are easier to handle but outdoor scenarios provide more exciting possibilities.
The remainder of this chapter is organized as follows: It starts with an introduction to the ACE robot and specifies its mission. Section 2.2 describes the architecture of the presented semantic mapping system, which was implemented using ACE as demonstrator. A detailed overview of the hardware and software architecture of ACE can be found in Appendix A.

2.1 The Autonomous City Explorer

These requirements are fulfilled by the autonomous city explorer (ACE) [12], which was co-developed at the Institute of Automatic Control Engineering within the scope of this thesis. The main mission for ACE is to find its way from the institute to the Marienplatz, a public square in the center of Munich, only by interacting with humans and without using prior map knowledge or GPS information. To fulfill this task, the robot must be able to perform vision guided dialogue-based navigation in an unknown urban outdoor environment. The robot must be able to find a human and initialize the interaction. Using a speech based dialog system, the most natural way of interaction, is impossible with the background noise at heavily frequented public places or with traffic noise. Therefore no speech-recognition system is used and the human-robot-interaction is performed via a touch screen and loudspeaker. To enhance the natural interaction, ACE has the ability to speak and to recognize human body poses. Another important ability for a robot in order to behave like a pedestrian is the robust detection of the sidewalk. ACE is not allowed to cross junctions or to drive on streets, so crossroads have to be detected reliably. To traverse crossroads safely, ACE tracks a person wearing a T-shirt with a chessboard pattern.
Outdoor experiments were conducted successfully on 30th and 31st of August, 2008 [11]. Starting from the institute, ACE succeeded in reaching the Marienplatz. The whole process took about 5 hours, while the route had a length of approximately 1.8 km, including heavily traveled roads and crowded public places. Figure 2.1 illustrates the approximate

Fig. 2.1: Approximate trajectory of ACE on its way to the Marienplatz with four exemplarily stations: (a) crossing a street, (b) and (c) interaction with a pedestrians, and finally (d) approaching the Marienplatz.

route of the ACE robot from the institute to the Marienplatz. 25 and 38 pedestrians, respectively, interacted with ACE and gave information about the direction of the destination. The relatively large number of pedestrians and long time are caused by interested pedestrians who initialized the interaction and interrupted ACE, just to see how the robot would react. An example for natural human-robot-interaction is shown in Figure 2.1 (b) and (c), where ACE is interacting with pedestrians. Figure 2.1 (d) shows ACE navigating in a highly crowded environment.

Parts of the semantic mapping framework presented in this thesis have been developed within the scope of the ACE project, namely parts of the system architecture, object detection, gesture recognition, two-dimensional mapping, and semantic labeling of places. As ACE was not allowed to cross streets autonomously, it had to stop at intersections. Therefore the robot had to be able to recognize traffic signs and traffic lights. As mentioned before, human body pose estimation was used to enhance the interaction. Therefore pedestrians were asked to point in the direction ACE had to drive. Both methods, detection of traffic signs and human body pose estimation can be referred to as object detection. The two-dimensional mapping system was used to recognize the sidewalk, in order to prevent ACE from falling down the curbs. Hence, the sidewalk was detected using texture analysis and the shape of the sidewalk was used to identify the type and thus for semantic labeling.

2.2 Software Architecture of a Semantic Mapping System

The semantic mapping system has emerged from the software architecture of ACE and makes heavy use of methods, which have originally been developed for the vision [93] and navigation systems, like the human body pose estimation, the two-dimensional mapping module, the laser rangefinder, and basic data structures like point clouds.

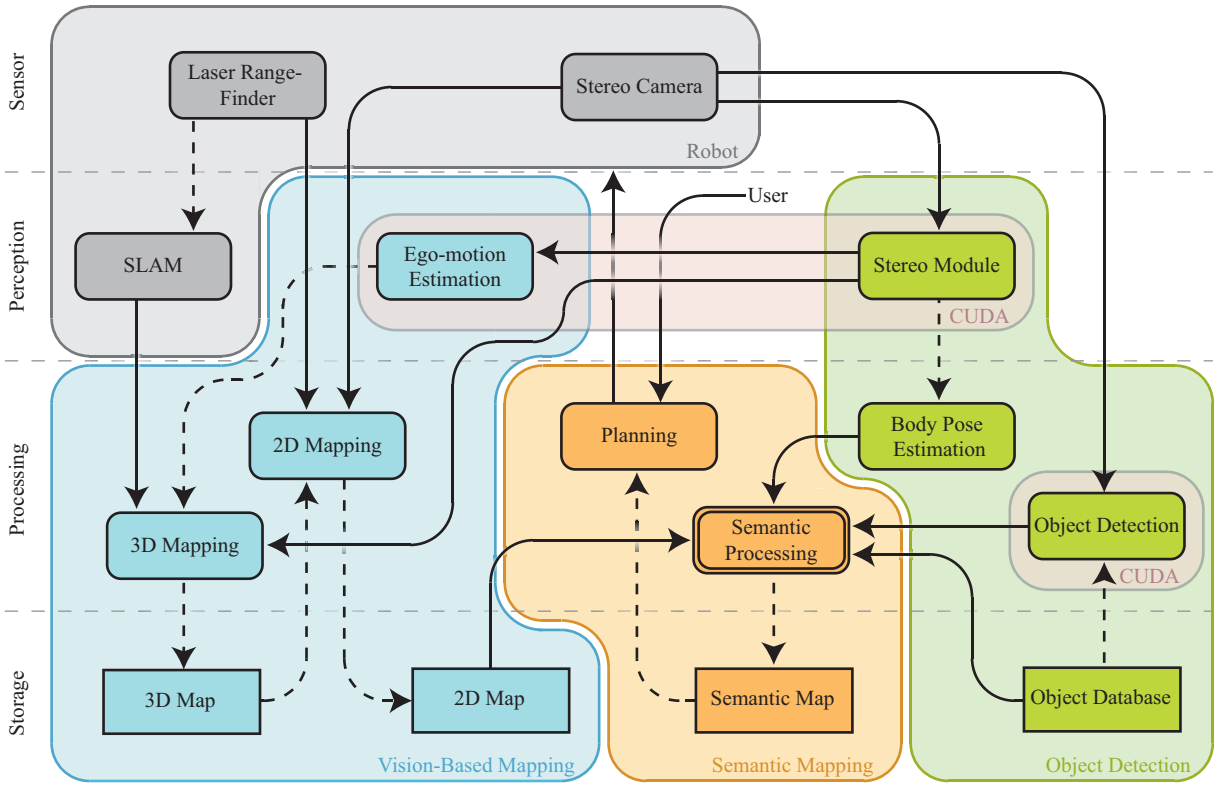


Fig. 2.2: System architecture of a semantic mapping system. Rounded boxes indicate modules, square boxes indicate data storage structures, solid arrows connections between the subsystems, dashed arrows connections between the modules in one subsystem, and dashed lines separate the different layers. The different subsystems are highlighted in different colors. The Ego-motion Estimation, the Stereo Module, and the Object Detection are executed on Cuda.

Figure 2.2 shows the system architecture of the semantic mapping system, which can be separated into four different layers and four subsystems, the *Robot*, the *Object Detection*, the *Vision-Based Mapping*, and the actual *Semantic Mapping*. Each subsystem is composed of several modules, which can be separated further into different submodules. The sensor data is gathered in the *Sensor Layer*, containing the **Stereo Camera** and a **Laser Rangefinder**. The sensor data is pre-processed in the *Perception Layer* containing a **Stereo Module** and a vision-based **Ego-motion Estimation**. Furthermore, the

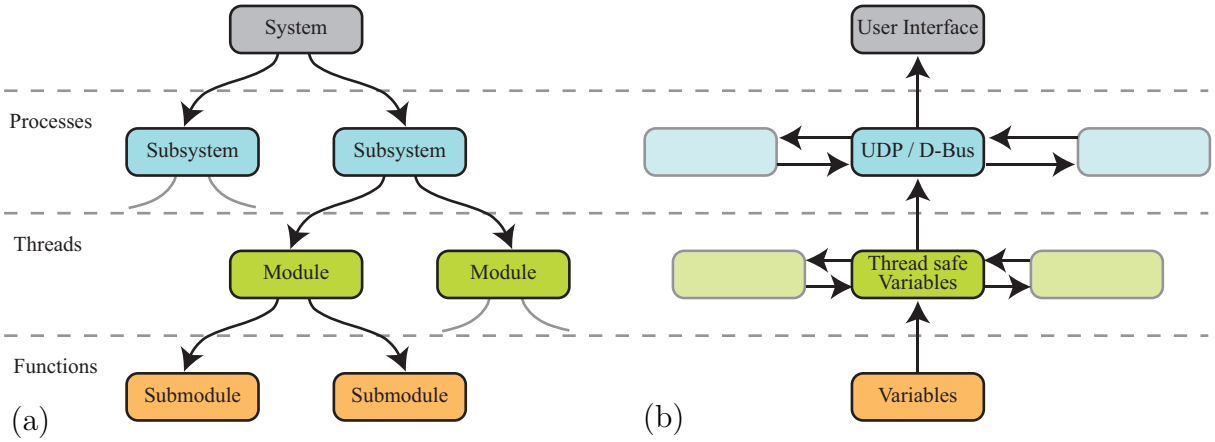


Fig. 2.3: (a) Hierarchy of the software architecture and (b) data flow with the corresponding communication protocols.

simultaneous localization and mapping (**SLAM**)-modules from the navigation system can be placed in this layer. The actual data processing is performed in the *Processing Layer* containing **2D** and **3D Mapping**, **Body Pose Estimation**, **Object Detection**, **Semantic Processing**, and **Planning**. To simplify the data exchange between the modules and subsystems, the results are stored in the *Storage Layer*, containing the data structures **2D Map**, **3D Map**, **Semantic Map**, and **Object Database**. Ego-motion estimation, stereo module, and object detection are executed on a CUDA-board, where ego-motion estimation and the stereo module are combined in one program. The object detection and vision-based mapping subsystems form the perception, while the semantic mapping subsystem marks the cognition.

Tab. 2.1: Hierarchy of the software architecture

	Instances	Protocol	System	Parent
System	1	-	Program	User
Subsystem	n	UDP / D-Bus	Process	System
Module	n	Thread safe variables	Thread	Subsystem
Submodule	n	Variables	Function	Module

The whole system is executed in different threads and processes. As illustrated in Figure 2.3, each subsystem is executed in at least one independent process containing only modules of the corresponding subsystem. Each process is separated into different threads, consisting of one or more modules. All submodules of a module have to be executed in the same thread. The only exception is made for processes executed on CUDA, where every function is called multiple times in parallel threads. More detailed information about CUDA and the integration of processes executed on CUDA can be found in Appendix B. As only read access is required, the object database can be instantiated multiple times in arbitrary

threads or processes. Whenever the object database is changed, a signal has to be sent to every thread or process and the database has to be reloaded. In addition, the processes don't have to be executed on the same machine. Hence, the computational power can be adjusted to the needs of the selected modules. The communication between the modules is accomplished with thread-safe data structures, while a communication protocol is used for the communication between the subsystems. This protocol is either based on a UDP connection when the processes are executed in a distributed system, or based on the D-Bus protocol when the system is executed on one single machine. Both communication protocols provide the same interface to the corresponding submodule, so that the protocol can be adjusted easily. Every subsystem, module, or submodule can only communicate with its parent or with other children of its parent of the same layer. Table 2.1 summarizes the system hierarchy. Further details about the implementation of the software system and the scheduling can be found in Section 5.3.2. The following sections introduce the main subsystems and give a short introduction to the most important modules.

2.2.1 Object Detection

The object detection subsystem provides different algorithms and methods to detect, locate, and identify objects in the vicinity of the robot. Furthermore, these methods can be extended to detect certain attributes of the identified objects, e.g. their color. A detailed description of the object detection subsystem, modules, and the corresponding submodules will be given in Chapter 3.

Stereo Module

The stereo module uses images obtained by the Bumblebee X3 stereo camera with a resolution of 640×480 pixels as input, where 150 disparities have to be computed. Before the stereo image can be reconstructed, both images have to be rectified to achieve epipolar geometry. To ensure real time-capability, the stereo module is implemented using CUDA, resulting in a speed of 15 Hz. The result of the computation is stored as a colored three-dimensional point cloud.

Body Pose Estimation

A point cloud obtained by the stereo module is used as input to estimate a human body pose. In the first step, possible humans are detected by using a skin color filter, followed by a segmentation of the point cloud based on the results of this detection. Now, a human model with 28 degrees of freedom is fitted into the point cloud and the body pose is estimated.

Object Detection

As this approach is based on the use of a cascade of different types of histograms, namely color histograms, histograms of oriented gradients, and color co-occurrence histograms, and thus both color and spatial information, it provides a fast and reliable method to detect previously learned objects. This method requires very few training images and is

able to deal with large occlusions. Again, it is implemented using CUDA and thus provides a real-time capable object detection.

Object Database

As mentioned above, objects have to be learnt before the object detection cascade can be performed. Due to the character of the histograms, very few training images are necessary for each object. A total of 18 images from different points of view will suffice. Threshold values for each object and each type of histogram and thus each step of the cascade are also stored in the object database and are furthermore connected to the type of the object.

2.2.2 Vision-Based Mapping

The vision-based mapping subsystem provides methods to create both two- and three-dimensional maps of the environment. As mapping requires knowledge about the robot's position, the sophisticated real-time capable stereo module is extended with a method for ego-motion estimation. A detailed description of the vision-based mapping subsystem, its modules, and the corresponding submodules will be given in Chapter 4.

Ego-motion Estimation

As both problems are somehow similar and strongly connected, the ego-motion estimation is computed concurrently with the stereo image processing, having full access to the results and temporary data. A frame rate of 10 Hz can be achieved for the combination of both algorithms.

3D Mapping

The 3D mapping module uses a modular implementation of the median fusion algorithm introduced by Nister et al. [2] to merge different point clouds obtained by the stereo module. The alignment of the different point clouds is computed based on the results of the ego-motion estimation or based on the result of the navigation's SLAM module, yielding an accurate, three-dimensional representation of the environment. Moreover, an algorithm to fuse laser data with vision data is presented.

2D Mapping

The 2D mapping module utilizes a texture-based approach to analyze two-dimensional images. Therefore the assumption that the area in front of the robot is free of obstacles, is made and the texture in front of the robot is compared to the texture in the rest of the image. By back-projecting the result of this comparison a metric map can be computed. Further obstacles are detected by analyzing the result of the 3D mapping module.

3D Map

The three-dimensional map is stored as a colored three-dimensional point cloud, where different functions can be applied on the point cloud, e.g. reduction of noise and outliers.

2D Map

An occupancy grid is used to store the two-dimensional map. Each cell represents a part of the ground with a certain size and is assigned with a probability. A high probability denotes the cell is free of obstacles, whereas a low probability denotes the cell is occupied by an obstacle.

2.2.3 Semantic Mapping

To plan actions, the robot needs to combine the results of both object detection and vision-based mapping. The semantic processing subsystem provides methods and data structures to store the gathered information in an adequate way and to draw logic conclusions. Consequently, the semantic mapping subsystem represents the cognition part of the presented cognitive architecture. A detailed description of the semantic mapping subsystem, its modules, and the corresponding submodules will be given in Chapter 5.

Semantic Processing

The results of object detection and two-dimensional mapping are combined in the semantic processing module, yielding a semantic map. Attributes of objects can be obtained by the object detection subsystem or can be predefined by a human operator. As it controls the other modules and subsystems, and processes the gathered data, this module is one of the core parts of the whole semantic mapping architecture.

Semantic Map

A semantic map is composed of an occupancy grid, containing information about obstacles and a semantic network, containing further information about the environment. Both are connected with each other. Hence, the occupancy grid contains links to nodes of the semantic net. These nodes represent objects and are again connected to cells of the occupancy grid. Thus, the position of each object can be accessed from the semantic network and the robot can obtain further information about objects in its vicinity.

Planning

Furthermore, a semantic map can be used for action and path planning. Therefore, object states are introduced. Thus, a desired states can be specified for an object and the action planner plans actions based on the information given in the semantic map to bring the object into the desired state.

This chapter gave an insight of the development of a system architecture for a mobile robot. Besides details about the hierarchy and the communication protocols, a short introduction of the actual modules forming the perception and cognition has been given. The following chapters give more in-depth information about these modules.

3 Object Detection

One of the most important cognitive skills of a robot acting in close collaboration with humans is the ability to detect and recognize objects in its environment. Great progress has been achieved in the field of object detection and object recognition during the last decades. Beginning with simple correlation functions, current hardware provides a vast amount of computational power, and thus allowing real-time implementations of complex algorithms. Two major groups of algorithms have evolved. The first group uses two-dimensional images as input, while the other group analyzes a three-dimensional representation. Those three-dimensional representations can be obtained by stereo image processing or by utilizing other sensors like laser rangefinders. While OpenCV [75] provides a wide variety of algorithms for two-dimensional image processing and SIFT [76] has evolved as a quasi-standard for object detection, there exist many approaches for three-dimensional object detection that are still subject to further research.
Most of the popular object detection algorithms use no color information, require many training images, and sometimes a complex supervised training process. Consequently, the training phase is time consuming and can hardly be automatized. On the other hand, new technologies like stream computing increase the available computational power significantly. Hence, even more complex algorithms can be used for real-time applications. This chapter presents different algorithms for object detection in two and three-dimensional representations, while the algorithms using three-dimensional representations are independent from the sensor type. As this thesis focuses on vision-based approaches, a sophisticated real-time capable implementation of a stereo image processing system is proposed. Both standard algorithms as well as novel algorithms are presented for the actual object detection. One of the novel algorithms is developed for the estimation of human body poses, but can easily be extended to recognize other skeleton based objects. This algorithm is based on the fitting of a skeleton model into a point cloud. In addition, an algorithm utilizing a cascade of different types of histograms and thus using spatial and color information is proposed. As they can be performed in real-time and deliver good hit-rates, the presented algorithms advance the state-of-the-art. Consequently, a robust object detection architecture is presented, which can furthermore be extended to recognize attributes like color. Thus, the presented object detection subsystem is an ideal base for a semantic mapping framework.
The remainder of this chapter is organized as follows: First, an overview of current object recognition algorithms is presented, followed by details about the stereo image processing algorithm. Section 3.3 presents the object recognition algorithms based on three-dimensional representations of the environment, including the human body pose estimation. Section 3.4 introduces several algorithms using two-dimensional images, starting with widely used algorithms like SIFT, a cascade of haar-like features, and the cascade of different types of histograms. The chapter concludes with experimental results and a discussion in Section 3.5 and 3.6, where the advantages and results of the different algorithms are compared.

3.1 Overview of different Object Detection Algorithms

The research field of object detection can be separated into two main challenges: two- and three-dimensional object detection. For vision-based three-dimensional object detection, a three-dimensional representation of the environment has to be created. Stereo vision is the most common approach to gain this representation. This section will give an introduction of current work in the area of stereo image processing, three-dimensional and two-dimensional object detection.

Stereo Image Processing

Most of the vision-based three-dimensional object detection systems require some sort of stereo image processing or structure from motion module. The most obvious approach is based on two images, which are obtained simultaneously by two cameras with different points of view. A three-dimensional representation can be computed by using stereo-triangulation. An overview of the most important algorithms can be found in [121]. The other group of algorithms is based on consecutive images, for instance a video stream, whereas depth information about the scene can be obtained by analyzing the camera movement [130]. Most of the current research activities in stereo vision deals with increasing the quality and is not focused on real-time capability. Klaus et al. [65] achieved the best quality, but the computation time for one image is between 14 and 22 seconds. Some research investigates different methods to increase the performance, e.g. with scanline optimization [44] or different aggregation steps [121].
The introduction of the compute unified device architecture (CUDA) by NVIDIA established a new branch in computer vision and data processing. In contrast to a conventional CPU, stream computing allows massive parallel processing of simple functions and provides an easy to use and intuitive framework for the development of new functions. Due to its parallel nature, stereo matching is well suited for the execution on a GPU. Wang et al. presented a GPU implementation with a computation speed of 12 Hz [154]. However, their algorithm only computes 15 disparities and is thus not eligible for application on a mobile robot. Consequently, an algorithm which is able to perform stereo reconstruction in both real-time and high quality is proposed.

Three-Dimensional Object Detection

Compared to two-dimensional images, the third dimension provides valuable additional information. On the contrary, using three-dimensional point clouds as input for object detection brings larger computational challenges. Hence, suitable algorithms that are real-time capable have to be developed. Three-dimensional object detection can be separated into two fundamental problems: modeling of the object and recognition of the objects. An introduction into these problems can be found in [109]. Funkhouser et al. propose an algorithm for retrieving similar shapes from a database containing objects [38]. CAD models can be converted into point clouds, which are then used for matching [160]. Schnabel et al. presented a complete framework for rapid object detection based on point clouds [123]. In this approach the point clouds are described by basic shapes acting as features, such

as planes and cylinders. Hiroshi et al. [52] represents the object as a manifold in a low-dimensional subspace by compressing the image set in the parametric eigenspace and tries to identify the object based on this manifold. Schneiderman et al. [124] present a statistical method for the detection of faces and cars, based on a histogram containing subsets of wavelet coefficients and their positions on the object.
Developed for realistic animation of the human body in Hollywood movies, the first applications for the reconstruction of human body poses have been motion capture methods. An actor was equipped with markers, so his movements could easily be recorded. On the other hand, markers are ineligible for outdoor scenarios. A mobile robot communicating with pedestrians must use available sensors, such as cameras or laser rangefinders. The field of camera-based body pose estimation can be divided into various approaches, which are summarized below.
Some approaches use monocular vision systems [105], where an image of the body with several landmarks serves as input, shape descriptors that are extracted from image silhouettes [31, 89], or probabilistic models [18]. Semi-supervised learning can be used to increase robustness [7]. Other approaches require prior knowledge about the human movements [58], which can be stored in a motion library [106]. Boulay et al. [16] maps typical postures into two-dimensional images. Other approaches, like the one presented in this thesis, focus on multi-view systems, such as stereo vision, which provide additional depth information. The fitting of the model can be described as an optimization problem [3] and can easily be combined with tracking people [66, 107]. A learning algorithm can be used [159], where training data is recursively classified into several clusters with silhouette and depth images. Another approach is the use of an image stream, where features can be tracked between the images [80]. Through the two-dimensional tracking of the features, their three-dimensional positions can be computed [94]. Bregler et al. [19] uses twists and exponential maps to recover high degree-of-freedom articulated configurations of the human body. Of course, the image stream can consist of monocular or stereo images. Gavrilla et al. [41] use an image stream with multiple views to fit a human model into the recovered scene, while others use three-dimensional voxel data created from multiple views [118]. However, the mentioned algorithms are limited to a certain type of sensor and can only estimate human body poses. This chapter presents an algorithm for human body pose estimation, which is independent from the sensor type and can be extended to other type of objects. Every object, which can be described by a skeleton, can be detected by the algorithm.

Two-Dimensional Object Detection

Besides object detection and recognition in three dimensions, there exist several approaches based on two-dimensional images. One of the most popular and robust algorithms is SIFT (scale invariant feature transformation) introduced by Lowe [76]. SIFT uses local image keypoints that can be used to describe objects. Real-time performance can be achieved using modern hardware in combination with small datasets. Several implementations on a GPU achieved real-time capability with larger datasets [24, 71]. A different approach was proposed by Viola and Jones [152]. They used a cascade of very simple haar-like [75] classifiers. An object is detected when it passes the whole cascade. If it fails one classifier

of the cascade, the remaining classifiers will not be computed. By using simple features and many stages, this algorithm delivers fast and reliable results. OpenCV offers an easy to use implementation of this algorithm. Some research focuses on robust matching algorithms [157] or face detection, which is also a part of object detection [50]. Krawiec et al. [68] use genetic algorithms for image analysis. The work of Redfield et al. [114] utilizes color information for object detection. With fixed illumination conditions only 16 colors are sufficient for object detection. In all those approaches no spatial information is used, ignoring valuable information. On the other hand, some approaches use only spatial information, e.g. by creating histograms of oriented gradients [29], or by analyzing multiscale affine invariant image regions [163]. Chang et al. proposed color co-occurrence histograms [8, 23] combining both color and spatial information, resulting in a classifier that is robust to scale, illumination and occlusions. However, color co-occurrence histograms are computationally expensive.
As they are based on a collection of relatively simple algorithms, which have to be computed many times, many object detection algorithms are eligible for parallel computing using stream processing. Even complex algorithms like the computation of color co-occurrence histograms can be performed almost in real-time. Consequently, a novel implementation using Cuda is introduced. To increase the computational speed even further, a cascade based on three classifiers is proposed, namely color histograms, histograms of oriented gradients and color co-occurrence histograms. By using different types of histograms, spatial and color information of the objects can be used. Only very few reference images of an object are required for training. Depending on the complexity of the object, not more than 18 images from different points of view are sufficient. Consequently, online training can be easily implemented. In order to achieve independence from changes in illumination, the HSV (hue, saturation and value) color space is used.

3.2 Stereo Image Processing

Vision-based three-dimensional object detection requires a high-quality stereo reconstruction module. For the usage on a mobile robot, this stereo module has to be real-time capable. By using stream computing on a GPU, both objectives can be fulfilled. This section starts with a short introduction to stereo reconstruction and introduces methods, which are well suited for an implementation on Cuda [153]. The major novelty of the presented methods is the implementation on a GPU and the thorough adaption of the parameters.
Stereo reconstruction algorithms are based on the idea that a point $\mathbf{p}$ of the environment is mapped unambiguously on two different image planes. Assuming that the two points $\mathbf{p}_L$ and $\mathbf{p}_R$ of the left I^L and right I^R input image planes are known, the position of the corresponding point $\mathbf{p}$ can be computed. Hence, the problem of stereo reconstruction is reduced to the search for the two corresponding points. To reduce the computational complexity, the images are rectified in a first step, ensuring epipolar geometry. As shown in Figure 3.1, all possible reconstructed points $\mathbf{p}$ of the point $\mathbf{p}_L$ and thus the corresponding image points $\mathbf{p}_R$ are aligned along one line. Consequently, it is adequate to search along this line and the two-dimensional search problem can thus be reduced to an one-dimensional search

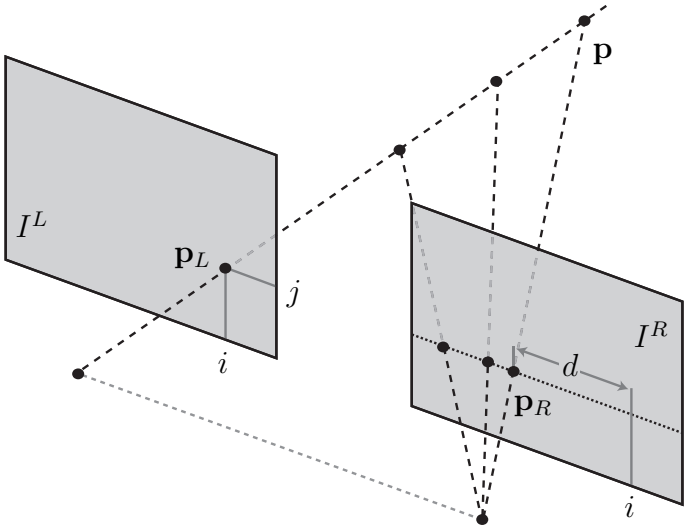


Fig. 3.1: Geometry of a stereo system.

problem. The corresponding image points in two rectified images are denoted as $I^L(i, j)$ for $\mathbf{p}_L$ and $I^R(i + d, j)$ for $\mathbf{p}_R$.
Hence, the main problem of stereo reconstruction can be stripped down to the so called correspondence problem. As a simple search function is applied to every pixel of one image and the function calls are independent, the correspondence problem can be parallelized easily. Recent research has shown that the stereo correspondence problem can be divided into four steps:

- The computation of the matching costs, where the costs of the different disparities are computed,
- the aggregation of the costs, where the different costs are improved,
- the computation of the actual disparity, and
- the post-processing, which improves the quality of the estimated disparities.

These steps will be described in the following sections. The post-processing of the stereo algorithm and the pre-processing of the object detection module are identical and will be described in Section 3.3.1.

3.2.1 Computation of Matching Costs

Due to the epipolar geometry, the corresponding pixel of the pixel $I^L(i, j)$ is placed on the horizontal line $I^R(i + d, j)$, with $d \geq 0$. Hence, the stereo algorithm scans along the corresponding horizontal line on the right image, compares the color information of the two pixels, and computes a cost function for each pixel. The pixel with the lowest cost is then selected and the distance d between the two pixels is selected as the resulting disparity. The disparity is also sometimes called the inverse depth, as a small d indicates a distant point, while a large d indicates a near point. Two algorithms for the computation of the

cost have evolved: pixel matching or block matching. While block matching shows better results, pixel matching is faster to execute. However, the aggregation step can compensate the disadvantage of pixel matching while keeping its speed. Pixel matching is therefore used for the estimation of the costs, which can be computed by using the sum of squared differences. Two different costs are computed for each pixel: intensity costs $C_I(i,j,d)$ and gradient costs $C_G(i,j,d)$:

$$C_I(i,j,d) = \sum_{c=\mathrm{R,G,B}} (I^L(i,j) - I^R(i+d,j))^2 \tag{3.1}$$

$$C_G(i,j,d) = \sum_{c=\mathrm{R,G,B}} \left(\frac{\partial I_x^L(i,j)}{\partial i} - \frac{\partial I_x^R(i+d,j)}{\partial i} \right)^2 + \left(\frac{\partial I_y^L(i,j)}{\partial i} - \frac{\partial I_y^R(i+d,j)}{\partial i} \right)^2, \tag{3.2}$$

while I_x and I_y denotes the gradients in x- and y-direction, respectively. The result is stored in a three-dimensional structure, the disparity-space image (DSI) with the dimensions i, j, and d. An element of the DSI can be computed as:

$$C_0(i,j,d) = w \cdot C_I(i,j,d) + \frac{1-w}{2}(C_G(i,j,d)), \tag{3.3}$$

while w denotes the weight of the intensity with respect to the gradients. Furthermore, outliers can be avoided by limiting each element $C_0(i,j,d)$ to a maximal value C_{Max}.

3.2.2 Aggregation of Costs

During the computation of the costs, each element $C_0(i,j,d)$ is not influenced by its neighbors. However, this would ignore valuable information. This disadvantage can be compensated by modifying each element. Therefore the surrounding elements of the DSI are examined in the aggregation of costs. There exist several different methods for the computation of the update value [121], which will be introduced blow. Some of these methods are shown in Figure 3.2 (b)-(d), while Figure 3.2 (a) shows the corresponding ij-plane from the DSI.

Square Window

The simplest approach is the computation of the average value over a local squared window with a fixed size of $N \times N$, with $N = 2m + 1$:

$$C_A(i,j,d) = \frac{1}{N^2} \sum_{k=-m}^{m} \sum_{l=-m}^{m} C_0(i+k, j+l, d), \tag{3.4}$$

while $C_A(i,j,d)$ indicates the updated element of the DSI. By computing the average value of each line and then computing the average of these averages, the whole computation can be separated in a horizontal and a vertical part and the computation speed can be increased. This reduces the complexity from $O(n^2)$ to $O(2n)$.

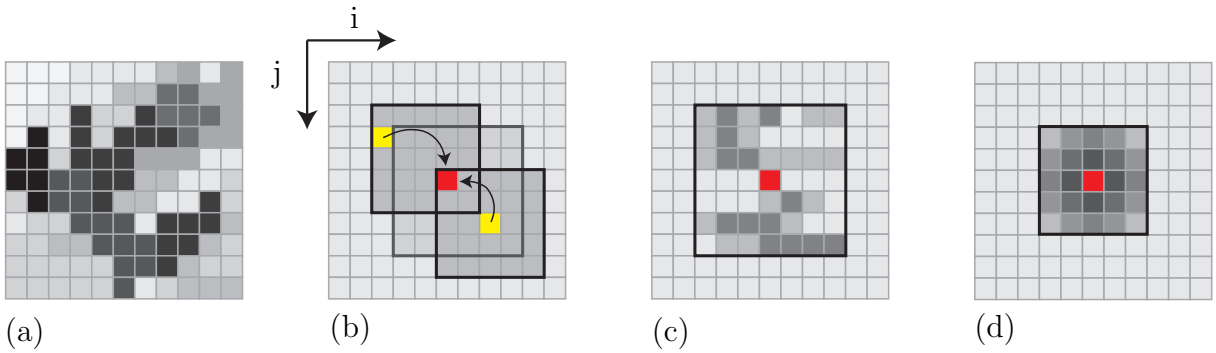


Fig. 3.2: Comparison of the different aggregation methods: (a) the ij-plane from the DSI, (b) the minimum filter approach, (c) the boundary guided window approach, and (d) the adaptive weight window approach. Darker colors of the pixels indicate a higher weight, while the red pixel indicates the current element $C_0(i, j, d)$.

Minimum Filter

Another modification is the minimum filter, which is based on the shiftable window approach presented by Gong et al. [44, 87]. Before the minimum filter can be applied, a square window has to be computed for the whole DSI. The actual minimum filter approach moves a support window of size N_{Min}, with $N_{\text{Min}} \leq N_{\text{Square}}$, within the range $i - m \ldots i + m$ and $j - m \ldots j + m$, respectively. $C_A(i, j, d)$ is then replaced by the element within all support windows, which has the lowest cost. Figure 3.2(b) shows two different support window positions. The yellow pixel indicates the element with the lowest cost of the shown support window, which will be used to replace the current element.

Adaptive Window

The adaptive window approach uses the minimum filter approach to estimate the window size for a square or an adaptive weight window. Therefore a minimum filter or the squared window approach is applied with several different window sizes yielding different possible results. By computing the average value, the minimal or the maximal value of one of these possible results is then selected as the final result. Despite the presented advantages, the squared window approach or the minimum filter approach have to be applied multiple times yielding a larger computation time. This results in the real-time incapability of the algorithm for the used hardware architecture.

Boundary Guided Window

Another approach is the utilization of a corner detection filter to include only those pixels, which are inside the boundary. As shown in Figure 3.2(d), this is realized by using different weighting factors that are computed based on the position of the pixel within the boundary. A detailed description of this computation and the different cases can be found in [87].

Adaptive Weight Window

The adaptive weight window is a modification of the square window approach, which uses different weights for the elements of the sum. As illustrated in Figure 3.2(d), pixels with a small distance to the center have a stronger weight $w(k,l)$:

$$w(k,l) = k \cdot \exp -(\frac{\Delta c_{kl}}{\gamma_c} + \frac{\Delta g_{kl}}{\gamma_p}), \tag{3.5}$$

while Δc_{kl} denotes the distance from the pixel to center, Δg_{kl} the difference in the color. γ_c, γ_g and k are weighting factors that have to be estimated heuristically.

3.2.3 Computation of the Disparity

After the costs have been computed and optimized in the aggregation step, the best disparity has to be selected. The simplest and most straightforward approach is the winner takes it all method, which selects the disparity with the lowest cost. This method is fast to execute, but as each element is estimated independently, not robust with respect to noise and outliers.
Scanline optimization computes a cost function for each line of the image. Hence, the algorithms selects an *id*-plane in the DSI and searches for a cost optimal path through this plane, where smoothness is considered as cost. This algorithm is slower but yields smoother results. However, only one line is considered and fragments between different lines can occur. Other possibilities for optimization include a median filter applied to the result of the winner takes it all method and algorithms for the reduction of noise and outliers. More details about these optimization algorithms can be found in [121] and Section 3.3.1.
Areas with low texture cannot be reconstructed by the algorithm, so they have to be identified and eliminated. Therefore a confidence measurement ζ is proposed:

$$\zeta = \frac{C_{\text{Max}} - \underset{d}{\text{argmin}}(C_A(i,j,d))}{C_{\text{Max}}}, \tag{3.6}$$

while the costs $C_A(i,j,d)$ have to be normalized. A suitable threshold ζ_{Min} has to be selected heuristically and all pixels with a confidence below the threshold are eliminated.

The presented stereo algorithm extends the state-of-the-art with a real-time capable implementation, allowing the reconstruction of images with a resolution of 640×480 pixels and 150 disparities. It provides an ideal base for further three-dimensional object detection algorithms, which will be presented in the following section.

3.3 Three-Dimensional Object Detection

Compared to flat images, the third dimension of point clouds provides valuable additional information, allowing easy access to spatial information. The presented algorithms are not only limited to point clouds obtained by stereo image processing, but also to those created by other sensors like laser rangefinders. Most three-dimensional object detection algorithms are based on the same basic steps:

- During the **pre-processing** noise and outliers are reduced,
- followed by the **segmentation**, where the point cloud is separated into spatial adjacent structures.
- Now, symbolic and numeric **attributes are extracted**, and
- then used for the actual **classification** process.

The following sections introduce these basic steps, followed by an algorithm to detect human body poses.

3.3.1 Pre-Processing

Depending on the further processing steps, different filters are applied during the pre-processing. Possible filters include reduction of noise and outliers, smoothing, and interpolation of missing points. Furthermore, the number of points contained in the point cloud can be reduced or increased. Thus, a desired resolution of a point cloud can be adjusted. This section gives an overview of the most important filters.

Reduction of Noise and Outliers

Due to the trade-off between quality, speed, and reconstruction errors in the stereo module, most of the point clouds are covered with noise and show outliers. Areas with low textures or overexposed areas are difficult to match during the stereo matching and lead to inevitable reconstruction errors. Hence, noise and outliers have to be identified and removed in the pre-processing step. The proposed algorithm is based on a nearest neighbor search and is relatively simple and thus fast to execute. Most of the execution time is spent searching for the nearest neighbor. Fortunately, the nearest neighbor problem is well explored. One of the most efficient approaches is the use of kd-trees [5], a space partitioning data structure. Kd-trees use splitting planes parallel to the axis to separate the space into partitions, each containing one point. Consequently, a nearest neighbor search algorithm can address directly the space and not all the distances to all points have to be computed. As outliers and noisy points are of similar topology, the reduction of both can be performed in one step. Algorithm 1 shows the developed algorithm for noise reduction of a point cloud $\mathbb{N}$. $\mathbf{x}_i$ denotes a single point and the function kd-search$(\mathbf{x}_i, R)$ returns the number of neighboring points within a certain search radius R. If this number is below the minimal number of neighbors N_{Min}, $\mathbf{x}_i$ will be removed from the point cloud.

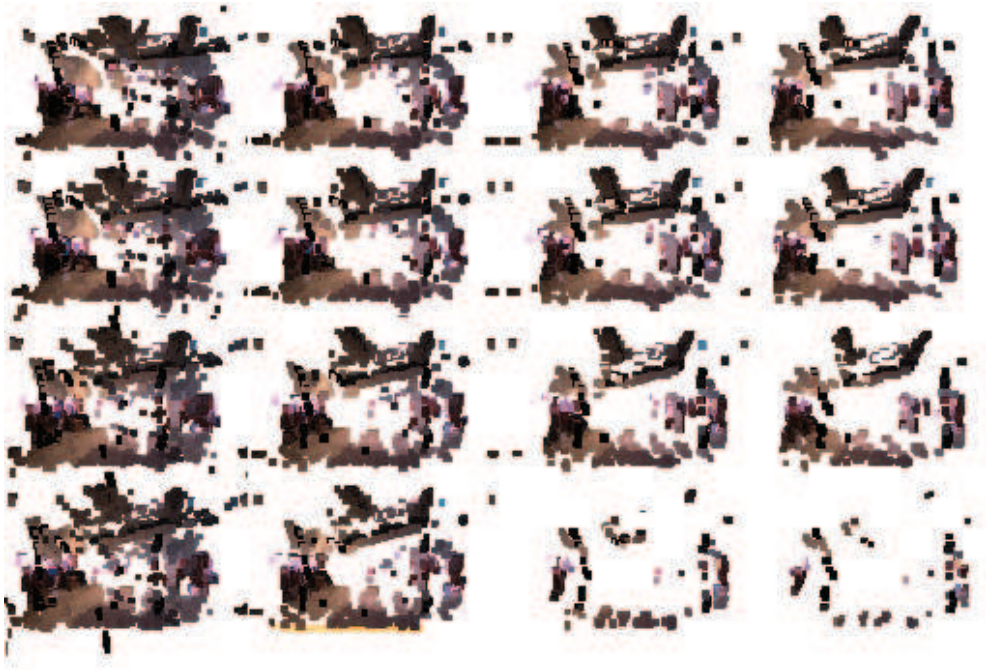


Fig. 3.3: Result of the reduction algorithm with different parameters. From left to right: decreasing search radius R, from bottom to top: increasing minimal number of neighbors N_{Min}.

Algorithm 3.1 Reduction of outliers and noise

```
Define search radius R and minimal number of neighbors N_Min
Build kd-tree
for all Points x_i of the point cloud ℕ do
    N = kd-search(x_i, R)
    if N < N_Min then
        Remove x_i from ℕ
    end if
end for
```

Figure 3.3 shows the result of the reduction algorithm with different parameter settings. Both the minimal number of neighboring points N_{Min} and the search radius R have to be selected carefully and are depending on the desired application, the resolution and quality of the used stereo algorithm, and thus on the density of the input point cloud.

Smoothing and Interpolation

Compared to their horizontal and vertical resolution (640×480 to 1280×960 pixels), most stereo algorithms provide a poor depth resolution (15 to 150 disparities). Hence, a reconstructed point cloud seems to be vertically sliced, leading to problems with three-dimensional object detection algorithms. In addition, the horizontal and vertical resolution decreases with increasing distance to the camera. Furthermore, most stereo algorithms are not able to reconstruct areas with low texture, leading to more missing points in the

reconstructed point cloud. Scaling the size of the point cloud will lead to a different density of points and thus to further missing points. Consequently, there is a need for an algorithm that is able to reconstruct the missing points and normally distribute the resolution of the point cloud.
The most obvious approach is an interpolation between existing points. The proposed interpolation algorithm is working in close proximity to the existing points and can be thus considered as a local algorithm. In a first step, the point cloud is quantized with the desired resolution, so that missing points can be identified. Next, the nearest neighbors of the missing points are computed. By weighting the neighbors with an inverse euclidean distance, the color and exact position of the new point can be computed. If the distances to the neighbors are too large, the point cannot be reconstructed. This novel algorithm provides a fast method to adjust the resolution of a point cloud with a concurrent smoothing process. Another method utilizes polygons to reconstruct point clouds, where existing points are used to compute a triangle mesh around the object. Missing points will lead to holes in the mesh, which have to be detected and closed.
Smoothing of a point cloud can be achieved by performing a three-dimensional discrete convolution with an adequate kernel function $\mathbf{K}$:

$$\mathbb{N}' = \mathbb{N} * \mathbf{K}. \tag{3.7}$$

Distribution functions with an integral of 1, like the gaussian filter, are well suited for smoothing:

$$h(x, y, z) = \frac{1}{\sqrt{2\sigma^2\pi}} \exp - \frac{x^2 + y^2 + z^2}{2\sigma^2}. \tag{3.8}$$

The pulse response h of the filter can be stored in a three-dimensional array:

$$\mathbf{K}(x, y, 1) = \begin{bmatrix} 0 & \frac{1}{40} & 0 \\ \frac{1}{40} & \frac{3}{40} & \frac{1}{40} \\ 0 & \frac{1}{40} & 0 \end{bmatrix}, \mathbf{K}(x, y, 2) = \begin{bmatrix} \frac{1}{40} & \frac{3}{40} & \frac{1}{40} \\ \frac{3}{40} & \frac{10}{40} & \frac{3}{40} \\ \frac{1}{40} & \frac{3}{40} & \frac{1}{40} \end{bmatrix}, \mathbf{K}(x, y, 3) = \begin{bmatrix} 0 & \frac{1}{40} & 0 \\ \frac{1}{40} & \frac{3}{40} & \frac{1}{40} \\ 0 & \frac{1}{40} & 0 \end{bmatrix}. \tag{3.9}$$

Other kernel functions, like the Laplace filter, can be used to detect edges. Depending on the further processing steps and the desired spatial attributes, it can be useful to apply a combination of several filter operations.

3.3.2 Segmentation of Three-Dimensional Point Clouds

The three-dimensional representation of the scene acquired by the stereo vision module contains a large colored point cloud consisting of several different objects. By using a segmentation algorithm, the different objects can be separated. As this algorithm is mainly used for human body pose estimation, objects that may be considered as humans have to be found. A skin color detector is used to detect parts of the point cloud that may belong to a human body. Consequently, the remaining parts of the human body have to be found. Beginning at the detected start point, the segmentation algorithm searches for locally ad-

(a) (b) (c)

Fig. 3.4: Example of the Segmentation Process: (a) Image with detected skin parts as start point for the segmentation process. This image has been obtained by a stereo camera with a high aperture and has already been rectified. (b) shows the three-dimensional reconstruction of the scene using a stereo matching algorithm and (c) the detected segments in the scene.

jacent structures and will create one cluster for each detected point. Needless to say that other start points have to be selected for the detection of other objects. Edge detection filters, saliency maps, two-dimensional attributes, or the result of an two-dimensional object detection algorithm can be used as such start points.
Algorithm 3.2 shows the algorithm used for the segmentation. Starting with a given point $\mathbf{x}$, all neighbors $\mathbb{N}^{\mathbf{x}} = \mathbf{x}_n \dots \mathbf{x}_m$ are examined. A neighbor $\mathbf{x}_n$ will be included in the cluster, if the distance $d = ||\mathbf{x} - \mathbf{x}_n||^2$ is smaller than a certain threshold d_{Max}. The point cloud $\mathbb{N}^{\mathbf{x}}$ includes all points fulfilling this condition. $\mathbb{N}^{\mathbf{x}}[i]$ accesses the i-th point of this point cloud. In the next step, all neighbors of the next point in the cluster are considered. This function is repeated, until no valid neighbors are found or all points in the cluster have been visited. As the main goal is the detection of humans, the algorithm adds only those points to the cluster, which are included in a cylinder describing the area that can be reached by a human.
Figure 3.4 (c) shows the detected segments of the scene with the result of the skin color detection (see Figure 3.4 (a)) as start points, while Figure 3.4 (b) shows the three-dimensional reconstruction of the same scene. After pre-processing and segmenting the point cloud, the actual object detection algorithm can be applied.

3.3.3 Object Detection using Local Attributes

After the point cloud has been pre-processed and segmented, each segment represents a possible object that has to be verified and later identified. The actual object detection process can be separated into two major steps: The extraction of the attributes and the classification.

Algorithm 3.2 Segmentation of three-dimensional point clouds

1: $\mathbf{x}$ = point, corresponding to the detected skin part
2: $i = 0$
3: **repeat**
4: Detect neighbors $\mathbb{N}^{\mathbf{x}}$
5: **for** $j = 1$ to number of neighbors **do**
6: $d_j = ||\mathbf{x} - \mathbf{x}_j||$
7: **if** $d_i < d_{\text{Max}}$ **then**
8: add $\mathbf{x}_i$ to cluster
9: **end if**
10: **end for**
11: increase i
12: $\mathbf{x}$ = next point in cluster
13: **until** $i < i_{\text{Max}}$

Extraction of Attributes

There exists a wide variety of different symbolic attributes, which can be distinguished into several types: *spatial*, *topological*, and *color* attributes. Spatial attributes contain the length, width, height, area, and the volume of the object, while topological attributes contain the shape and the centers of mass and area. Color attributes contain the average color in the red, green and blue color channel and the color distribution, which can be estimated by using color histograms. An object can be unambiguously described by a suitable n-dimensional vector $\mathbf{b}$ containing n attributes $b_0, \dots b_{n-1}$.

Classification

Reference objects are used to compute a set of m reference vectors $\mathbf{B}^R$, containing the selection of the attributes. $\mathbf{B}^R$ is used to estimate the type of the segment by comparison to the object's attribute vector $\mathbf{b}$, which has to contain the same attributes. Most of these methods are based on a distance function. For instance, an euclidean distance is computed between the object's vector $\mathbf{b}$ and each reference vector $\mathbf{b}_i^R$, with $0 \leq i < m$. Using the nearest neighbor is the simplest way to select a class. However, this will always yield a result and thus a high probability of false positives. This can be solved by assigning a probability to each distance and by selecting the class with the highest probability only, if this probability exceeds a certain threshold, which has to be determined heuristically.
Support vector machines (SVM) use kernel functions to transform the problem into another space, where two different classes can be separated linearly. There exists two different ways of classification with support vector machines: using SVMs to distinguish between a reference class $\mathbf{B}^R$ and one class containing all remaining classes, or to compute all possible combination of classes and use the SVM on every pair of reference classes. The first method requires $n - 1$ comparisons for n classes and the second possibility $(n - 1)^2$ comparisons. The second methods yields a higher accuracy. Due to the large number of comparisons, SVMs are computationally expensive and thus impractical in handling a large number of objects. Furthermore, the kernel function has to be selected carefully.

3.3.4 Human Body Pose Estimation

The estimation of human body poses is a special case of three-dimensional object detection. In contrast to existing approaches, the algorithm proposed in this section tries to fit a human model into a point cloud instead of using features [91]. A skin color filter is used to compute start points for the segmentation in order to find possible humans. A description of the algorithm used to detect the skin color segments can be found in [136]. As the skin detector may detect false positives and two or more skin parts of the same human, the clusters are validated based on different attributes. One of the key advantages of the algorithm is the expandability, which is achieved by the possibility to replace the model used for fitting. Hence, any object that can be described by a skeleton with different poses can be found and the pose can be estimated. These objects include obvious ones like animals or tables and chairs, but also more complex ones like biped humanoid robots or larger machines like cranes or excavators. Eventually, the pose of a manipulator with many degrees of freedoms could be estimated. Accordingly, an eligible algorithm to find a suitable start point has to be found. As mentioned before, a color filter may not be practical for all models and may be replaced with a two-dimensional object detection algorithm.

Validation of the Segment

After the clusters have been created, they have to be validated. To discard invalid clusters, the following filters are applied:

- The number of points in the segment is taken into account. If the number of points is too small, the cluster is most likely caused by a false positive.
- If too many points have been found, the skin detector has found something else than a human.
- The fitting of the model into the cluster is starting with the head, so only the clusters are valid, where the detected skin part matches the head. To check this, the centroid of the cluster is computed and compared with the start point.
- Large clusters with a low density of points are most likely false positives, so they will be rejected.

The numbers and thresholds in these filters are depending on the model and have to be estimated during the creation process of the model. After the application of these filters, almost all invalid clusters will be rejected. The remaining invalid guesses will be removed in the next step, when no valid body pose can be found. If another skeleton is used, these filters have to be adjusted to the new model.

Human Model

Figure 3.5 (a) shows a schematic view of the human model with 15 links and 28 degrees of freedom, respectively. Each link provides one, two, or three degrees of freedom and is rotated around the axis of the coordinate system of the link it is connected to. Table 3.1 shows the links, the number of degrees of freedom, the hierarchical structure, and the

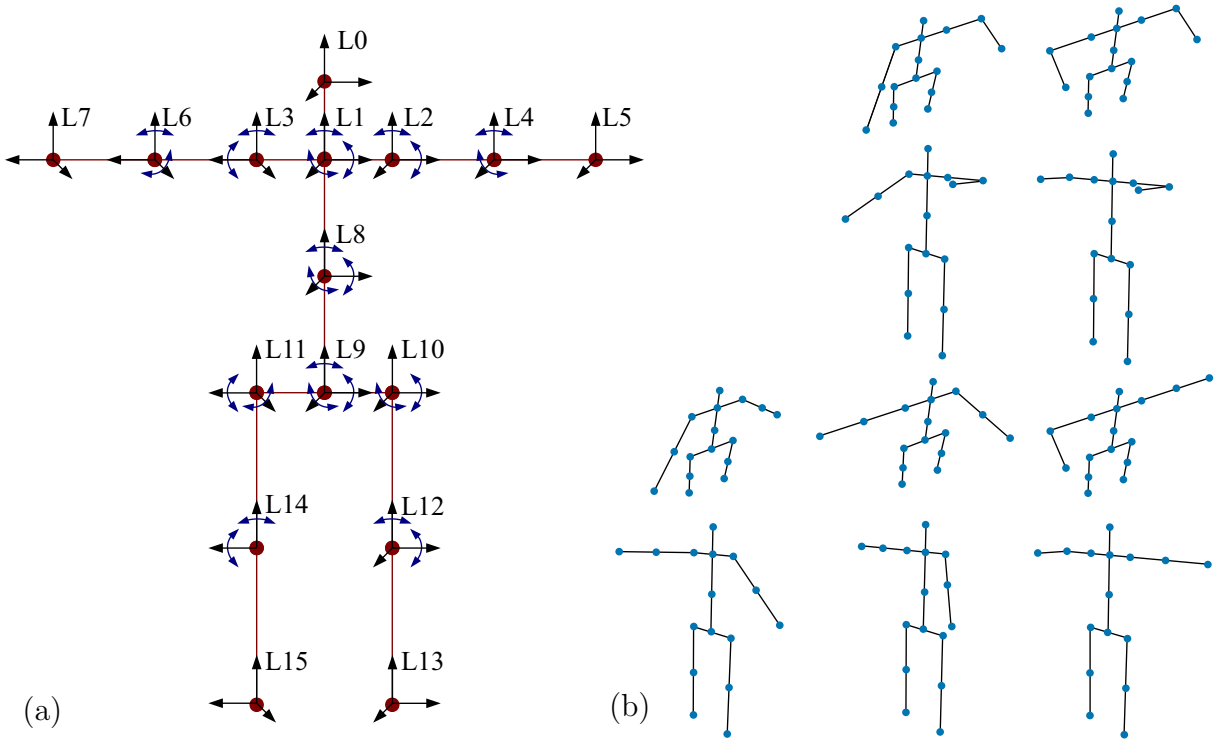


Fig. 3.5: (a) Reduced human model with 15 links, the corresponding coordinate systems and the degrees of freedom and (b) five typical body poses of the reduced human model from different points of view.

length of the link. For example, the lower left arm (link 4) will be rotated around the coordinate system of the upper left arm (link 2) and both the left and the right shoulder (link 2 and 3), are rotated around the neck (link 1). As the hands and the feet are too small to be detected robustly by the stereo matching, they have not been taken into account. Hence, this model is reduced by 10 degrees of freedom. To increase the accuracy of the estimated body pose, 27 typical poses are considered. As the configuration of the arms is of particular interest for interaction, the relevant poses differ mostly in the arms. To be able to recognize the pose from every point of view, the whole body model is rotated in steps of 33.3°. Figure 3.5 (b) shows some of the different poses.
To validate the human model, the link lengths and the angles between the links are considered. A minimal and a maximal value for each parameter of each link has been estimated. As they are coupled, the left and right shoulder are treated specially. The left shoulder has three degrees of freedom and whenever it is moved, the right shoulder is limited in its movement and will show a similar movement. Consequently, Table 3.1 shows no degrees of freedom for the right shoulder.

Extraction of the Body Pose

After one cluster has been computed for each human in a scene, they can be used to extract the body poses. The algorithm will be executed once for each cluster, so the accurate number of humans can be found. For each of the possible humans, the algorithm

Tab. 3.1: Links of the human model

Link	Name	Connected to Link	DOF	Length (in m)
0	Start (Head)	-	0	-
1	Neck	0	0	0.25
2	Shoulder Left	1	3	0.25
3	Shoulder Right	2	0	0.25
4	Upper Arm Left	2	2	0.375
5	Lower Arm Left	4	2	0.375
6	Upper Arm Right	2	2	0.375
7	Lower Arm Right	6	2	0.375
8	Upper Back	1	3	0.5
9	Lower Back	8	3	0.5
10	Hip Left	9	3	0.25
11	Hip Right	9	0	0.25
12	Upper Leg Left	10	2	0.5
13	Lower Leg Left	12	2	0.5
14	Upper Leg Right	11	2	0.5
15	Lower Leg Right	14	2	0.5

tries to fit all 27 typical body poses and computes an error metric for each pose. The pose with the lowest error will be selected as winner. As the algorithm should be able to deal with all different types of colors and clothes, color provides little useful information and is consequently not used. Both segmentation and estimation of the body pose are only based on the position of the points. First, the actual fitting method will be described, followed by the error metric and the validation of the estimated body poses.

Method for Fitting a Body Pose

Starting with the head, the algorithm tries to fit the attached links iteratively. The order of the links is the same as described in Table 3.1. If a link is not part of the image or is occluded by an object in front of it, no valid fit can be made and the link and all links attached to it will not be included in the resulting human model. The links are fitted based on the following method: start point of a link will be the end point of the previous link. Based on the model, the algorithm knows, where the end of the link should be placed in an ideal case, namely at the so-called reference point $\mathbf{p}_r$. In a real case, the end of the link will be placed somewhere near this reference point. The algorithm will search the points $\mathbb{N}^{\mathbf{p}_r}$ around the reference point $\mathbf{p}_r$ for possible ends of the link. An end point can only be valid, if the restrictions of the link (e.g. link length or the angles between the previous links) are not violated. In addition, the point density of the cluster along the link is not allowed to fall below a certain limit. After the link has been fitted, the error metric will be updated and the next link can be computed. This will be repeated iteratively until all links are fitted.

Algorithm 3.3 Fitting a reference body pose

(a) Function findBestFit($\mathbf{p}_r$):

1: $\mathbb{N}^p$ = points near($\mathbf{p}_r$)
2: **for** $i = 1$ to number of points in $\mathbb{N}^\mathbf{p}$ **do**
3: $\quad\mathbf{p}_t = \mathbb{N}^\mathbf{p}[i]$
4: $\quad e_1$ = compute point density near link
5: $\quad e_2$ = compute link restrictions
6: $\quad\mathbf{e}_t[i] = e_1 + e_2$
7: **end for**
8: m = compute best $\mathbf{e}_t$
9: return $\mathbb{N}^\mathbf{p}[m]$

(b) Main Algorithm:

1: $\mathbb{N}_r$ = Get Endpoints for Pose n
2: $e_l = 0$
3: **for** $i = 1$ to number of links **do**
4: $\quad$**if** $\mathbb{N}_s[i-1]$!= invalid **then**
5: $\quad\quad\mathbf{p}_r = \mathbb{N}_s[i-1] + \mathbb{N}_r[i]$
6: $\quad\quad\mathbf{p}_s$ = find best fit($\mathbf{p}_r$)
7: $\quad\quad$**if** $\mathbf{p}_s$ is valid **then**
8: $\quad\quad\quad\mathbb{N}_s[i] = \mathbf{p}_s$
9: $\quad\quad$**else**
10: $\quad\quad\quad\mathbb{N}_s[i]$ = invalid
11: $\quad\quad$**end if**
12: $\quad\quad e_i$ = compute error metric($\mathbf{p}_r$)
13: $\quad\quad e_l = e_l + e_i$
14: $\quad$**end if**
15: **end for**
16: $\mathbf{e}[n] = e_l$+ compute error metric ($\mathbb{N}_s$)

Algorithm 3.3 (a) shows a description of the used algorithm that is performed for every pose n. The current link is denoted as i, the error metric as e_l, the vector containing all error metrics as $\mathbf{e}$. The reference end point for a link is denoted as $\mathbf{p}_r$ and the real end point as $\mathbf{p}_s$. The list of the reference end points for the links is denoted as $\mathbb{N}_r$, the list with real end points as $\mathbb{N}_s$. Again, $\mathbb{N}_{r,s}[i]$ accesses the i-th point of the point cloud $\mathbb{N}_r$ or $\mathbb{N}_s$, respectively. The algorithm for the computation of the best fit of a link's end point can be found in Algorithm 3.3 (b). $\mathbb{N}_p$ denotes the list of points near the reference point $\mathbf{p}_r$ and the resulting temporary error metrics are stored in $\mathbf{e}_t$. Figure 3.6 illustrates the link fitting algorithm after the left and right shoulder already have been fitted. Starting with the upper left arm (a), the lower left arm (b), and the upper right arm (c) are fitted. Figure 3.6 (d) shows the completely fitted body pose. In (b) the end point of the reference link would be placed outside the segment. The algorithm tries to find possible end points, which are placed inside the segment, and selects the one violating the least constrains.

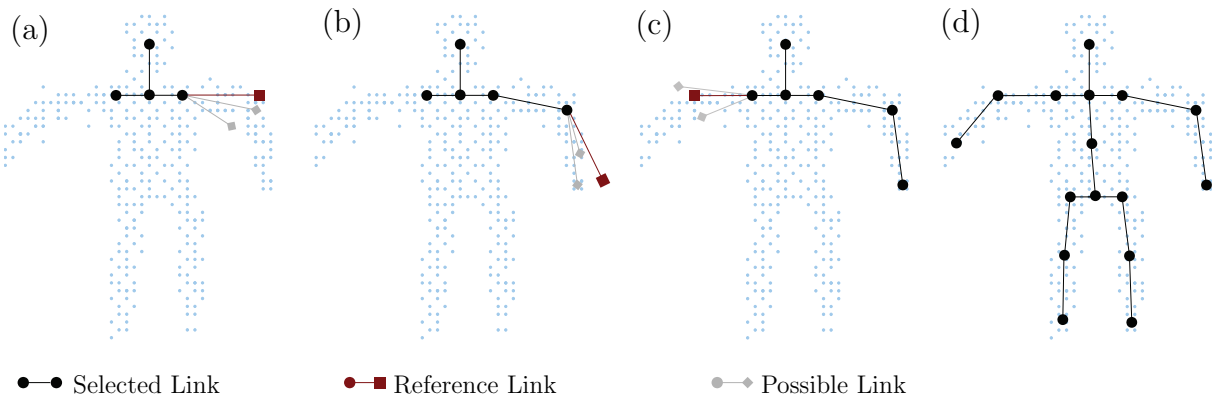


Fig. 3.6: Illustration of the link fitting algorithm. The red links illustrate the reference links, the gray ones the possible links detected by the find best fit algorithm.

Error Metric

To select one of the 27 poses that have been fitted, an error metric has to be computed for every pose. This error metric tries to identify the best fitting pose:

$$\mathbf{e}[n] = \alpha_1 \cdot e_l + \alpha_2 \cdot \sum_{i=1}^{N} a_i + \alpha_3 \cdot (1 - \frac{N_u}{N_c}) + \alpha_4 \cdot \sum_{i=1}^{N} s_i. \tag{3.10}$$

The error metric for a pose n considers the following parameters, each weighted by a parameter α:

- **The number of links.** For each missing link i, a penalty a_i will be added. Links that have other links attached (like the shoulder) will lead to a higher penalty than links with no other attached links (like the lower arm). A successfully fitted link will have the penalty $a_i = 0$.

- **The distance between the reference end points and the detected end points of each link** is also considered. High distances may lead to distorted body poses. This error is stored in e_l.

- **The number of points of the cluster that are not in close proximity to the pose.** When a pose does not use all points of a cluster, it is almost certain that one or more links couldn't be fitted correctly. The number of used points is denoted as N_u, the number of points in a cluster as N_c.

- **The density of points near a link.** If the density becomes locally low, it may be an invalid link. Every time the density near the link i falls below a threshold, a penalty is added to s_i.

Validation of a Body Pose

After all links have been computed for a body pose, it has to be validated to avoid invalid configurations. Again, the lengths of the links are analyzed. Contrary to the validation of a single link, all link lengths are analyzed simultaneously. Equation 3.11 computes the median scaling factor s of the estimated link lengths l^c compared to the reference link lengths l^r. N denotes the number of the estimated links.

$$s = \frac{1}{N} \sum_{i=1}^{N} \frac{l_i^c}{l_i^r} \tag{3.11}$$

After the median scaling factor has been computed, it is compared to the scales of the single links. If the difference is larger than twice the standard deviation, the difference between the link lengths is considered as too large and the pose is invalid. Furthermore, the rotations between the links are considered. The human body is subject to certain restrictions regarding the movement of the links. Many configurations are impossible or futile. To avoid these configurations, a minimal and a maximal angle are considered for every degree of freedom. All necessary angles are computed and if one exceeds the defined interval, the whole body pose will be considered as invalid.

Complexity Analysis

As described above, the algorithm is used on a mobile robot, so the computational consumption should be as small as possible. Both the resolution of the colored point cloud as well as the number of detected skin parts will influence the computation time and the detection of the skin parts scales with the resolution of the input image. Kd-trees have a complexity for the construction of $O(n \cdot \log(n))$ and the expected complexity for a nearest neighbor search is $O(\log(n))$. One kd-tree has to be computed for the whole scene and one for each detected cluster. The strongest influence on the computation is the resolution and therefore the number of points in a cluster. The complexity for the segmentation for each detected skin part is $O(n \cdot n_e \cdot \log(n))$, as it scales linear with the number of points n in the scene and with the expected number of neighbors n_e of each point. Fitting a body pose for a segment has also a complexity of $O(n \cdot n_e \cdot \log(n))$.

This section gave an insight to existing three-dimensional object detection algorithms and introduced a novel algorithm, which is used for the estimation of human body poses, but can easily be extended to detect other objects. Although they provide more information, three-dimensional object detection algorithms are sometimes unpractical. Hence, two-dimensional object detection algorithms will be presented in the next section.

3.4 Two-Dimensional Object Detection

Compared to those using three dimensions, object detection algorithms using two dimensions are more intuitive and less computationally complex and thus faster. By moving a search window across the image the segmentation can be avoided. This section begins with the presentation of two of the most common approaches and a cascade based on different histograms combining both spatial and color information.

3.4.1 Conventional Algorithms

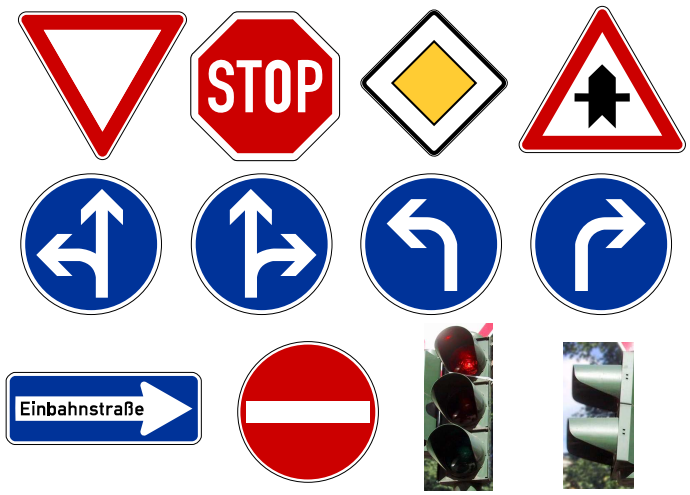


Fig. 3.7: Images of the traffic signs, which can be found at every junction in the city center of Munich, the operational area of ACE.

The most common approaches used for object detection are scale invariant feature transformation (SIFT) [76] and a cascade composed of simple haar-like features [152]. Implementations of both can be accessed easily by using OpenCV, a state-of-the art open source computer vision library. The ACE robot uses both methods to enhance its ability to detect crossroads robustly. For the detection of crossroads, the assumption that every crossroad is equipped with traffic signs or traffic lights can be made for the operational area. The algorithm used for ACE searches for traffic lights and traffic signs as shown in Figure 3.7.
SIFT searches for local features, which are invariant to scale, rotation, illumination, and change in viewpoint. To perform object recognition, feature points of different images can be compared. On the contrary, OpenCV uses a cascade of simple classifiers with haar-like features. The resulting classifier consists of several stages that are applied consecutively to an image until the candidate is rejected at some stage. Figure 3.8 shows, how such a classifier is composed. If a candidate passes all stages, the corresponding object is assumed to be found. Small features have been chosen, so that the size can easily be changed. Hence, an object can occur with different sizes, while all sizes will be detected. The algorithm uses haar-like features to detect lines, corners and center-surrounded features. The results

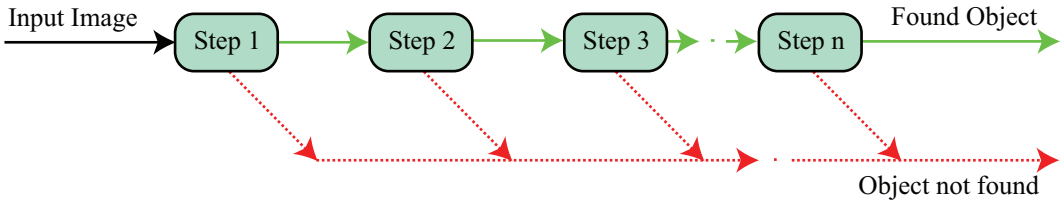


Fig. 3.8: Illustration of the haar-like-cascade. Green solid lines illustrate a successfully executed classifier, red dashed lines illustrated a failed classifier.

obtained by the first tests using SIFT were poor, only 20% of the test images could be classified correctly. Therefore only OpenCV's rapid object detection was used. To achieve good results, over 10000 images have been made to train the haar-like features [75]. One classifier has been trained for each traffic sign or traffic light. When ACE is moving, the algorithm will search consecutively for traffic signs and lights. Every positive result will be tracked for a number of images to be able to deal with false positives. A false positive will most lightly appear in only one single image and not in a longer sequence. If a traffic sign or light is found in more than one consecutive images, the algorithm assumes that it has found a real sign.
To perceive further information, the distance to the traffic sign is measured using stereo triangulation. By measuring the size of the sign in the two-dimensional image and the distance of the sign, the size of the real sign can be computed. If this size is smaller or larger than a defined size, the algorithm knows that a false positive was found. Of course, a large margin has to be used for the defined size, as many traffic signs differ in their size. Both methods can be used easily on other objects, assuming enough training images are available. SIFT and OpenCV's rapid object detection are based on gray scale images and thus ignoring valuable information. Furthermore many training images are necessary yielding a complicated and impractical online learning.

3.4.2 Histogram-based Object Detection Cascade

A novel object detection cascade based on three different types of histograms is proposed, which uses spatial and color information, is capable of dealing with large occlusions, requires few training images and is thus avoiding the disadvantages of other approaches. Color histograms (CH) examine color information, histograms of oriented gradients (HOG) spatial information, and color co-occurrence histograms (CCH) examine both. The main design of each classifier is identical for the whole cascade and will be described below. At first, a set of reference histograms is computed from the training data. In the next step a search window is moved across the image and a histogram is computed for each search window. In contrast to existing cascade architectures, the cascade will not be computed for every search window. Another novelty of the presented approach is its parallel implementation. All histograms of one image are computed in parallel, yielding a two-dimensional set of histograms. Each histogram of this set is intersected with each reference histogram and a matching error is computed for each intersection, leading to a two-dimensional probability distribution. By using a region of interest (ROI), only those regions with a small error

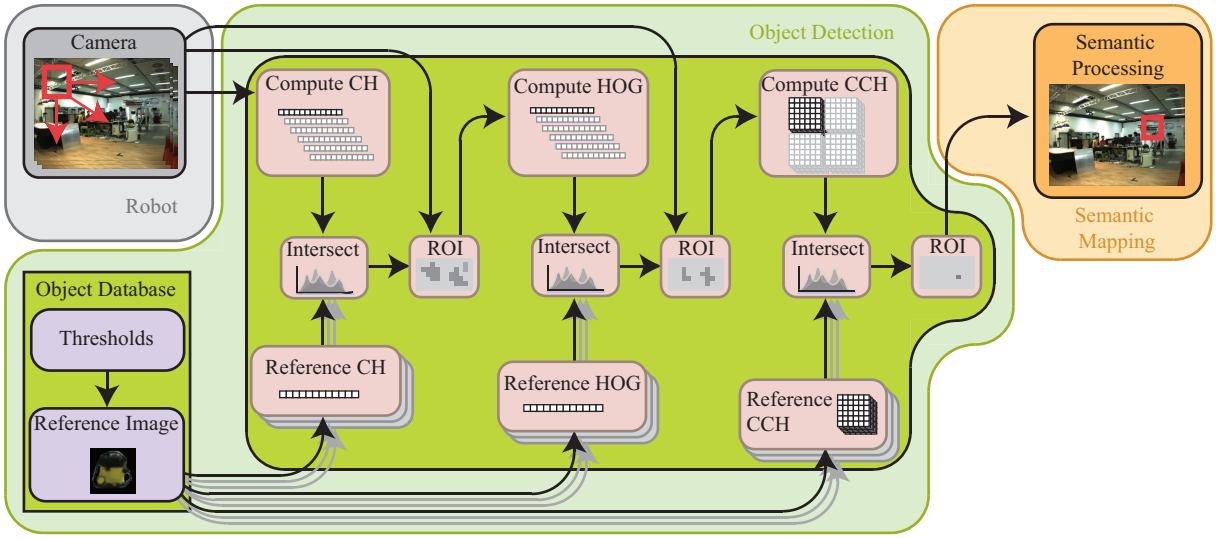


Fig. 3.9: Architecture of the Object Detection Cascade.

will be processed by the next classifier of the cascade. This yields a higher performance of the cascade. An overview of the proposed cascade is depicted in Figure 3.9.

3.4.3 Different Types of Histograms

The following section describes and compares the different histogram types.

Color Histograms

A color histogram is a representation of the color distribution of an image. As only the color of an object and no spatial information like corners is considered, it provides one of the most intuitive and computationally fastest methods for object detection. Figure 3.10 shows how a color histogram with 4 bins is computed for an image with 20 colors. Due to its simplicity, object detection with color histograms suffers some disadvantages. Objects with the same color distribution but different shapes cannot be distinguish. Furthermore, only those objects with a colorful texture can be detected and when using the RGB color space, color histograms are sensitive to changes in illumination. However, this can be solved by using the HSV (Hue, Saturation and Value) color space. As they are fast to compute and can filter most parts of the image, color histograms form the first classifier of the cascade. To avoid large histograms, the number of colors in the color space has to be reduced. The resulting color histogram can be written as $\mathrm{CH}(c)$, where c denotes the color bin.

Histograms of Oriented Gradients

As color histograms contain no spatial information, histograms of oriented gradients are used by the next classifier of the cascade. The basic idea of HOGs is that the shape of the object can be described by the distribution of local gradients, even without knowledge

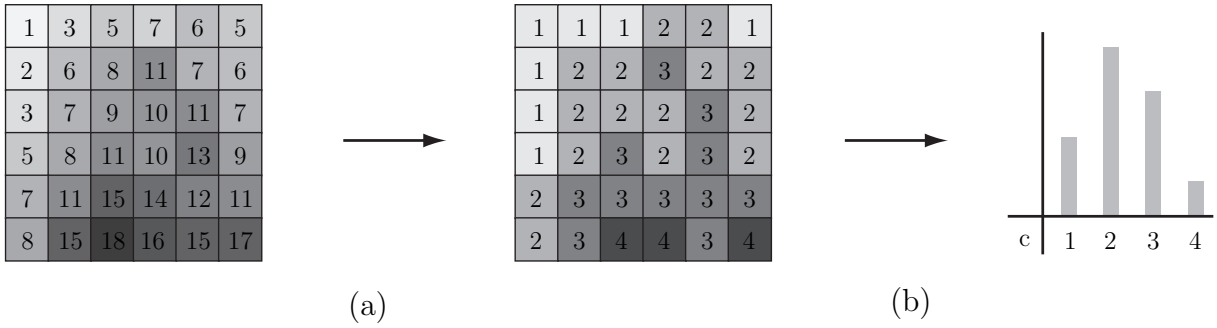


Fig. 3.10: (a) Reduction of color space and (b) computation of a color histogram.

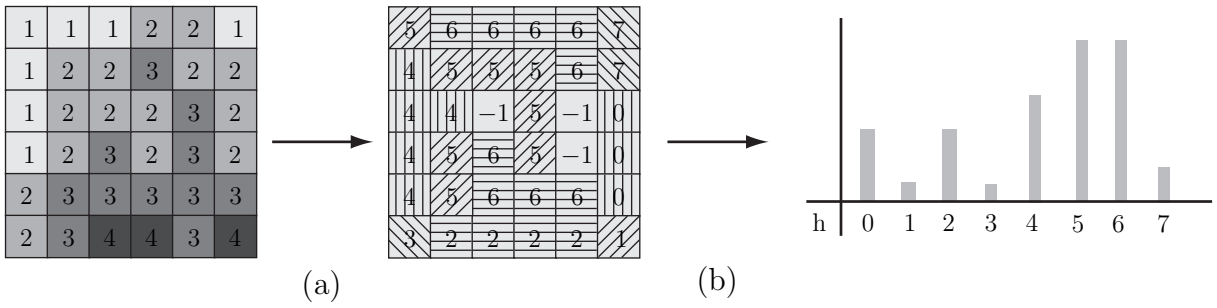


Fig. 3.11: Computation of a histogram of oriented gradients in two steps: (a) computation of the gradients and (b) creation of the histogram.

of their position [29]. To compute the gradient's lengths and orientation, the image $\mathbf{I}$ is convolved with a sobel filter in both x- and y- direction:

$$\mathbf{G}_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * \mathbf{I} \quad , \quad \mathbf{G}_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{I}. \tag{3.12}$$

Now the orientation $\theta_{i,j}$ and length $d_{i,j}$ of the gradient can be computed for each element i, j:

$$\begin{aligned} \theta_{i,j} &= \operatorname{atan}(\mathbf{G}_y(i,j), \mathbf{G}_x(i,j)), \\ d_{i,j} &= \sqrt{\mathbf{G}_x(i,j)^2 + \mathbf{G}_y(i,j)^2}. \end{aligned} \tag{3.13}$$

After computing the gradients, all gradients with a length shorter than a certain threshold are discarded. The remaining gradients are used to compute the histogram. Figure 3.11 shows the computation of a HOG with 8 orientation bins. Discarded gradients are denoted with -1. The resulting histogram of oriented gradients can be written as $\mathrm{HOG}(\theta)$, where θ denotes the orientation bin. As the computational cost of the computation of the gradients can be neglected compared to the computation of the histograms, the computational cost of HOGs is in the same order of magnitude as CHs. They provide a robust classifier to

detect the form of objects, but do not consider color information. Hence, they are selected as second classifier in the presented cascade.

Color Co-occurrence Histograms

By computing the distances between pairs of colors, color co-occurrence histograms utilize both color and spatial information. First introduced as a co-occurrence matrix by Haralick et al. [48], a CCH is a set containing the number of pairs of two certain colors at a certain distance in the image. A CCH can be written as $\mathrm{CCH}(c_1, c_2, \Delta x \Delta y)$, where c_1 and c_2 denote colors and Δx and Δy denote the offset between the colors in x- and y-direction. By only considering the distance $d = \sqrt{\Delta x^2 + \Delta y^2}$, as described by Chang et al. [23], CCHs can be made invariant to rotation in the image plane. Figure 3.12 illustrates the computation of a CCH by showing the computation for the element at position $(4, 3)$, where all elements with a distance $d = 2$ are highlighted. Hence, the cell at $(4, 39$ has the color c_2 and is surrounded at a distance of 2 with one time color c_1, eight times color c_2 and three times color c_3. Such an one-dimensional histogram must be computed for every cell of the input image and for every desired distance. The resulting histograms are then summed up in a last step. In order to reduce some computational costs, only integral distances are considered.

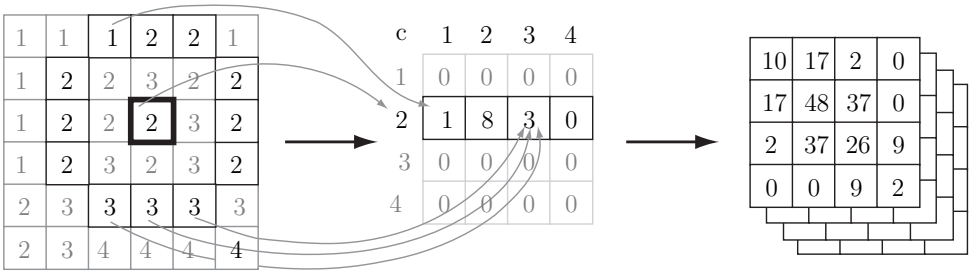


Fig. 3.12: Computation of a color co-occurrence histogram. The given example shows the computation for the element at $4 : 3$ and a distance of $d = 2$.

As the resulting histogram can be described as a three-dimensional array, the memory consumption and the computation time for the intersection algorithm is highly depending on the number of colors and the number of distances. More specific, the time needed to compute a CCH is depending on the number of used distances, while the number of colors only affects the memory consumption. Naturally, the time needed to compute a CCH is larger than the time needed to compute a CH or HOG. On the other side, the quality of the result is better. Details on the computation time and the results of the different types of histograms can be found in Section 3.5.5.

Comparison of the Histogram Types

Table 3.2 compares the most important attributes of the different histogram types. The order of the classifiers in the cascade was determined based on the complexity and am-

Tab. 3.2: Comparison of the different histogram types.

	CH	HOG	CCH
Feature	color	gradients	spatial and color
Complexity	low, $O(n)$	low, $O(n)$	high, $O(n^3)$
Typical size	4096	360	$20 \times 20 \times 25$
Confidence of hit	low - middle	low - middle	middle - high
Confidence of miss	high	middle	high

biguity of the different histograms. As it has a high confidence of miss and consequently is able to exclude most parts of the input image, the first classifier is a color histogram, followed by the histogram of oriented gradients and the computationally expensive color co-occurrence histogram. The complexity of the intersection of the different types is equal to the complexity of the creation, however the execution time is much shorter.

3.4.4 Intersection of Histograms

Based on the intersection introduced by Swain et al. [141], the error e of histogram H compared to the reference histogram H_r can be computed as the sum of absolute differences between the intersected histogram and the reference histogram:

$$e = \sum_{i=1}^{n_b} |H_r(i) - \min(H(i), H_r(i))|. \tag{3.14}$$

n_b denotes the number of color bins in a color histogram and the number of orientation bins in a histogram of oriented gradients, respectively. The computation of the error of a color co-occurrence histogram is more computationally expensive:

$$e = \sum_{i=1}^{n_c} \sum_{j=1}^{n_c} \sum_{k=1}^{n_d} |H_r(c_i, c_j, d_k) - \min(H(c_i, c_j, d_k), H_r(c_i, c_j, d_k))|. \tag{3.15}$$

n_c denotes the number of color bins and n_d the number of distances. Different Minkowski distances can be used to weight larger or smaller errors. Figure 3.13 (a) shows the computation of the error of two one-dimensional histograms, while Figure 3.13 (b) depicts the resulting intersected histogram and the error, which has to be summed up in a last step. In an ideal case, the error $e = 0$. Due to rotation, scale, occlusions, and changes in illumination this ideal case will never be achieved. Hence, a suitable threshold t has to be selected. When using a larger threshold, the algorithm will be able to deal with larger occlusions. However, this will increase the number of false positives. Experiments have shown that the threshold has to be estimated for every object independently. Details about the selection of the thresholds and the correlation between capability of dealing with occlusions and false positives can be found in Section 3.5.5.

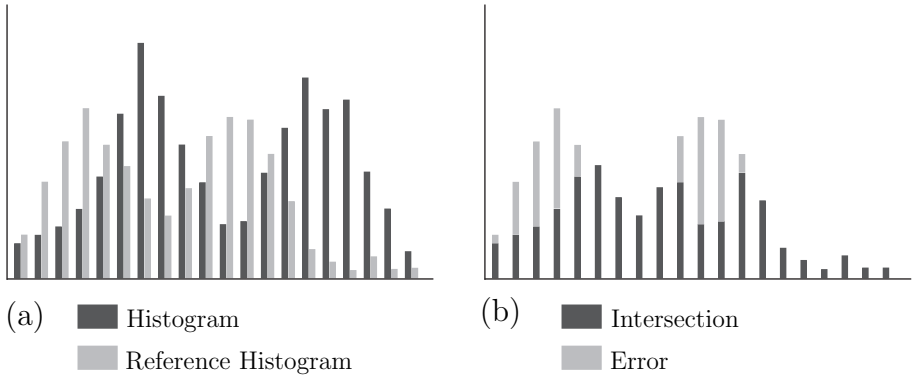


Fig. 3.13: (a) Intersection of histograms and (b) estimation of the error.

3.4.5 Implementation Details

The developed object detection framework can be separated into two parts:

- the **initialization**, where image data and parameters are transferred to the GPU memory and the reference histograms are created out of the reference images, and
- the **main loop**, where a sequence of images is processed as shown in Figure 3.9.

The main loop is only executed on the GPU, where the initialization uses both GPU and CPU. Even small functions, where no speedup could be achieved, have been implemented on CUDA, so unnecessary memory transfers between main memory and GPU memory are avoided.

Computation of Reference Histograms

Before the reference histograms can be computed, the images need to be pre-processed. Every reference image of size $x_r \times y_r$ can be separated into a background and foreground image. The background image is defined by the color with the RGB-value $(0, 0, 0)$ and is ignored during the further processing. In a first step, the remaining image is transformed to the HSV color space. Furthermore, the orientations of the gradients as described in Section 3.4.3 are computed for the whole image. During the last pre-processing step, the number of colors of the input image is reduced twice, once to the number of color bins n_{ch} for the color histograms and once to the number of colors n_{cch} for the color co-occurrence histograms. The number of orientations is also reduced to the number of orientation bins n_{hog} for the histograms of oriented gradients. This results in three images for each reference image: I_{ch} as input for the CH, I_{hog} as input for the HOG and I_{cch} for the CCH. In the last step, the reference histograms can be computed and stored to the GPU memory.

Tab. 3.3: Different Block- and Thread- sizes.

	Thread size	Block size
CH	16×16	$n_x \times n_y$
Intersection	$n_{\text{ch}} \times 1$	$n_x \times n_x$
HOG	16×16	$n_x \times n_x$
Intersection	$n_{\text{hog}} \times 1$	$n_x \times n_x$
CCH	16×16	$n_x \times n_x$
Intersection	$n_{\text{cch}} \times n_{\text{cch}}$	$n_x \times n_x$

Main Loop

Similar to the computation of the reference histograms, every image of the sequence with a size of $x_i \times y_i$ needs to be pre-processed into the three input images I_{ch}, I_{hog} and I_{cch}. Before the first classifier of the cascade can be applied, the input images are separated into $n_x \times n_y$ subregions of size $x_s \times y_r$:

$$x_s = fx_r, \quad y_s = fy_r,$$

with $f \geq 1$. As they have a large influence on speed and quality, the parameters n_x, n_y and f have to be chosen carefully. Furthermore, the subregions have to overlap so that an object at any position is included in at least one subregion. Therefore the overlap has to be equal or larger than the size of a reference image. Consequently, the following constrains have to be fulfilled:

$$n_x \geq \frac{x_i - fx_r}{x_r} \quad , \quad n_y \geq \frac{y_i - fy_r}{y_r}. \tag{3.17}$$

However, a large number of subregions leads to a high computation time and a large size of subregions to less accurate results.
After the configuration of the subregions, the first classifier can be applied to I_{ch}. After all histograms are computed, they can be intersected with the reference histograms, resulting in an error map m^i_{ch} of size $n_x \times n_y$ for each reference object o_i. To compute the region of interest for the next classifier, all error maps are merged into one map m_{ch}. Therefore, the lowest error of each cell is selected:

$$m_{\text{ch}}(x, y) = \min(m^0_{\text{ch}}(x, y), m^1_{\text{ch}}(x, y) \ldots m^{k_i}_{\text{ch}}(x, y)), \tag{3.18}$$

where (x, y) denotes the coordinates of a cell and k_i the number of reference histograms of the object o_i. If the error of a cell is smaller than a threshold t_{ch}, the next classifier is computed for this cell. Hence, the number of computed cells decreases with each classifier. The remaining classifiers are computed in the same manner, resulting in several error maps m^i_{hog} and m^i_{cch}. An object k is found in a subregion, if the error of the corresponding cells of all error maps are below the thresholds t_{ch}, t_{hog}, and t_{cch}, respectively.
Additionally, the specification of thread- and block- size has large influence on the execution time. Table 3.3 shows the selected sizes, where the number of distances is

denoted by n_d. If the thread size exceeds the hardware limitation (512 or 768 threads, depending on the used hardware), the thread size is limited to the maximal value and a function has to be called twice or more in one thread. Of course, this yields a higher execution time and should be avoided. As is leads to a high occupation of the GPU, a thread size of 16×16 was chosen for the computation of the histograms. The thread size for the intersections are defined by the number of bins in the histograms. Future architectures with larger memories and more computational power will allow larger thread sizes.

As they are fast to execute and can be applied to a manifold of different objects, two-dimensional object detection algorithms provide an ideal addition to those working in three dimensions. With SIFT and a cascade of haar-like features, two well know approaches have been presented in this section. A novel cascade using different types of histograms has been introduced, which contributes to the state-of-the-art by being real-time capable, requiring few training images, and being able to deal with large occlusions. A thorough experimental investigation of the stereo processing algorithm and the two types of object detection algorithms will be given in the next section, together with details about the evaluated parameter settings.

3.5 Experimental Results

This section shows some experimental results from the main modules of the object detection subsystem, starting with the stereo image processing. The remaining main modules include the human body pose estimation and the two two-dimensional object detection algorithms. Other three-dimensional object detection algorithms have shown no satisfying results and have thus not been included into the subsystem.

3.5.1 Experimental Setup

The following experiments have been conducted using the vision processing PC on the ACE robot, which was equipped with an AMD Phenom Quad-Core CPU running at 2.5 GHz, 4 GB of physical memory, and two GeForce 9800 GX2 cards and hence four Cuda enabled devices. Unless stated otherwise, only one Cuda device has been used for a module.

3.5.2 Stereo Image Processing

Robust three-dimensional mapping and object detection implicates certain minimal requirements for the underlying stereo module regarding computational speed and quality. A minimal resolution of 640×480 pixels is required and 150 disparities have to be computed at a frame rate of 10 Hz. This frame rate includes the estimation of the camera's ego-motion, which is required for the three-dimensional mapping. Consequently, the sole stereo algorithm has to be even faster. Four datasets with 150 images each have been used for the experimental investigations. The quality of the algorithm was estimated using the

Fig. 3.14: Result of the stereo processing system. The upper row shows the left image from the stereo camera and the lower row shows the resulting disparity map. Points with a low confidence are indicated with black color.

tsukuba stereo set and the evaluation methods provided by the middlebury stereo page[1]. The main challenge during the experiments was to find suitable parameters, which achieve the required speed at the best possible quality. Therefore the influence of the different parameters has been investigated.

As the task requires no subpixel accuracy, the gradients can be neglected for the computation of the costs. Hence, the parameter w from Equation 3.3 can be selected to $w = 1$. The influence of the maximal cost value C_{Max} is relatively small, so it was estimated heuristically to $C_{\text{Max}} = 50$. Most influence on the quality comes from the used algorithm for the aggregation of the costs and the selected window size. In the case of the squared window approach, a larger window size N yields better results. The boundary guided window approach shows the same behavior. However, a $N \geq 7$ yields to a dramatically increasing computation time due to a shortage of the GPU's registers. Registers have to be swapped out to the slower local memory. Furthermore, the adaptive weight window approach cannot be implemented on a GPU so far due to too large memory consumption during the parallelized computation of the weighting factors. The minimum filter approach showed best results for a large window size with a relatively small computational delay. Hence, a window size of $N_{\text{Square}} = 15$ for the squared window algorithm and $N_{\text{Min}} = 5$ for the minimum filter has been chosen.

Figure 3.15 shows the correlation between the window size, the number of mismatches, and the computation time for the squared window approach. For a real-time application, the computation time is of of great importance. All images of all four datasets have been computed five times and the average time for every computation was measured to estimate

[1] `http://vision.middlebury.edu/stereo/eval/`

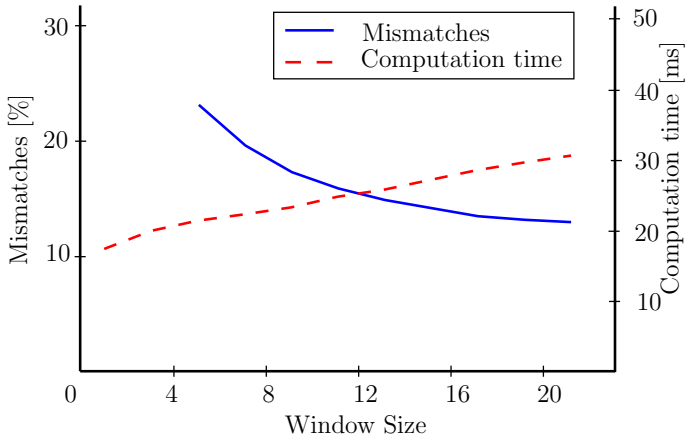


Fig. 3.15: Correlation between window size, mismatches (blue solid line), and computation time (red dashed line).

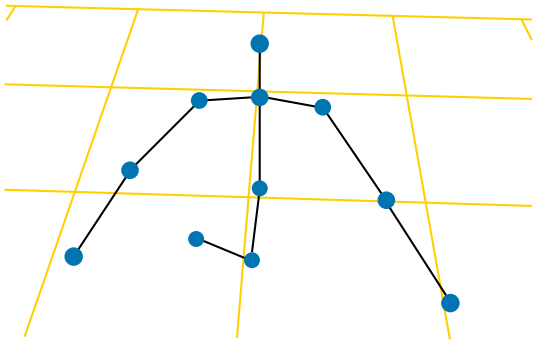


Fig. 3.16: Computed body pose.

the speed. The total computation time of 60 ms is composed as follows: the computation of the costs is performed within 6 ms, the aggregation when using the squared window approach in 28 ms, the application of a minimum filter in 23 ms, and the winner takes it all method for the estimation of the actual disparity in 4 ms. On the contrary, the boundary guided window approach is computed in 50 ms to 270 ms and is consequently unsuited for a real-time application with the present hardware. Future architectures will have other constraints but will allow faster computations with a higher quality.

3.5.3 Human Body Pose Estimation

The estimated body pose of the image given in Figure 3.4 is shown in Figure 3.16. To show the capability of the algorithm to compute the correct body pose in all three dimensions, the body pose is shown from above. The results of the skin detection and segmentation have already been shown in Figure 3.4. Figure 3.17 shows the results from five different scenes. The first row shows the rectified images as seen from the camera and the detected skin parts, while the next two rows show the estimated human body poses from two different points of view.

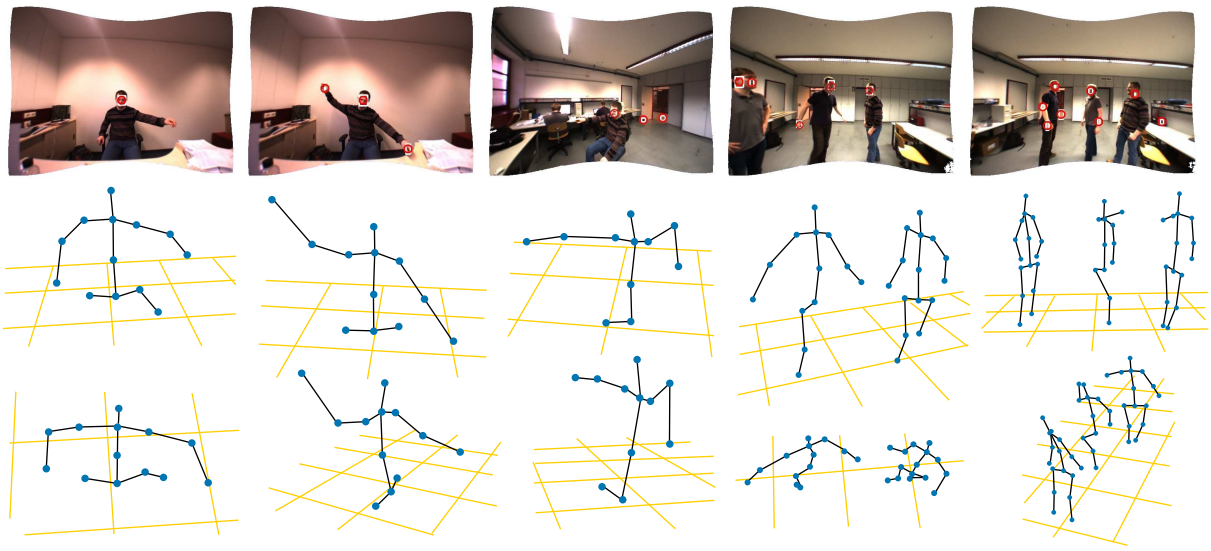


Fig. 3.17: Results of the human body pose estimation.

As a stereo matching algorithm is used to create the point clouds, similar problems as in stereo matching are encountered. The stereo matching algorithm is unable to compute a disparity for large areas of the same texture, so only silhouettes and no filled representations can be found. Furthermore, some invalid stereo matches may lead to unpredictable behavior. An experiment has been conducted to give a qualitative evaluation of the algorithm. Users have been asked to point in a direction, and the measured angle of the direction they are pointing to has been compared to the computed one. A measurement was assumed to be incorrect, when the algorithm was not able to fit the arms or the error was larger than 45°. 150 measurements have been recorded and the algorithm was able to estimate 80.2 % of the postures correctly, while the median error was around 6.8°. Only 3.2 % of the false positives could not be detected. As the algorithm will leave occluded body parts out, it is not able to compute the pointing direction when the user is pointing away from the camera and the arm can not be seen. Almost half of the body poses, the algorithm was not able to estimate in the experiment, can be explained by this problem. The other invalid matches can by explained by noise in the point cloud or with other objects that have been mixed up with body parts. The use of color may be an useful extension to increase the robustness of the algorithm.
Time constraints are hard on a mobile robot, so the estimation of the body poses must be completed in nearly real-time. Without optimization, the segmentation and human body pose estimation is performed in less than 150 ms. Together with the skin color detection and stereo matching, a frame rate of 5 fps is achieved on a standard PC. Using a dual core processor or another skin color filter could increase the computation speed up to 10 fps.

3.5.4 Detection of Traffic Signs using a Cascade of Haar-Like Features

The hit rate of the cascade of haar-like features is within a range of 77 to 88% and the number of false positives within 0 to 6%, depending on the classifier. Table 3.4 shows the result of the used classifiers and the selected stage. For the selection of the number of stages, a compromise had to be found. A higher number will yield to less false negatives, but to more missed objects. Figure 3.18 shows the correlation between the number of stages, the miss-rate and the false positives for the classifier used to detect the yield sign. As most crossroads are equipped with more than one traffic sign and the algorithm is able to deal with a small number of missed signs, the number of stages near the intersection of the miss-rate and the number of false positives was selected.

Tab. 3.4: Results of the classifiers

Classifier	Stage	Hits	False positives
Yield	20	78 %	0 %
Stop	20	79 %	3 %
Have Priority	20	88 %	6 %
Arrow Left	18	88 %	3 %
Arrow Right	16	84 %	0 %
One Way	15	77 %	5 %
Traffic light	20	81 %	2 %

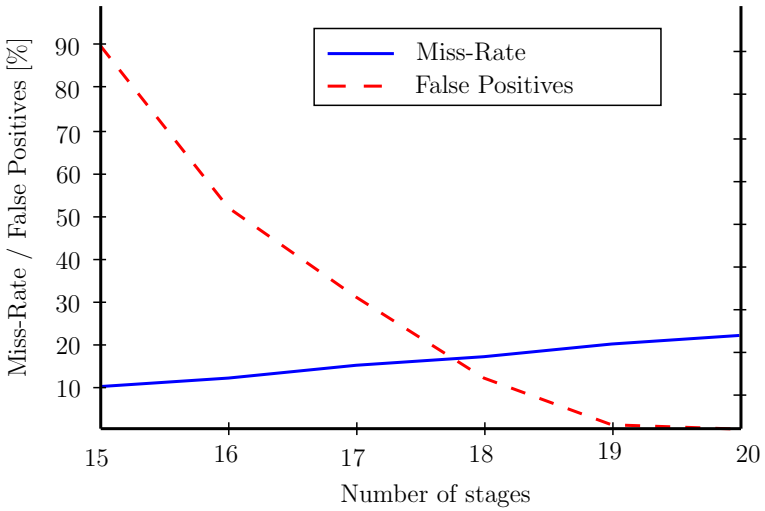


Fig. 3.18: Correlation between number of stages, miss-rate (blue solid line), and false positives (red dashed line).

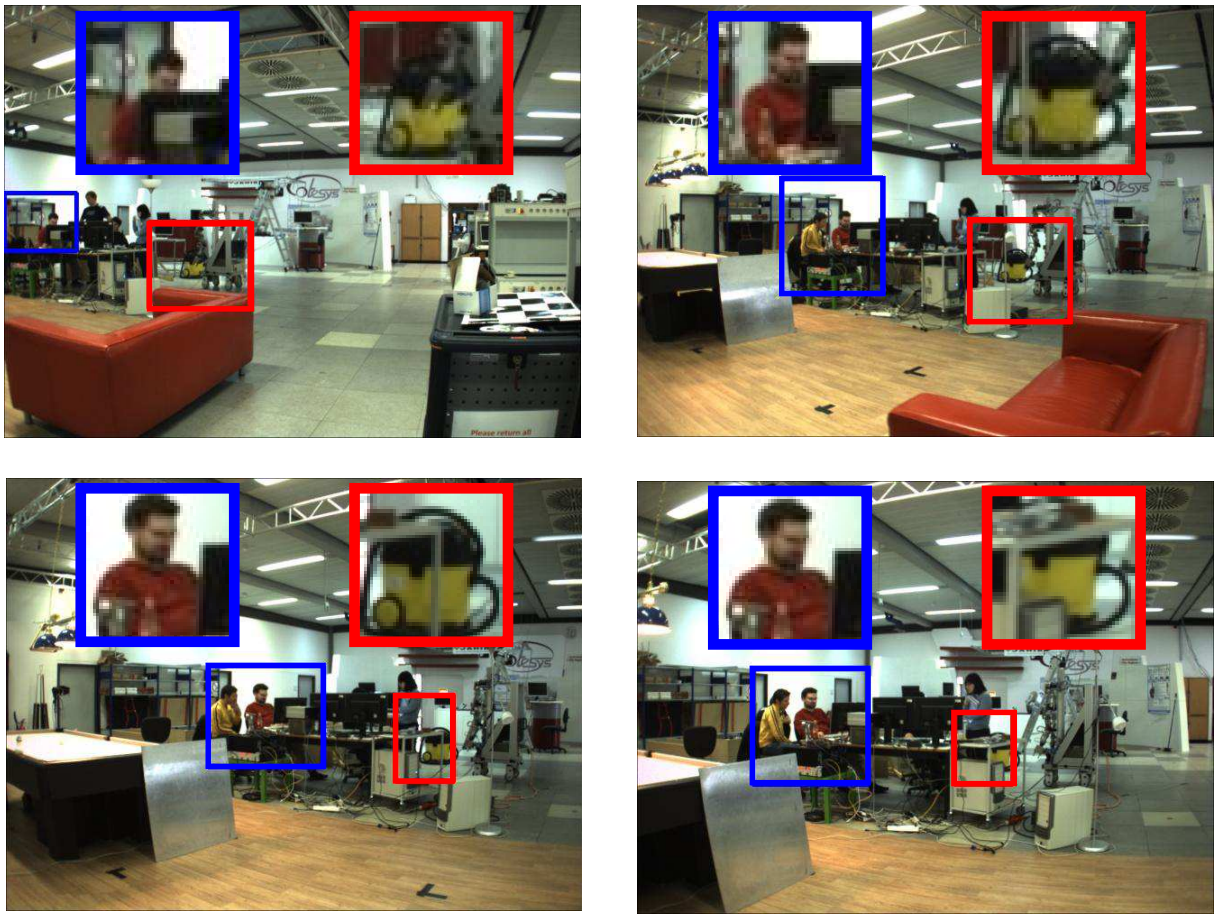


Fig. 3.19: Result of the object detection cascade with the two objects *face* and *cleaner* at different levels of occlusion. The area with the object is magnified in the top section of each image. The reference image has a size of 48×48.

When the robot is approaching a crossroad, the traffic signs will be seen in more than one image. False positives will only occur in one single image, so they can be detected and discarded by tracking the detected traffic signs. If the algorithm misses a traffic sign in one single image, it will be able to detect the sign in the rest of the sequence. Consequently, most of the not detected signs and false positives can be handled. Depending on the number of stages and haar-like features, one classifier can be computed within 100-200ms. As the process can be easily parallelized, a frame rate of 3Hz can be achieved on a dual-core processor and 6Hz on a quad-core, respectively.

3.5.5 Object Detection using a Cascade of Histograms

The influence of different histogram sizes and search window sizes on the performance and quality has been investigated in several experiments using a total of 500 images with objects, 75 images without objects and 4 objects with 8 reference images each. Figure 3.19 shows a sequence, featuring several different levels of occlusion of the vacuum cleaner. The size of an input image is $x_i \times y_i = 640 \times 480$, of a reference image $x_r \times y_r = 48 \times 48$, and the

size of the search windows was chosen as $x_s \times y_s = 64 \times 64$, so $f = 4/3$. A total of 38×28 subregions have been computed, yielding to an overlap of 16 pixels in each direction. The lower right image in Figure 3.19 was used as input for the following experiment.
Figure 3.20 (a) shows the error map of the intersection of the color histograms with one

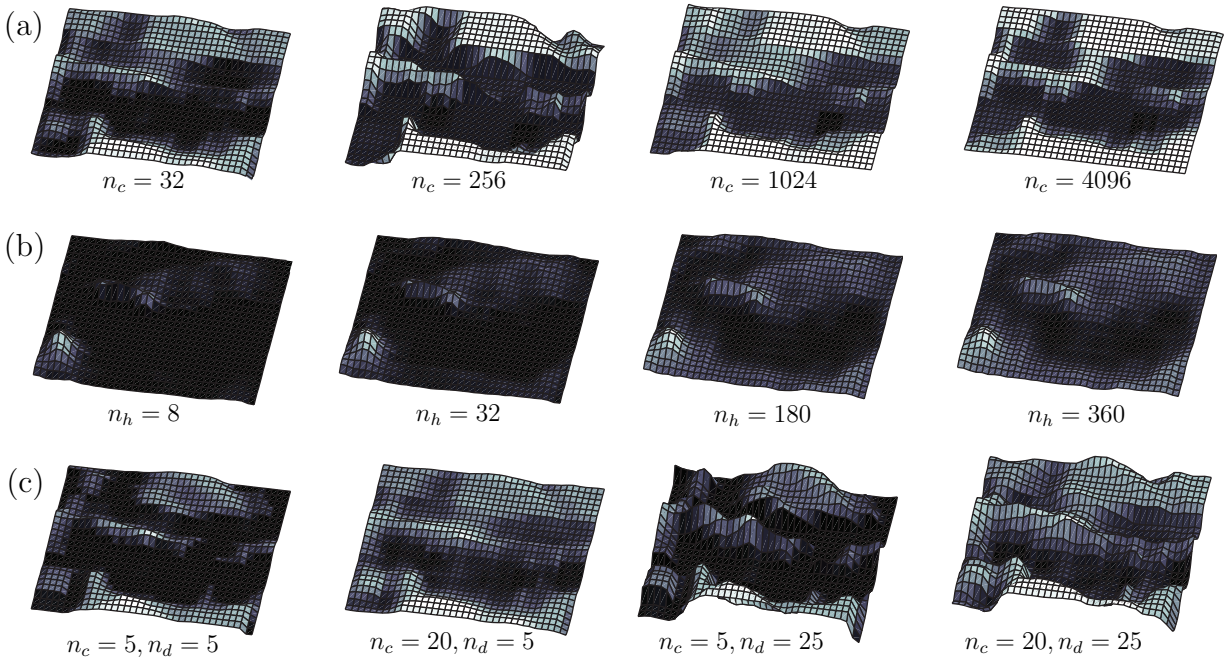


Fig. 3.20: Result of the intersection of different histograms types with different parameters. A darker color indicates a lower error and thus a better result: (a) Intersection with color histograms with different numbers of colors, (b) intersection of oriented gradients with different numbers of orientations, and (c) intersection of color co-occurrence histogram intersection with different numbers of colors and distances.

reference histogram. Dark colors indicate a lower error. For $n_c = 32$, the computation time of all histograms was around 190 ms, for $n_c = 256$ around 140 ms, for $n_c = 1024$ around 110 ms and for $n_c = 4096$ around 85 ms. The faster execution time for a higher number of colors can be explained easily. As the computation of a color histogram is executed in many parallel threads, accessing the memory is the limiting factor. With a smaller n_c, there are less registers in the GPU memory, which have to be accessed more often and the threads are blocking themselves. Hence, a larger n_c requires more registers and yields less threads blocking other threads and thus a faster execution time. Conveniently, this goes in hand with a better quality of the result. The computation time for an intersection is around 3 ms with additional 0.3 ms for every reference histogram. In the first classifier around 75% to 80% of the input image can be excluded from the further processing. The impact of the number of orientations for the histograms of oriented gradients is shown in Figure 3.20 (b). Again, a larger n_h yields to a better quality and a faster computation speed, this time ranging from 140 to 40 ms for a whole image. As not the whole image is processed, an effective computation time of 10 ms including the intersection can be achieved. The computation time for the intersection is identical to the one necessary for

the color histograms. Again, around 75% to 80% of the remaining input image can be excluded. As it is the most complex histogram, the time needed to compute the color co-occurrence histograms is larger. Figure 3.20 (c) shows the quality of the results for different numbers of colors and numbers of distances. Again, a larger number of colors leads to a faster computation. With $n_c = 5$ and $n_d = 5$, a time of 3000 ms is necessary, for $n_c = 20$ and $n_d = 5$ a time of 750 ms, for $n_c = 5$ and $n_d = 25$ a time of 2400 ms and for $n_c = 20$ and $n_d = 25$ a time of 1600 ms. As an unoptimized CPU-implementation requires a computation time of 200 seconds, a remarkable speedup was achieved. As only 5% to 10% of an image are processed in this classifier, an effective computation time of 15 ms including the intersections can be achieved. The total computation time is around 100 ms. By using two CUDA devices, a frame rate of 15 Hz could be achieved. Furthermore, the texture of the scene has an influence on the speed. Images with many different colors lead to a faster execution time compared to images with large areas of the same color. Again, this can be explained by threads blocking the memory accesses of other threads.
Next to the speed, the quality of the results is another important attribute of an object detection algorithm. Therefore several sequences have been used to test the algorithm, where false positives and missed objects have been counted. Only 7 false positives and 23 missed objects have been counted in the 575 test images, yielding to a detection rate of 95%. As shown in the lower right part of Figure 3.19, the presented algorithm is able to deal with large occlusions. The thresholds have to be tuned carefully, as high values will yield to a large number of false positives and too small values to too many missed objects. Figure 3.21 shows the correlation between the detected occlusions, the miss-rate and the

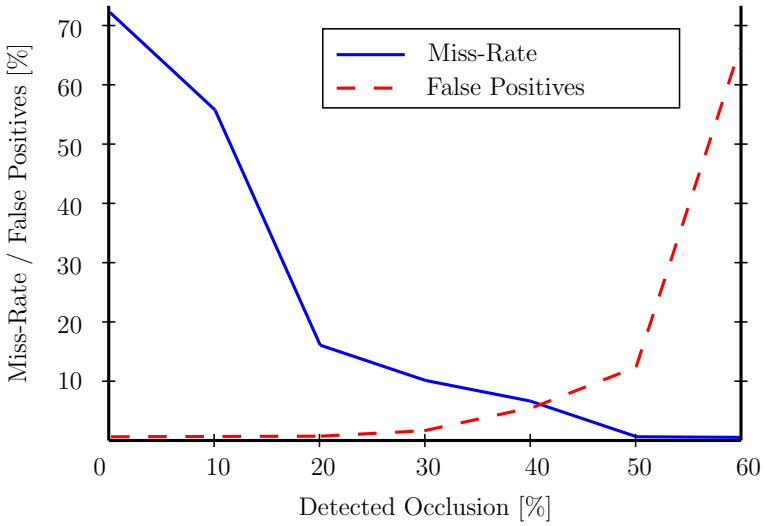


Fig. 3.21: Correlation between detected occlusions, miss-rate (blue solid line), and false positives (red dashed line).

rate of false positives. The same datasets as before have been used to evaluate the miss-rate and the number of false positives, while the occlusion was measured with synthetical data. A suitable trade-off marks the intersection, showing that the algorithm is able to detect objects with an occlusion of up to 40 % at a low miss-rate.

3.6 Discussion

This chapter introduced several algorithms for robust object detection, starting with a fast stereo algorithm running at a resolution of 640×480 pixels and 150 disparities with a speed of almost 20 Hz. Although the algorithm is not able to reconstruct a full image, it is able to eliminate bad results and the quality of the resulting disparity map is thus adequate for further processing. Based on the stereo reconstruction, a three-dimensional model of the environment is created, which is serving as input for several three-dimensional object detection algorithms. The estimation of human body poses was presented as a special case of three-dimensional object detection and is able to estimate most of the body poses correctly. This algorithm can easily be extended to detect other types of objects. Two state-of-the-art algorithms for two-dimensional object detection, namely SIFT and OPENCV's rapid object detection cascade, have been presented and experimentally investigated. Moreover, a cascade of different types of histograms has been introduced. This cascade marks an object detection algorithm running at a high accuracy with a speed of 15 Hz and is capable of dealing with images with large occlusions of 40 % and furthermore requires very few training images. Consequently, the algorithm is well suited for a mobile robot's cognitive architecture.
Despite the promising results, there exist some limitations to the presented algorithms. Due to memory limitations on a GPU, the presented stereo algorithm is not able to reconstruct stereo images with a higher resolution, omitting valuable information. The algorithm for body pose estimation is depending on a reliable detection of the start point. Whenever the head of a person is occluded, the body pose cannot be computed correctly. Further skeleton based objects still have to be included. Estimating the pose of other manipulators is a fascinating research topic. Last but not least, the cascade of different histograms is not able to detect attributes like color at its current state of development. Future work in the area of object detection might deal with real-time capable stereo algorithms running at high resolutions and more complex three-dimensional object detection algorithms. A possible implementation of the presented cascade in three dimensions would require vast amounts of computational power, but promises excellent results.
As it is capable of detecting different types of objects, running in real-time, and allowing online learning, the presented object detection subsystem provides an ideal base for the presented cognitive architecture. However, a metric map is crucial for path planning and relating the objects with further information. The following chapter presents a vision-based mapping system.

4 Vision-Based Mapping

As robots navigating in our environment or assisting humans have to be reliable, they must have the ability to perceive and understand their environment. This understanding process can be separated into two major steps: object recognition and mapping. In the first step, robots have to be able to find and identify objects in their environment. On the contrary, knowing all objects would be useless without being able to locate them in a map. Consequently, a mobile robot has to be equipped with a mapping system. Like object detection, vision-based mapping can be divided into two major groups: two- and three-dimensional mapping, with several suitable representations each. Two-dimensional maps are typically described by an occupancy grid, while three-dimensional maps are mostly stored as a point cloud or a polygon grid.

Most current two-dimensional mapping systems are based on a texture analysis and the assumption that the area in front of the robot is free of obstacles. However, these algorithms are mainly developed to detect streets and have problems when the texture is varying and are hence unsuited for changing terrain or indoor environments. Consequently, this chapter proposes a two-dimensional mapping algorithm, which uses a memory to remember older textures and is thus robust to changing terrains. The detected ground is then reprojected into a metric map, which can be used for path planning. This novel method is well suited for both indoor and outdoor scenarios.

Two dimensions have proven to be insufficient for planning complex manipulation tasks. Other applications for three-dimensional mapping include the reconstruction of distant places. Hence, the ability to build accurate three-dimensional maps is essential for a high level of cognitive understanding and thus autonomy. Recent research has revealed different ways to create those maps, mainly differing in the type of the used sensor. Laser rangefinders provide good quality and are easy to use. On the other hand, they provide no color information. The most obvious way to create three-dimensional maps is inspired by the human vision system, where the use of two eyes allows depth perception. Existing algorithms are encapsulated systems with highly specified and interconnected submodules like stereo reconstruction, ego-motion estimation, and the actual fusion algorithm. Hence, a three-dimensional mapping subsystem with a novel modular design is introduced. This subsystem utilizes the previously presented real-time stereo algorithm combined with a sophisticated ego-motion estimation and the actual fusion algorithm. As the modules of the system can be replaced by others with the same functionality, even laser rangefinders can be included. Finally, the genetic ICP (iterative closest point) algorithm is presented, a novel method to merge different types of sensors, namely laser rangefinders and a stereo camera system. In a last step, the three-dimensional map can be used to detect obstacles that cannot be found by the two-dimensional mapping system.

The remainder of this chapter is organized as follows: An overview of current vision-based mapping systems is given in Section 4.1, followed by the two-dimensional mapping module in Section 4.2. Section 4.3 presents the three-dimensional mapping module and the ego-

motion estimation. The chapter concludes with the fusion of different sensor types in Section 4.4, experimental results in Section 4.5, and a discussion.

4.1 Overview of current Vision-Based Mapping Systems

This section gives an overview of the state-of-the-art in current vision-based mapping systems, starting with two-dimensional mapping, followed by the ego-motion estimation and the three-dimensional mapping. The section concludes with the fusion of different sensor types.

Two-Dimensional Mapping

Knowledge about the type of ground, a mobile robot is currently driving on, is mandatory for safe operation. When examining a two-dimensional image, the areas showing the ground have to be identified and isolated. To create an accurate map of the environment, the robot has to know his absolute position. The most common approach to this problem is the use of a laser rangefinder to scan the environment and to use a simultaneous localization and mapping (SLAM) algorithm, which is well explored [32, 62, 97]. However, SLAM algorithms require a laser rangefinder.
Most vision-based algorithms for ground detection are road following algorithms [147]. These algorithms are limited to their original purpose and are not able to resolve small structures and thus will not work properly on a sidewalk or inside buildings. Algorithms designed for mobile robots or wheelchairs use depth maps to detect obstacles in front of the robot [28, 82]. Another algorithm uses an affine transformation and a border detection filter to find the borders of the sidewalk [64]. The most promising approach to robust classification and detection of the terrain, the robot is currently driving on, is the use of color histograms [131, 145]. On the other hand, these algorithms cannot deal with a changing type of ground and have therefore to be extended with memory.

Ego-Motion Estimation

Odometry offers the simplest way to estimate the movement of a robot. However, the odometry needs to be synchronized with the camera and some robots are not equipped with an odometry module. The use of laser rangefinders is a common approach to estimate the robots position, by attempting to solve the SLAM problem. Some approaches separate SLAM and the creation of a three-dimensional map [36, 140], others use a three-dimensional SLAM approach and split the measured data into smaller parts of a fixed size and use ICP based algorithms to register and align the single parts. These algorithm can be separated in two groups: those using only laser rangefinders [59, 103] and those using both cameras and laser rangefinders [15, 60].
Vision-based estimation of the camera's ego-motion is a well suited alternative to classical odometry or SLAM. An overview of the different methods for the ego-motion estimation

can be found in [10]. First approaches used a monocular camera by estimating the optical flow [77, 143]. Horn et al. [95] tried to eliminate the depth mathematically, while Shi et al. [144] proposed an algorithm for the estimation of the image's deformation by using the parallax. Another class of algorithms is based on stereo vision [57], trying to estimate the movement in cartesian coordinates [30]. These methods are closely related to the stereo reconstruction and can consequently be computed simultaneously. Hence, the proposed algorithm for ego-motion estimation is based on the stereo reconstruction algorithm presented in Section 3.2.

Three-Dimensional Mapping

Building three-dimensional maps is of great interest for a wide variety of tasks, such as autonomous systems in underwater environments [53] or in distant places like the mars surface [129]. An augmented virtual reality is a desired tool to handle remotely controlled systems [108]. Another important application is the reconstruction of crime scenes that facilitates and accelerates the preservation of the setting [128]. One of the first approaches for scene reconstruction was developed by Fua et al. [37] and combines a representation based on particle filters with an image based optimization strategy. Every particle represents a surface element, which position and orientation is then optimized. Other modern algorithms are based on this approach [47] and utilize other optimization methods and strategies.
Reconstruction algorithms can be separated into two groups. One group is based on volumetric elements (voxels), where each depth map is transferred into the voxel-space and the surfaces are approximated [43, 120]. Another well-known algorithm is the ICP algorithm, which can be used to merge two overlapping point clouds [14]. The other group of object reconstruction algorithms is based on depth maps. A depth map can be converted into a polygon grid, whose edge-points are then used as state space of a kalman-filter [67]. Other approaches use local correspondences between two or more depth maps [128]. One of the most advanced methods is the median fusion algorithm developed by Nister et al. [99], which projects depth maps into a reference perspective before applying the actual fusion algorithm. As this method is real-time capable [85], the system presented in this chapter is based on this approach.
A robot is equipped with a sophisticated stereo system, providing a sequence of disparity maps, which are converted into three-dimensional colored point clouds. After application of pre-processing steps like the reduction of noise and outliers, these point clouds are merged into one large point cloud using the median fusion algorithm. As these point clouds provide a large overlap. This redundancy is used to increase the quality of the reconstruction. One of the advantages of the presented module is its modularity due to well defined interfaces. For instance, the user can choose a stereo module with particular quality, resolution and speed. Arbitrary pre-processing steps can be included easily and the module is consequently not only limited to merging point clouds obtained by stereo vision, but also other sensor types. For instance, laser-range finders could be included easily instead of or in cooperation with a stereo module.

Fusion of Vision and Laser Data

Before laser and vision data can be merged, the extrinsic parameters (i.e. the position and orientation) of the laser rangefinders have to be measured. As the alignment of the laser scanners is changed frequently, the measurement should be automated. This procedure is called calibration. Although previous work deals mostly with the calibration of the intrinsic parameters of a laser rangefinder [25] and the calibration between a laser rangefinder and a camera [84, 146, 162], some examine the calibration between two or more laser rangefinders [155]. Previous approaches are specialized for a single setup of the system and type of laser rangefinder. A common and easy to use approach allowing the automated calibration of all types of laser rangefinders is still subject to further research.
When it comes to registration of two or more scans, the ICP algorithm [14] is widely spread, yielding many different types of the algorithm. The most common types are based on quaternions [14, 56, 72] or use a singular value decomposition [156] to compute the transformation between the point clouds. Other algorithms use standard optimization methods [69, 88] to minimize an error function. Another approach is the use of extended gaussian images [55, 79], an alternative representation of the shape of surfaces. An overview of other ICP-variants, which use different methods to optimize the execution time [61, 127, 158] can be found in [116]. The main disadvantage of the ICP-algorithm is its disability to align point clouds with different resolutions, noise, or with a small overlap. So far, no algorithm that can register those point clouds properly and reliable has been developed.
If the measured data cannot be merged in a proper way, the usage of multiple types of sensors is in vain. Previous work has dealt with the fusion of image data and point clouds created with a laser rangefinder. The most common application is the creating of an accurate textured reconstruction of indoor or outdoor scenes [78, 113, 122]. Other applications are localization [96] and object scanning [17, 139, 161]. These algorithms transform the point cloud measured by the laser to the cameras point of view and perform a ray based mapping of color information. If the transformation is not correct, this will yield displacements and distortions and the whole scan is corrupt. Although it is an obvious and promising approach, the use of stereo image processing is not common. Possible errors will only have local effects and the rest of the scan can be used for further processing.

4.2 Two-Dimensional Mapping

A robot driving in an outdoor or an indoor setting needs to gain knowledge about the ground and obstacles. Vision systems are well suited to fulfill this task. This section presents an algorithm for the detection of the ground in two-dimensional images. A metric map can be built and then used to obtain more semantic information. The proposed algorithm was used by the ACE robot to determine, if the robot is allowed to drive on the ground in front of the robot or not. To fulfill this task, the algorithm has to be able to distinguish between the sidewalk and the street, which are both of similar color and only differ slightly in the texture. Hence, parameters have to be adjusted carefully and the algorithm must be able to distinguish the different textures. When this task is accomplished, further semantic information can be obtained by analyzing the resulting metric map. This

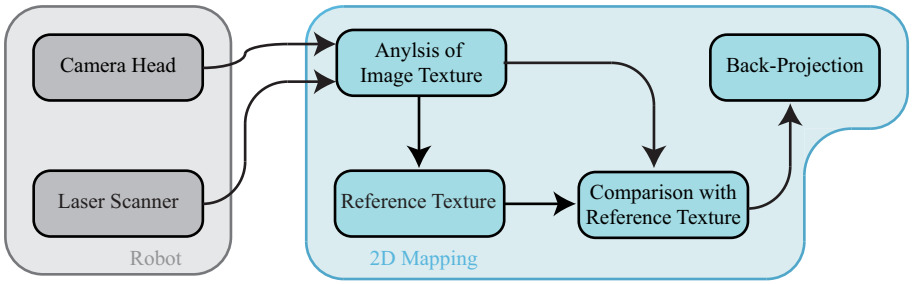


Fig. 4.1: Architecture of the two-dimensional mapping module.

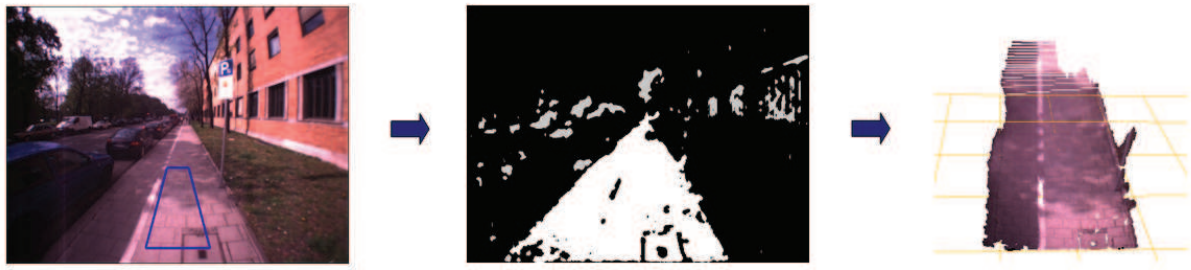
information includes the type of the ground and can be used to verify the result.
To detect the ground, an algorithm based on the one presented in [131, 145] is used. The proposed method uses one camera and is real-time capable at high resolutions. The algorithm is based on the assumption that the area in front of the robot is free, an assumption that can be cross checked with a laser scanner. If the area is free, the texture in the area is compared to the texture of the rest of the image. All areas in the image with a similar texture are assumed to be part of the same type of ground, the robot is currently driving on. To deal with different terrain types, a novel version of the algorithm is proposed. This advanced version is extended with a memory, so that the robot remembers older textures, which are known to be part of a valid terrain. Consequently, the history of textures is used to detect previous valid terrains in an image. In order to increase the robustness for sudden changes in the terrain's texture, the oldest textures are weighted with a low weighting factor and the new textures with a higher one.
Figure 4.1 shows the architecture of the algorithm, starting with a camera and the optional laser scanner. Now, the reference texture is analyzed, post-processed, back-projected and stored in a map.

4.2.1 Detection of the Ground

The analysis of the image texture is explained on the basis of a simplified version, where the reference texture is computed for every single image and no memory is used.

Analysis of the Image Texture

The algorithm is based on the assumption that a reference area in front of the robot is free of obstacles. This reference area is marked by the blue trapezoid in Figure 4.2 (a). Since a laser rangefinder mounted near the ground and sweeping in a two-dimensional plane can be found on almost every mobile robot, it can be used to validate this assumption. This laser rangefinder is used to detect obstacles with a positive height in front of the robot, whereas negative obstacles like holes cannot be found. The sweeping plane L of the laser rangefinder is illustrated in Figure 4.4.
As it is robust to different light conditions, the HSV (hue, saturation and value) color space is used for further processing. The simplified algorithm computes a two-dimensional color histogram of the reference area, containing the *saturation* and the *value*. Color histograms



(a) (b) (c)

Fig. 4.2: Processing steps of the ground detection algorithm with (a) a picture of the scene, the (b) result of the comparison with the reference texture and (c) a back projection of the scene.

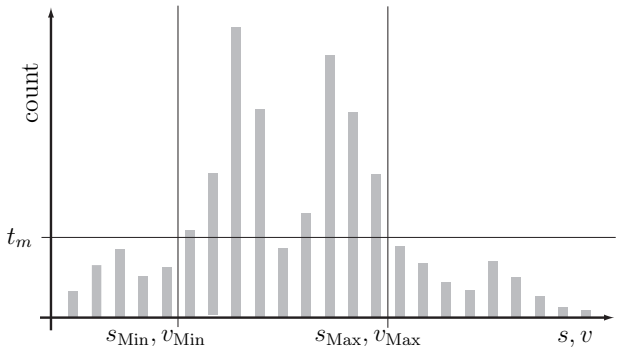


Fig. 4.3: Histogram during a texture analysis, containing saturation (s) or value (v), the threshold t_m and the variables s_{Min} and s_{Max}, and v_{Min} and v_{Max}.

can be used to model both the general color of the reference texture and of the current background. Histograms offer some advantages, as they can be obtained and compared quickly. Figure 4.3 shows a color histogram with several bins. Starting with the smallest, each bin is analyzed. If the height of the bin is lower than a threshold t_m, the bin will be discarded and a reduced histogram is created. Hereby image noise can be reduced. For the reduced *saturation* histogram, two variables s_{Min} and s_{Max} will be computed and v_{Min} and v_{Max} for the reduced *value* histogram, respectively. s_{Min} denotes the color value of the lowest bin, s_{Max} of the highest bin. v_{Min} and v_{Max} are computed in the same way. The threshold and the resulting parameters are also shown in Figure 4.3.
Now, the *saturation* and *value* of each pixel of the whole image is compared to the computed histogram. Again, the saturation s_i and the value v_i of the pixel with index i is computed. The algorithm will recognize the pixel as part of the ground, if s_i and v_i are within a certain range:

$$s_{\text{min}} \leq s_i \leq s_{\text{Max}}, \tag{4.1}$$

$$v_{\text{min}} \leq v_i \leq v_{\text{Max}}. \tag{4.2}$$

As seen in Figure 4.2 (b), the algorithm is able to detect obstacles that are located in the reference area, which was originally assumed to be free. Due to the threshold t_m, small objects laying in the reference area will not be included in the reference histogram.

Post-Processing

After the ground is detected, the resulting image can be post-processed to reduce noise and to remove fragments. Therefore an average filter is used. Every part of the ground, the robot is allowed driving on, must be connected to the part of the ground where the robot is currently standing on. Consequently, the parts of the ground, which are not connected to the reference area, are removed. A simple flood-fill algorithm that is starting at the reference area can be applied for this computation. White spots in Figure 4.2 (b) indicate parts of the ground, which are connected to the reference area. Red parts are not connected and will be discarded.

4.2.2 Computation of the Reference Texture

The simplified algorithm works fine, if the texture of the ground is constant. If the texture is changing abruptly, the algorithm will not be able to validate the area in front of the robot. Consequently, the algorithm has to be modified to deal with different textures. Therefore the ability to remember the texture is introduced by storing the parameters s_{Min}, s_{Max}, v_{Min} and v_{Max} in a ring-buffer of the size N_b. i denotes the current index of the buffer. A metric m_j is computed for every pixel p with the index j of the image. p_j is compared to every histogram of the ring buffer as described in Equations 4.1 and 4.2. The index k denotes the histogram at position k in the buffer. If both equations are fulfilled, the weighting term m_j^k will be added to m_j:

$$m_j^k = \begin{cases} \frac{N_b-(i-k)}{N_b}, & \text{if } k \leq i \\ \frac{k-i+1}{N_b}, & \text{if } k > i \end{cases} . \tag{4.3}$$

The newest histogram will get a high weight, while the oldest histogram will get a low one. If the sum of all weighting terms m_j is higher than a certain threshold t_w, the pixel will be recognized as part of the ground. Once learned, the robot will remember a certain texture. If a robot is not seeing the texture for a while, it will start to forget it. As the weighting factor gets lower and lower for old textures, forgetting is a fluent process. Through the high weighting factor of the newest texture, the algorithm is able to react to sudden changes in the texture quickly. t_w and the size of the ring buffer N_b are depending on the computation rate, on the robot's speed, and on the diversity of the ground's texture. Both have to be adjusted heuristically through experiments.

4.2.3 Back-Projection

Assuming the extrinsic camera parameters and the intrinsic camera parameters are known and the camera image is undistorted, the computed terrain can be back-projected to compute an accurate map of the environment. Figure 4.4 illustrates the camera coordinate

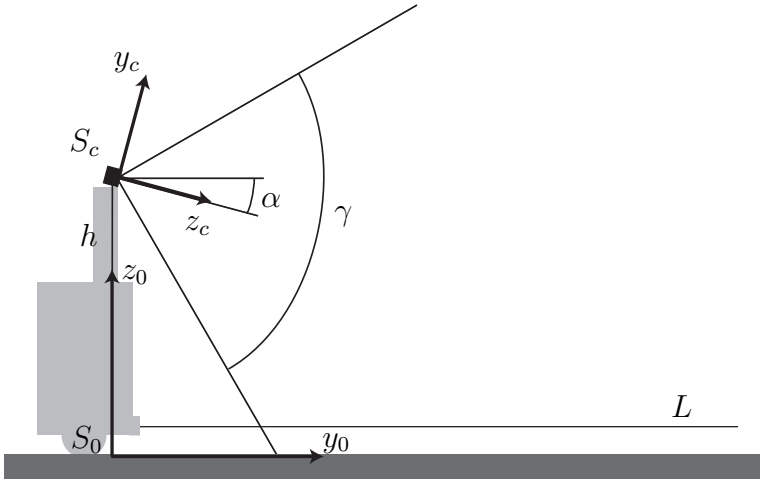


Fig. 4.4: Extrinsic and intrinsic camera parameters used for the back projection: the height h,the tilt angle α and the field of view γ.

system S_c, the reference system S_0, the cameras field of view γ, and the other camera parameters necessary for the computation. The extrinsic parameters needed for the projection are the height h and the tilt angle α, while the required intrinsic parameter are the the focal length f, the corresponding field of view γ and the chip size l. Some of the intrinsic parameters are related:

$$\gamma = 2 \arctan \frac{l}{2f}. \tag{4.4}$$

To simplify the computation, the assumption that the tilt angle α is small can be made. A virtual image point d'' is introduced, which can be computed for every image point d' measured by the camera. d' can be estimated by using the camera size l_x and l_y, the pixel coordinates p_x and p_y, as well as the pixel size s_p:

$$d'' = \frac{d'}{\cos(\alpha)} = \frac{(ps_p)/l}{\cos(\alpha)}. \tag{4.5}$$

By using the intercept theorem, Equation 4.6 computes the real distance d_y of the virtual image point d''_y. As $h >> d''_y$, Equation 4.6 can be simplified:

$$d_y = \frac{f(d''_y + h)}{d''_y}, \tag{4.6}$$

$$d_y = \frac{fh}{d''_y} = \frac{fh\cos(\alpha)}{(p_y s_p)/l_y}. \tag{4.7}$$

The distance d_x between the robots axis and the object point d is depending on the distance d_y:

$$d_x = d_y \frac{s_p p_x}{l_x}. \tag{4.8}$$

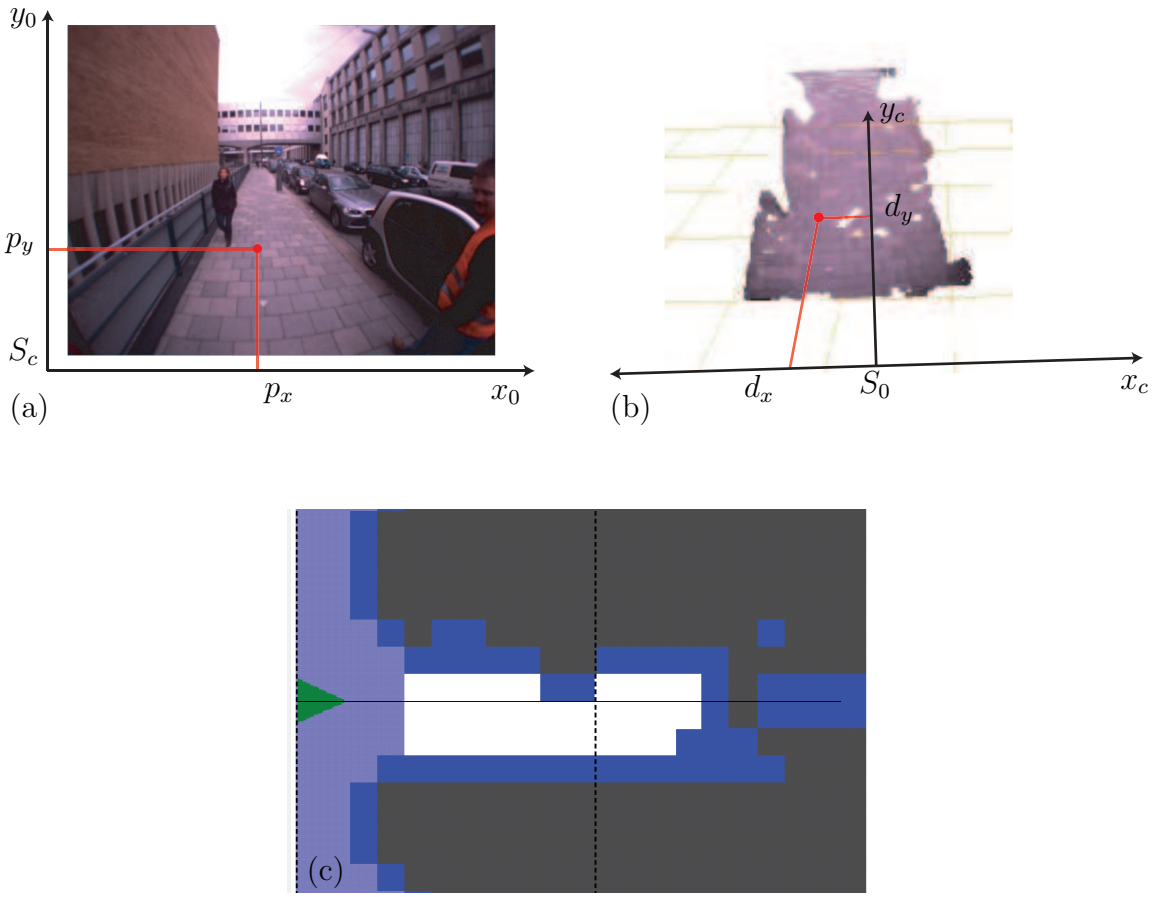


Fig. 4.5: Illustration of the different coordinate systems: (a) orientation of the camera coordinate system S_c, the pixel $\mathbf{p}$, (b) the corresponding reference coordinate system S_0 as well as the object point $\mathbf{d}$, and (c) the resulting grid with a cell size of 45 cm.

The correspondence between the pixel coordinates p_x and p_y, and the reconstructed object points d_x and d_y is illustrated in Figure 4.5 (a) and (b). After the real position was computed for every pixel of the detected ground, a map containing the ground can be built. A result of the back-projection algorithm can be found in Figure 4.2 (c). By accessing the mobile platform's odometry or the ego-motion estimation, several maps of consecutive images can be stored in a bigger map. To store the map in the memory, a 2.5D map, a so-called *occupancy grid* is used. In order to reduce the size, the grid is composed out of cells having a fixed size s_c. Each cell has an assigned probability, stating whether the cell is containing a valid ground or not. A free cell will have the probability 0, a blocked cell 1. If the algorithms detects an obstacle in a cell that is assigned free, the probability will be increased by a certain value. If a blocked cell is detected as free, the probability will be decreased. This occupancy grid can be used for the classification of the ground. For further, more complex processing, the obtained map could be combined with the map obtained by the robot's SLAM submodule or SLAM techniques could be used to enhance the map. Figure 4.5 (c) shows the map created out of the back projection of Figure 4.5 (a). The robot is illustrated by the green triangle, while the light blue area shows regions, which

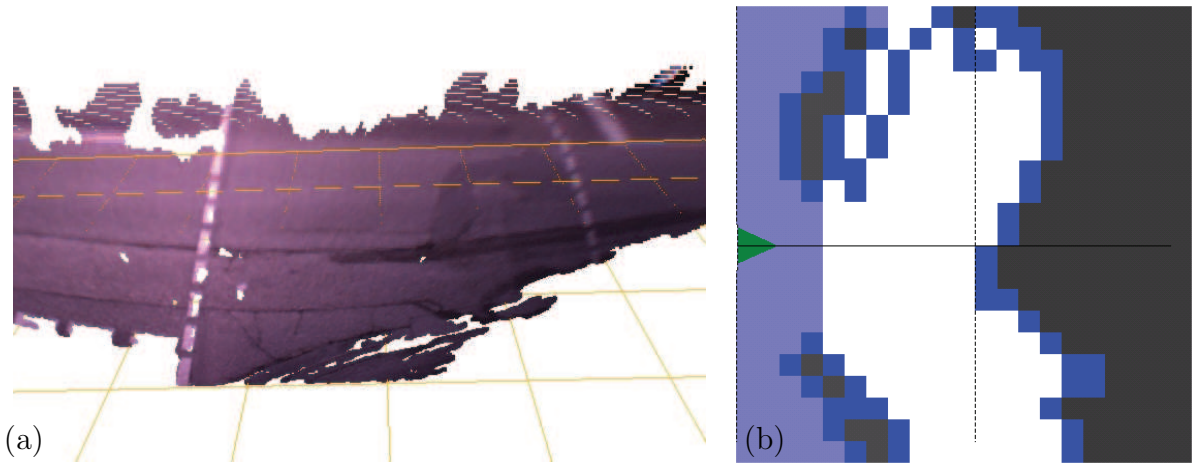


Fig. 4.6: (a) Reconstruction of a crossroad and (b) the corresponding occupancy grid.

are invisible to the robot's camera. White spots illustrate a probability of 0, meaning the algorithm has found a valid ground while dark areas illustrate a probability of 1, meaning there is no valid ground.

Classification of the Terrain

The last step is the classification of the terrain. During the experiments with the ACE robot, this method was used to estimate whether the robot is driving on the right side of the street or on the left side. As the robot is asking for directions, this provides useful information for processing the input. As traffic signs are sometimes hidden for the robot, another important application is a second possibility to detect crossings.
A correlation function is used to classify the type of the terrain. For the application on the ACE robot, the algorithm is trained to distinguish between five types: *left side*, *right side*, *middle*, *crossroad* or *free space in front* and *unknown*. For each of the first four types a reference map has been created. Each reference map is composed out of 20 measured maps of the corresponding type. This typical maps have been selected manually. In the next step, a correlation term is computed with each reference map. If the best correlation value is higher than a heuristically selected threshold, the hypothesis for the corresponding type is increased and the hypothesis of the remaining types are decreased. If all values are smaller than the threshold, all hypothesis except the hypothesis *unknown*, which is increased, are decreased. The hypothesis with the highest value will be returned as detected type of sidewalk. A back projected crossroad is illustrated in Figure 4.6 (a). ACE is standing on a sidewalk and is orientated towards the street. The corresponding occupancy grid can be found in Figure 4.6 (b). This scene was used as one of the typical types to learn the reference type *crossroad*.
This algorithm is not limited to the application on ACE, but can easily be extended to other terrains. By learning different, more complex reference types, the algorithm could distinguish between more complicated types of the terrain.

A real-time capable two-dimensional mapping algorithm was proposed in this section. In contrast to existing algorithms, it is equipped with a memory and is thus able to remember older valid terrains. However, the algorithm is not able to detect all types of obstacles and two-dimensional maps are insufficient for complex manipulation tasks. Consequently, a three-dimensional mapping algorithm is presented in the following section.

4.3 Three-Dimensional Mapping

Knowing objects in the robot's field of view is useless for complex manipulation tasks without any knowledge of their exact positions and without knowledge about the environment's structure. Hence, building three-dimensional maps of the environment is a crucial component of a cognitive system. The most common approach is the use of laser rangefinders mounted on a pan-tilt platform, providing an ideal base for a three-dimensional SLAM algorithm. On the other hand, laser rangefinders come with some disadvantages. For instance, they cannot be mounted easily on a biped walking robot. Moreover, three-dimensional SLAM requires expensive hardware and extensive data processing. On the contrary, a stereo camera system is a convenient alternative, allowing both two-dimensional and three-dimensional mapping. This section presents a fast implementation of an ego-motion estimation algorithm, which is computing the motion directly in the disparity maps [51]. Hence, it can use temporary results of the stereo algorithm. Furthermore, a modular approach of the median-fusion algorithm is presented, which is used to merge different point clouds.

4.3.1 System Architecture

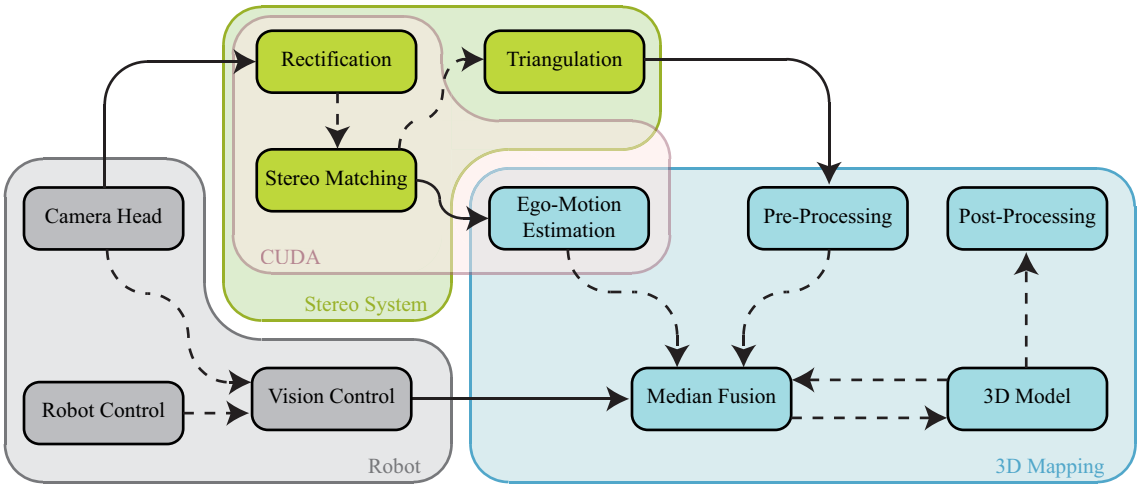


Fig. 4.7: Software architecture of the scene reconstruction module with both position estimation based on the robots odometry and based on the ego-motion estimation module.

Figure 4.7 shows the software architecture and the main modules and submodules required for three-dimensional mapping, while Figure 4.8 shows the different processing steps. The

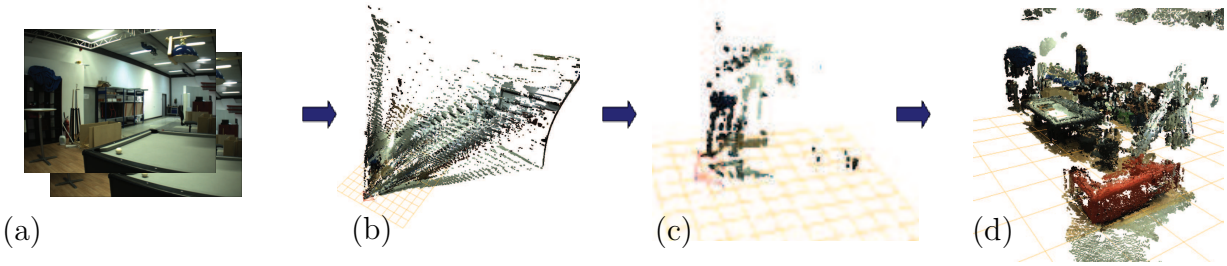


Fig. 4.8: Processing steps of the scene reconstruction module: (a) the stereo images, (b) the point cloud before and after (c) the post-processing, and (d) the merged point cloud.

modules **Camera Head** and **Robot Control** transmit the orientation of the head and the robot or to the **Vision Control** module, where the absolute position of the head is computed. If no SLAM module or laser rangefinder is available, the ego-motion estimation can be used instead. A main part is the **Stereo Module**, where the images arerectified before the actual disparity map is computed in the **Stereo Matching** submodule. Two different stereo matching submodules are available, the presented CUDA implementation or an alternative implementation based on the OPENVIS-library [104], which mainly focusses on quality. As it provides a higher resolution, only the Bumblebee X3 camera is used for three-dimensional mapping, while the other camera is used for two-dimensional mapping and other tasks like human tracking. In the **Triangulation** submodule, this disparity map is transformed into a point cloud. Afterwards this point cloud is then transmitted to the **Pre-Processing** submodule, where noise reduction and the removal of outliers is performed. In the next step, the actual **Median Fusion** algorithm is performed, followed by a **Post-Processing** step. The resulting three-dimensional model is stored as a point cloud.

The interfaces between the different modules and submodules have been designed carefully. Besides supporting different stereo modules and cameras, the resolution of the images can be chosen arbitrarily and the input point clouds can consist of both colored points as well as grayscale points. They can be created by stereo triangulation based on a depth map. By supporting drivers for laser rangefinders, the stereo module can be replaced with an adequate laser scanner. Each point cloud provides the corresponding position and orientation of the camera head.

The following sections describe the main modules and submodules, the ego-motion estimation based on stereo vision and the actual median fusion algorithm. The method used for the reduction of noise and outliers has been described in the pre-processing step in Section 3.3.1. A suitable trade-off between quality and a sufficient number of remaining points has to be determined experimentally. Input and output of the pre-processing submodule is a single point cloud, whereas the median fusion algorithm requires a set of point clouds as input and delivers a single point cloud as output. New point clouds can be added iteratively to the resulting model.

4.3.2 Stereo Vision-Based Ego-Motion Estimation

Ego-motion estimation algorithms require temporary result of calculations, which have already been made by the stereo module. Due to the hard time and quality constraints, using this pre-computed data is mandatory. This data includes the result of the edge detection algorithm, which is used to compute the corners of the images. As they provide a good texture and are thus easy to track, only corners are used for the ego-motion estimation. Therefore, the corners of the current and previous image are computed by using the *good features to track* algorithm presented by Shi et al. [133], tracked, and then used to compute the movement of the objects relatively to the camera. Afterwards, wrong correspondences can be removed and the actual movement of the camera and thus the robot can be estimated. The main contributions of this algorithm include the fast implementation on CUDA, the strong interconnection with the stereo algorithm, the use of temporary results, and the removal of invalid correspondences [153]. Future hardware architectures will provide more memory and computational power, so the parameters of the presented algorithms have to be adjusted to utilize the full potential of the hardware.

Computation of the Corresponding Corners

The computation of the optical flow is based on the assumption that the intensity of a pixel remains constant in two consecutive images. Like in stereo reconstruction, the whole problem of ego-motion estimation can be stripped down to the computation of the two corresponding corners in the current and the previous image. Again, the algorithm can easily be parallelized. However, the search problem cannot be reduced to an one-dimensional one. Hence, a cost function has to be computed for all possible matches by using block matching with a window size of $N_{\text{Search}} \times N_{\text{Search}}$, with $N_{\text{Search}} = 2m + 1$:

$$C(i,j,k,l) = \sum_{u=-m}^{m} \sum_{v=-m}^{m} |V_{\text{Previous}}(i+u, j+v) - V_{\text{Current}}(i+u+k, j+v+l)|, \qquad (4.9)$$

while $V_{\text{Previous}}(i,j)$ denotes a corner in the previous image and $V_{\text{Current}}(i+k, j+l)$ a possible match in the current image. k and l are limited to a search window of size $N_{\text{Ego}} \times N_{\text{Ego}}$, which is determined by the robot's speed. As small disparities will lead to a small movement of the corner and large disparities will lead to a large movement, N_{Ego} is depending on the disparity of the current corner $d(i,j)$:

$$N_{\text{Ego}} = \sigma \cdot d(i,j) + r_{\text{const}}, \qquad (4.10)$$

while the parameters σ and r_{const} have to be adjusted heuristically. The corresponding corner $V_{\text{Current}}(i + \partial i, j + \partial j)$ can be found by minimizing the cost function with respect to k and l:

$$\partial i = \underset{k}{\operatorname{argmin}}(C(i,j,k,l)) \ , \quad \partial j = \underset{l}{\operatorname{argmin}}(C(i,j,k,l)). \qquad (4.11)$$

The computation time can be reduced by only computing valid corners.

Removal of Invalid Correspondences

Invalid correspondences can be removed by computing the median movement in a circular area with radius r_{Min} around each corner. If the movement of a corner in this area differs from the median movement, the probability of an error is high and the corner is discarded. However, a minimal number of correspondences has to remain in the window. The minimal number of correspondences and the threshold have to be estimated heuristically.

Computation of the Camera Motion

The movement of the camera and thus the movement of the robot can be computed directly in the disparity space, without transforming the disparity into cartesian coordinates. Therefore the previous and current pixel positions are transformed according to the robot's movement. This can be computed for every pair of corresponding corners and the resulting overdetermined system of equations can be solved by using the method of least squares:

$$\begin{pmatrix} \alpha \\ {}_c t_x \\ {}_c t_z \end{pmatrix} = (\mathbf{D}^T \cdot \mathbf{D})^{-1} \cdot \mathbf{D}^T \cdot \mathbf{p} \quad \text{, with} \tag{4.12}$$

$$\mathbf{D} = \begin{pmatrix} -(x_1 + \frac{f^2}{\hat{x}_1}) & -\frac{f\delta_1}{B\hat{x}_1} & \frac{\delta_1}{B} \\ -x_1 & 0 & \frac{\delta_1}{B} \\ -x_1 & 0 & \frac{\delta_1}{B} \\ \vdots & \vdots & \vdots \\ -(x_n + \frac{f^2}{\hat{x}_n}) & -\frac{f\delta_n}{B\hat{x}_n} & \frac{\delta_n}{B} \\ -x_n & 0 & \frac{\delta_n}{B} \\ -x_n & 0 & \frac{\delta_n}{B} \end{pmatrix}, \quad \mathbf{p} = \begin{pmatrix} f(\frac{x_1}{\hat{x}_1} - 1) \\ f(\frac{y_1}{\hat{y}_1} - 1) \\ f(\frac{\delta_1}{\hat{\delta}_1} - 1) \\ \vdots \\ f(\frac{x_n}{\hat{x}_n} - 1) \\ f(\frac{y_n}{\hat{y}_n} - 1) \\ f(\frac{\delta_n}{\hat{\delta}_n} - 1) \end{pmatrix}. \tag{4.13}$$

f denotes the focal length, B the stereo basis, x_i and y_i the position of the i-th corner in the current image, and $\hat{x}_i$ and $\hat{y}_i$ the position of the i-th corner in the previous image, respectively. The movement of the camera is denoted by the rotation α and the translations ${}_c t_x$ and ${}_c t_z$. After this computation, the movement of the camera has to be transformed into the movement of the robot. A detailed mathematical background can be found in Appendix C.
The presented ego-motion estimation can replace the robot's odometry. Experimental results have shown that the algorithm is prone to bad light conditions and fast rotations. Hence, a further correction step is desirable.

4.3.3 Median Fusion Algorithm

Most of the noise and outliers can be identified and removed during the pre-processing. Since this step affects only single depth maps, it is still possible to reduce the amount of erroneous data even further, by combining measurements from several different viewpoints.

Algorithm 4.1 Median fusion algorithm

```
1: Transform all depth maps f_i, with i = 1...n in the reference perspective P_a
2: Store nearest point in F_a(x)
3: for i = 1 to i = n - 1 do
4:    S(x) = 0
5:    N(x) = 0
6:    for k = 1 to k = n do
7:       transform f_i into P_a
8:       Store nearest point in O(x)
9:       for all Points x_a in P_a do
10:         if O(x_a) ≤ F(x_a) then
11:            S(x_a) = S(x_a) + 1
12:         else
13:            N(x_a) = Min(N(x_a), O(x_a))
14:            Transform x_a into P_k
15:            if x_k is a regular point in P_i then
16:               S(x_a) = S(x_a) − 1
17:            end if
18:         end if
19:      end for
20:   end for
21:   for all Points x_a in P_a do
22:      if S(x_a) < 0 then
23:         F_a(x_a) = N(x_a)
24:      end if
25:   end for
26: end for
```

For certain types of errors occurring during the reconstruction process, in particular the quantization error, it is impossible to reach better results relying on measurements from merely one perspective. Next to the error correction, the minimization of redundancy is another important task. Furthermore, errors from the ego-motion estimation can be reduced. Generating a multitude of depth maps from nearby viewpoints typically yields large amounts of redundant data points. While being useful when identifying reconstruction errors, they have to be removed in order to reach a memory efficient representation. An algorithm for depth map fusion that is capable of further resolving inconsistencies as well as reducing redundancy is the median fusion algorithm presented by Nister et al. [98]. The algorithm uses a given set of n depth maps to compute an optimized fused depth map by using one of the input depth maps as reference viewpoint. f_i denotes a depth map, P_i the corresponding viewpoint and $\mathbf{x}_i$ a point of the depth map and accordingly the corresponding point cloud $\mathbb{N}_i$. The index a indicates the reference view with the depth map f_a, the viewpoint P_a, the point cloud $\mathbb{N}_a$, and a point $\mathbf{x}_a$. Combining multiple datasets into a single one typically results in several possible depth values for every pixel in the reference view. Hence, an optimization criterion is introduced, which is defined in order to resolve ambiguities. Based on two dual visibility-relations between measured points, i.e. depth

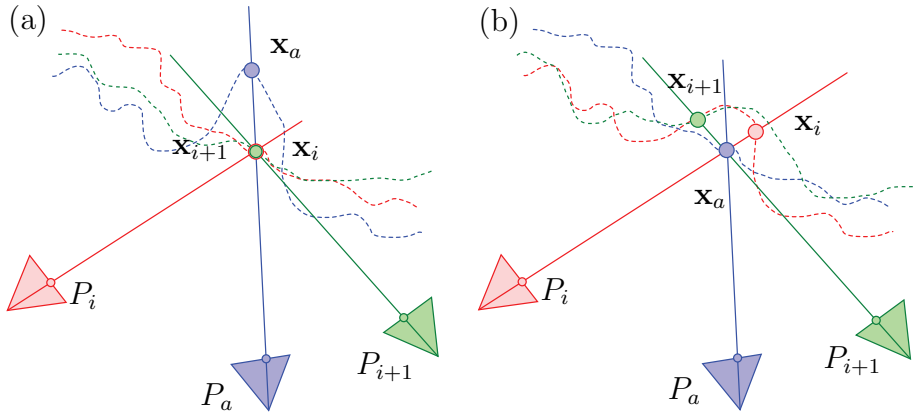


Fig. 4.9: Illustration of the median fusion algorithm with the two error models, (a) the **direct occlusion** and (b) the **indirect occlusion**.

values, a stability measure for the depth value is estimated for each pixel in the reference view. A point observed in the reference view $\mathbf{x}_a$ is assumed to be occluded if there is at least one other point $\mathbf{x}_i$ in another depth map of the input set that lies in shorter distance than this point (observed from the reference viewpoint). This error is called **direct occlusion**. On the contrary, the **indirect occlusion** is defined as a point $\mathbf{x}_a$ observed in the reference view that is violating another point's $\mathbf{x}_i$ free-space by occluding that point when observed from another viewpoint.
Obviously, for low depth values occlusions occur less frequently while violations of free-space constraints are more likely. For large depth values the opposite effect can be observed. This leads to the definition of a stability measurement that accumulates and then subtracts the number of occurrences of occlusions and free-space violations for each depth value under investigation. Finally, since both events reveal inconsistencies between multiple depth maps, for each pixel in the reference view that value is chosen, which yields an equilibrium between the two visibility violations, for instance a stability of close to zero.
Figure 4.9 illustrates the two error types and the stability criterion. In Figure 4.9 (a) the point $\mathbf{x}_a$ is occluded by the two points $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$. A high probability that the estimated point $\mathbf{x}_a$ is too far away from P_a. In Figure 4.9 (b) $\mathbf{x}_a$ occludes $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$ yielding a high probability that $\mathbf{x}_a$ is too close to P_a. The corresponding algorithm is shown in Algorithm 4.1 and the resulting point cloud is stored in $\mathbb{F}_a$. $O(\mathbf{x})$ denotes a temporary memory containing results of the stability analysis and $N(\mathbf{x})$ contains the depth-value, if the current value is proven to be stable.

4.3.4 Obstacle Detection in Three-Dimensional Maps

In a last step, the three-dimensional maps can be used for obstacle detection allowing to detect obstacles, which cannot be found by the two-dimensional mapping module. For instance, overhanging objects like tables cover only small parts of the ground.
To perform the obstacle detection, the space is divided into several regions of a fixed size of $20 \times 20 \times 20$ cm. Figure 4.10 (b) shows a point cloud derived from a stereo image with

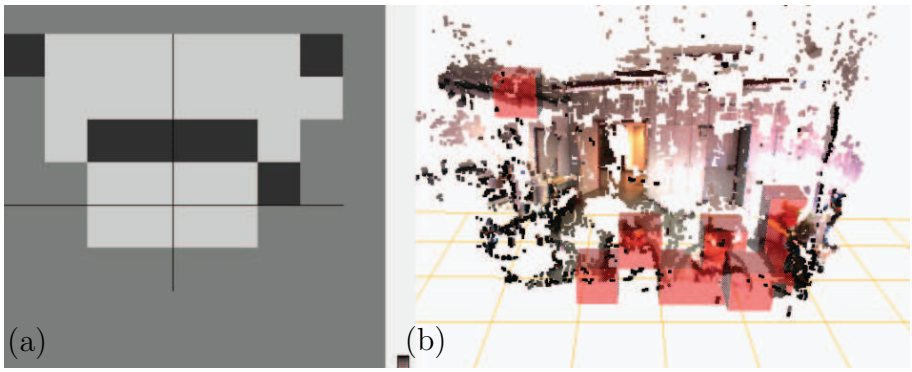


Fig. 4.10: Obstacle detection in 3D Maps. (a) shows the resulting occupancy grid and (b) the scene, while detected obstacles are indicated by red boxes.

several persons standing near the robot. The point density is computed for each space region and when it exceeds a threshold, the corresponding box is assumed to be obstructed. Otherwise it is assumed to be free of obstacles. In the last step a two-dimensional occupancy grid can be computed. Therefore the height h of the robot must be known and the occupancy grid should have the same resolution as the space regions. Each cell of the occupancy grid at the position (x, y) is influenced by several regions with the position $(x, y, \mathbf{z})$, where $\mathbf{z}$ contains all values between 0 and h: $\mathbf{z} = [0, \ldots, h]$. If one region is blocked, the whole cell is assumed to be blocked and thus assigned with a probability of 1. If no region is blocked, the cell will be assigned with a probability of 0. Those regions not in the robots field of view are assigned with a probability of -1 denoting unknown regions. To create larger maps, several occupancy girds can be merged. Furthermore, the resulting map can be merged with the map obtained by the two-dimensional mapping module. Compared to most other obstacle detection algorithms, the presented algorithm is able to detect floating objects, which are not covering the ground. Furthermore, it is fast to execute and is independent from the used sensor.

Equally important as the two-dimensional mapping module, all necessary modules to compose a three-dimensional mapping system have been presented. Those modules include the ego-motion estimation, which is strongly connected to the stereo module, a modular implementation of the median fusion algorithm, and an obstacle detection algorithm. The presented methods are based on stereo vision, but the median fusion algorithm and the obstacle detection can be also applied to other types of sensors. As laser rangefinders provide a much higher spatial resolution, it is obvious to combine the advantages of laser rangefinders with stereo vision.

4.4 Mapping with Different Sensor Types

Stereo reconstruction provides color information, but come with a low spatial resolution of only a few centimeters. On the other hand, laser rangefinders provide no color information, but the spatial resolution is higher by one order of magnitude. Hence, a combination of laser rangefinders and stereo images yields a colored representation with a high resolution. Due to the different resolutions and point densities of laser rangefinders and stereo images, the median fusion algorithm is not suited for the fusion of stereo and laser data. As both point clouds have a large overlap, the ICP algorithm seems to be well suited. However, it is not able to register the two point clouds, as it tends to run into local minimums. Consequently, a novel genetic ICP algorithm is proposed, performing a genetic mutation whenever a local minimum is detected [90].
The area seen by one laser rangefinder may be too small, so it is beneficial to use two or more, each providing one point cloud. Besides, using two laser rangefinders will yield to a higher accuracy. To create a single accurate point cloud, the single point clouds have to be merged. Therefore, the exact position and orientation of each laser has to be know. As the setup of the lasers is changed frequently, the calibration progress should be automated. A fast, easy-to-use and reliable algorithm is proposed to perform the calibration of most of the extrinsic parameters. The intrinsic parameters are depending on the configuration of the range-finders and are assumed to be known. The algorithm scans an object with a known height to calculate the following parameters:

- The absolute y and z position of the scanner.
- For one laser, the x position has to be known. The x position of the other range-finders will be computed relatively to this position.
- The absolute yaw- (Ψ), pitch- (Θ) and roll (Φ) angles of the orientations around the z-, x- and y-axis.

The extrinsic parameters and the coordinate systems are shown in Figure 4.11. One coordinate system is attached to each sensor. S_0 denotes the reference system, which is attached to the robot and is located in the robot's center of rotation. The robot rotates around the z-axis and drives in direction of the y-axis. Figure 4.12 shows the setup of a mobile robot R with the two laser rangefinders L_1 and L_2. Both scanners have a Φ of 45° or −45° with respect to the reference system. The areas A_1 and A_2 denote the scanning area. When the mobile Robot is rotated around the z-axis, the whole calibration object C can be scanned. Furthermore, the stereo camera S is depicted.

4.4.1 Requirements

Before the calibration is performed, an initial guess for Φ of each laser rangefinder should be made. This will decrease the execution time and avoid absurd results. An accuracy of about ±45° is sufficient for the initial guess. Furthermore, the x-position of the first laser should be known. Inaccurate results may occur otherwise. The calibration object must be scanned using all laser rangefinders, its height h_O must be known, the base point of the

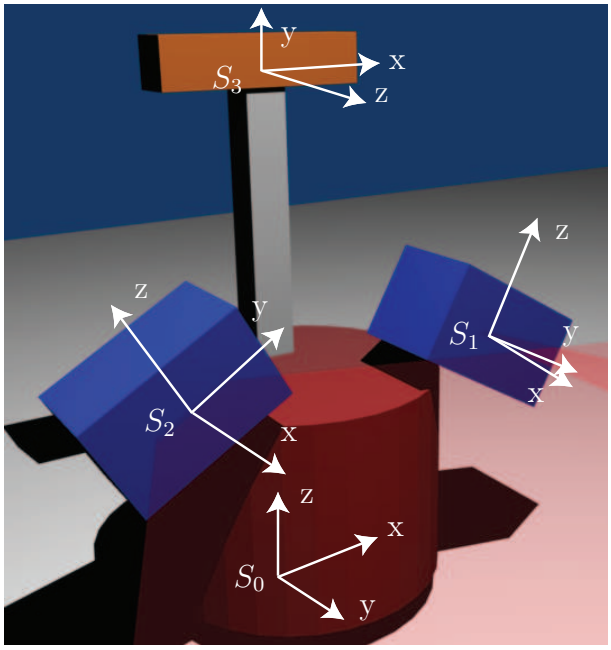


Fig. 4.11: The different coordinate systems needed for calibration, S_1 and S_2 for the left and and right laser rangefinder, S_c for the camera, and S_0 for the reference system. The yaw angle Ψ is rotated around the z-axis, the pitch angle Θ around the x-axis and the roll angle Φ around the y-axis, respectively.

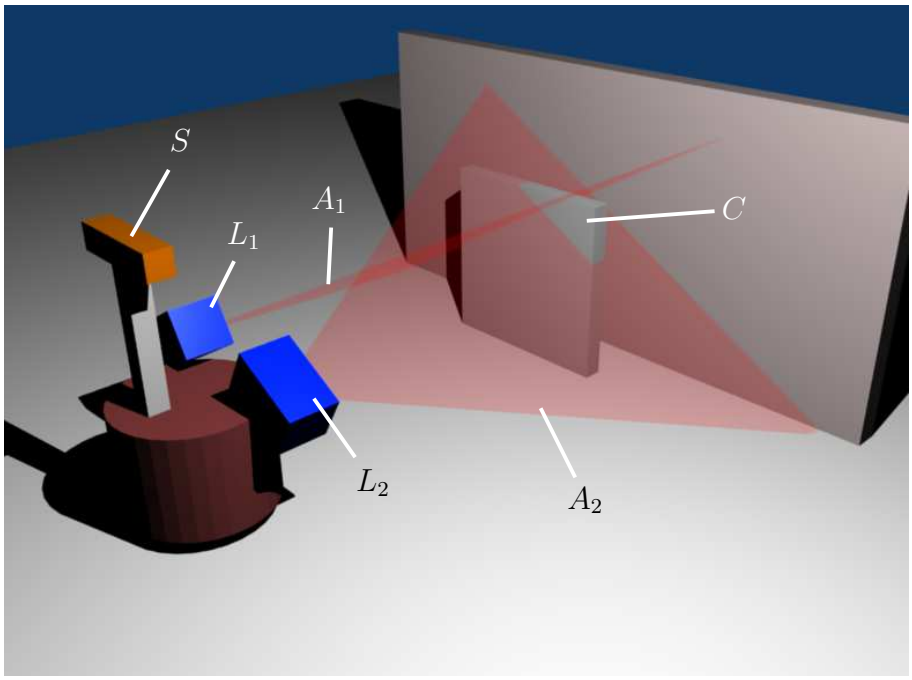


Fig. 4.12: Setup of a calibration-scene with the camera S, the two laser rangefinders L_1 and L_2, the corresponding scanning areas A_1 and A_2, and the calibration object C.

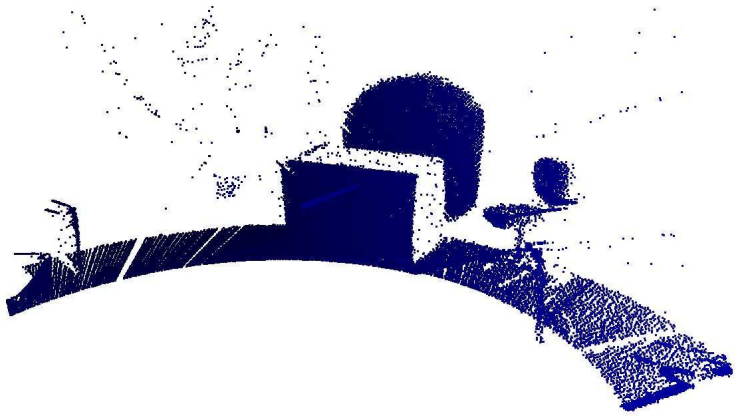


Fig. 4.13: Scan of the calibration-scene.

object must lay on the floor (with $z = 0$), and the floor has to be flat, so that no scans below the $x/y-$plane can occur. After the scan has been performed, the object should be extracted manually from the single laser scans. Figure 4.13 shows the laser scan of the calibration scene described above. The calibration object can be seen in the center of the picture.

4.4.2 Calibration of the Laser Rangefinders

Algorithm 4.2 Calibration

```
for each scan do
  while |z_Min| > ε and ||z_Max − z_Min| − h_O| > ε do
    Calculate z_Min and z_Max of the scanned object
    if z_Min < 0 + ε then
      increase z
    end if
    if z_Min > 0 − ε then
      decrease z
    end if
    if (z_Max − z_Min) < h_0 + ε then
      increase Φ
    end if
    if (z_Max − z_Min) > h_0 − ε then
      decrease Φ
    end if
  end while
end for
```

After the scans have been made, the algorithm is executed in two steps. As described in Algorithm 4.2, the z-position and Φ of the single laser rangefinders are computed iteratively in the first step. Changing these parameters will result in a sheared representation of the

scanned object and a wrong height. The algorithm is finished, when a minimal error ϵ is reached. In a real environment, ϵ should be in the same order of magnitude as the resolution of the laser rangefinders, so it is sufficient to set $\epsilon = 0.1$ mm.
After the single laser rangefinders have been calibrated, the positions of the scanners relative to a reference scanner are computed, using the first scanner with known x-position as reference. The position and orientation of the other scanners will be computed relative to the reference scanner. For the calculation of the x- and y-position, as well as Ψ and Θ, the presented genetic ICP algorithm (see Section 4.4.3) is used. If the y-position of the reference laser is not known, the y-axis will be placed between the laser rangefinders.

4.4.3 Registration of Vision and Laser Data

The ICP algorithm is widely spread for the registration of two point clouds. It estimates the relative translation and rotation between two point clouds $\mathbb{C}_1$ and $\mathbb{C}_2$ and is repeated in iterations. $\mathbb{C}_1$ and $\mathbb{C}_2$ have N_1 and N_2 points, respectively. It can be described in the following way:

1. For each point $\mathbf{p}_i^1$ in $\mathbb{C}_1$: allocate the closest point $\mathbf{p}_i^2$ in $\mathbb{C}_2$.
2. Compute $\mathbf{R}$ and $\mathbf{t}$ to minimize the median error $e(\mathbf{R}, \mathbf{t})$:

$$e(\mathbf{R}, \mathbf{t}) = \frac{1}{N_1} \sum_{i=0}^{N_1} ||\mathbf{p}_i^1 - (\mathbf{R} \cdot \mathbf{p}_i^2 + \mathbf{t})||^2. \tag{4.14}$$

3. Transform $\mathbb{C}_2$ with the calculated $\mathbf{R}$ and $\mathbf{t}$.
4. Iterate until $e(\mathbf{R}, \mathbf{t})$ converges.

Existing ICP algorithms work fine for similar point clouds with a large overlap and a good guess of the initial transformation. In an ideal case, it converges in less than 10 iterations. If the initial guess is not of sufficient quality, there is a risk that the algorithm converges to a local minimum. Another problem will occur, if the overlap of the two scans is not large enough. The algorithm has to determine, which points are included in both point clouds and use only those overlapping points for the registration. A reduced point cloud, containing only the overlapping points $\tilde{\mathbf{p}}_i^x, i \in 0...N_x$ is denoted as $\tilde{C}_x$.

Computation of the Transformation

The main challenge of ICP algorithms is the computation of $\mathbf{R}$ and $\mathbf{t}$. Since it has some advantages compared to a SVD (singular value decomposition) based algorithm in two- and three-dimensional spaces, the presented algorithm is based on a quaternion based ICP algorithm as described in [14] and [56]. A cross-covariance matrix is used to solve the least square problem and compute the transformation:

$$\underset{\mathbf{R},\mathbf{t}}{\operatorname{argmin}} = \frac{1}{\tilde{N}_1} \sum_{i=0}^{\tilde{N}_1} ||\tilde{\mathbf{p}}_i^1 - (\mathbf{R} \cdot \tilde{\mathbf{p}}_i^2 + \mathbf{t})||^2. \tag{4.15}$$

Only those points are considered, where an overlap is expected. $\tilde{\mathbb{C}}_1$ includes $\tilde{N}_1$ points, which fulfill the following relation:

$$0 \leq d(\mathbf{p}_i^1, \mathbf{p}_i^2) \leq \frac{d_{\text{Max}} - d_{\text{Min}}}{2}, \text{ with} \tag{4.16}$$

$$d(\mathbf{p}_i^1, \mathbf{p}_i^2) = ||\mathbf{p}_i^1 - \mathbf{p}_i^2||^2,$$

where d_{Max} denotes the maximal measured distance between two points and d_{Min} the minimal distance, respectively.

The Genetic ICP Algorithm

As a standard ICP algorithm is not able to align vision data and laser scans properly, a hybrid algorithm to find the global minimum of Equation 4.15 is proposed. The algorithm consists out of two main parts, a standard ICP algorithm and a genetic algorithm. Every time the standard ICP algorithm converges against a local minimum, the genetic algorithm is activated to find alternative solutions. The genetic algorithm performs both random mutations and reproduction of two individuals. Each individual is thereby described by six parameters: x-, y- and z-position as well as Ψ, Θ and Φ. Before the genetic ICP algorithm is executed, the point clouds are clustered. To remove small misplaced points, which do not belong to any object, clusters having less than a minimal number of points are discarded.
The algorithm is described in Algorithm 4.3, where i denotes the current generation and s the size of the population. In a first step, the overlap is computed for both point clouds and the standard ICP algorithm is applied. The genetic mutation will not be performed every generation, but only if the distance metric d_{i-1} converges, i.e. the difference between d_{i-1} and d_{i-2} falls below a threshold. Algorithm 4.3, lines 8 to 17, describes the genetic mutation (rand(x) returns a random integer between 1 and x, rand2(x) a floating number between $-x$ and x), where parameter $p = 1$ denotes the x-position, $p = 2$ the y-position, and so on. By changing the allocation between p and a parameter, the algorithm can easily be limited to search in an arbitrary number of dimensions. Furthermore, it is possible to lock arbitrary parameters.
The distance metric d_i for the iteration i is computed after each transformation of $\mathbb{C}_1$. If d_i is smaller than the previous best distance metric d_{Best}, the transformation $\mathbf{R}_i$ and $\mathbf{t}_i$ will be stored and added as a new individual to the population. If the maximal number of individuals in the population is reached, the oldest individual will be replaced. If d_i is larger than d_{Best}, the transformation $\mathbf{R}_i$ and $\mathbf{t}_i$ will be discarded. To minimize the mean square error while maximizing the number of points used for the matching, the distance metric has to be chosen deliberately:

$$d_i = (2 - (\frac{\tilde{N}_1}{N_1})^2) \cdot \left(1 + \left(\frac{1}{\tilde{N}_1} \sum_i^{\tilde{N}_1} d(\mathbf{p}_i^1, \mathbf{p}_i^2) \right)^{\frac{1}{2}} \right), \tag{4.18}$$

Algorithm 4.3 Genetic algorithm

reset d_{Best}
$i = 0$
while $i < i_{\text{Max}}$ **do**
  $i++$
  Compute $\tilde{\mathbb{C}}_1$, $\tilde{\mathbb{C}}_2$ and corresponding points
  Compute $\mathbf{R}_i$, $\mathbf{t}_i$ with ICP
  if $|d_{i-2} - d_{i-1}| < \epsilon$ **then**
    $v = \text{rand}(s)$
    load individual I_v
    $w = \text{rand}(s)$
    load individual I_w
    $p = \text{rand}(6)$
    substitute parameter p of I_v with parameter p of I_w
    $p' = \text{rand}(6)$
    $f = \text{rand2}(f_{\text{Max}})$
    increase parameter p' of I_v by f
    load $\mathbf{R}_i$ and $\mathbf{t}_i$ from individual I_v
  end if
  transform $\mathbb{C}_1$ with $\mathbf{R}_i$ and $\mathbf{t}_i$
  Compute distance measurement d_i
  if $d_i < d_{\text{Best}}$ **then**
    save $\mathbf{R}_{\text{Best}}$ and $\mathbf{t}_{\text{Best}}$
    $d_{\text{Best}} = d_i$
    if $s < s_{\text{Max}}$ **then**
      add $\mathbf{R}_i$ and $\mathbf{t}_i$ as individual I_i to population
      $s++$
    else
      remove oldest individual from population
      add $\mathbf{R}_i$ and $\mathbf{t}_i$ to population
    end if
  else
    discard $\mathbf{R}_i$ and $\mathbf{t}_i$
  end if
  Return $\mathbf{R}_{\text{Best}}$ and $\mathbf{t}_{\text{Best}}$
end while

while $\tilde{N}_1$ denotes the number of points used for matching. The relative number of points used is denoted as $\frac{\tilde{N}_1}{N_1}$. The radicand denotes the median distance d_m between all pairs of corresponding points. Figure 4.14 shows a plot of the distance metric. The absolute minimum is placed at $\frac{\tilde{N}_1}{N_1} = 1$ and a median distance of 0. If one of these values is increased, the resulting distance will increase as well. Increasing both of them, will lead to the absolute maximum of the distance metric. By changing the exponents, the weight can be shifted between emphasizing the number of points used or the median distance.

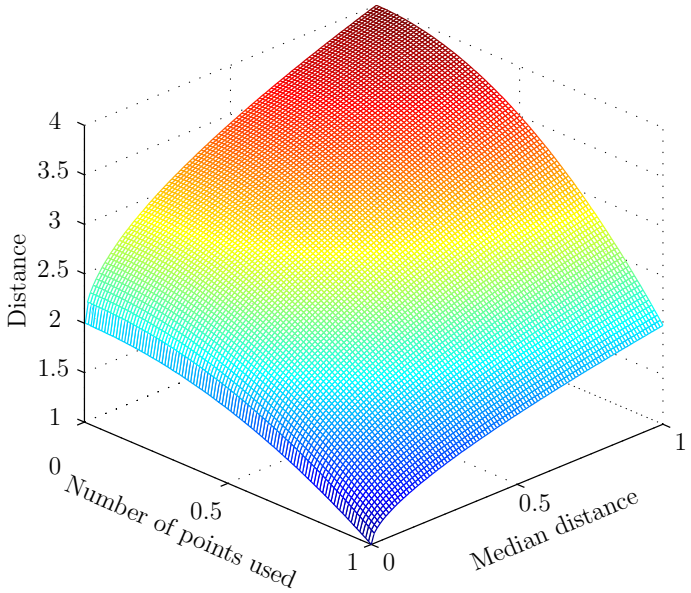


Fig. 4.14: Plot of the distance metric d_i depending on the median distance d_m and the relative number of points used $\frac{\tilde{N}_1}{N_1}$.

Fusion of Vision and Laser Data

After the transformation has been computed and the point cloud has been transformed, the point clouds have to be merged. $\mathbb{C}_V$ denotes the point cloud of the stereo vision module and $\mathbb{C}_L$ the point cloud obtained by the laser rangefinders. $\mathbb{C}_V$ and $\mathbb{C}_L$ are aligned properly. To merge $\mathbb{C}_V$ and $\mathbb{C}_L$ into $\mathbb{C}_M$, it is adequate to search for every point $\mathbf{p}_j^L$ in $\mathbb{C}_L$ that point $\mathbf{p}_j^V$ in $\mathbb{C}_V$, which has the smallest distance to $\mathbf{p}_j^L$. The resulting point $\mathbf{p}_j^M$ will have the position of point $\mathbf{p}_j^L$ and the color of point $\mathbf{p}_j^C$. Moreover, a standard distribution has been selected for the probability function $f_j(d_j)$, which has to be calculated and stored for each point in $\mathbb{C}_M$:

$$f_j(d_j) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}(\frac{d_j}{\sigma})\right), \tag{4.19}$$

with the distance $d_j = d(\mathbf{p}_j^L, \mathbf{p}_j^V)$ between $\mathbf{p}_j^L$ and $\mathbf{p}_j^V$. σ is the standard deviation and has to be determined experimentally depending on the quality of the vision data. The size of $\mathbb{C}_M$ will be equal to the size of $\mathbb{C}_L$. In a last step, the probability $f_j(d_j)$ can be used as a confidence metric and thus to eliminate uncertain measurements.

As using laser data allows increasing the spatial resolution of stereo vision, the fusion of different sensors completes a sophisticated mapping system. The novel algorithm used for the fusion is derived from a quaternion based ICP and a genetic algorithm. Whenever the ICP algorithm runs into a local minimum, a genetic mutation is performed and the minimum can be avoided. A thorough experimental investigation of the ego-motion estimation algorithm, the two- and three-dimensional mapping, and the fusion of different sensors will be given in the next section, together with details about the evaluated parameter settings.

4.5 Experimental Results

This section shows some experimental results from the main modules of the vision-based mapping subsystem, starting with the two-dimensional mapping. The modules of the three-dimensional mapping module will be presented individually, starting with the ego-motion estimation, followed by the fusion algorithms for different point clouds and the sensor fusion.

4.5.1 Experimental Setup

The following experiments have been conducted using the vision processing PC on the ACE robot, which was equipped with an AMD Phenom Quad-Core CPU running at 2.5 GHz, 4 GB of physical memory, and two GeForce 9800 GX2 cards and hence four Cuda enabled devices. Unless stated otherwise, only one Cuda device has been used for a module. In addition, the robot is equipped with two cameras, a bumblebee X2 with a resolution of 640×480 and a field of view of 97°, which is used for sensor fusion and two-dimensional mapping, and a bumblebee X3 camera with a resolution of 1280×960 pixels at a field of view of 66°, which was used for the three-dimensional mapping.

4.5.2 Two-Dimensional Mapping

The presented algorithm is able to detect and classify the ground. To achieve a good compromise between false positives and misses, the threshold t_m was heuristically selected to 80. With this value, the algorithm is able to detect small obstacles on the ground. On the other hand, if t_m is too large, the algorithm won't be able to detect a terrain with a varying texture. The number of histograms N_B was selected as 500. When driving with a speed of 1 $\frac{\text{m}}{\text{s}}$ and a camera frame rate of 5 Hz, the robot is able to remember the texture for 100 m. When a larger frame rate is used, it is advisable to increase N_B.

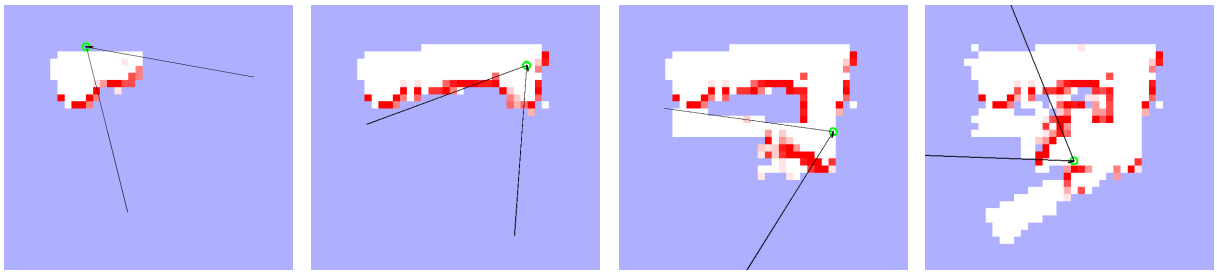


Fig. 4.15: Result of the mapping algorithm over a long distance.

Figure 4.15 shows, how an occupancy grid with a size of 10×8 m is created in several steps. This map was built using only the presented method for back-projection and the robot's odometry module. Using existing SLAM techniques enables loop-closing and will lead to better results after long distances.

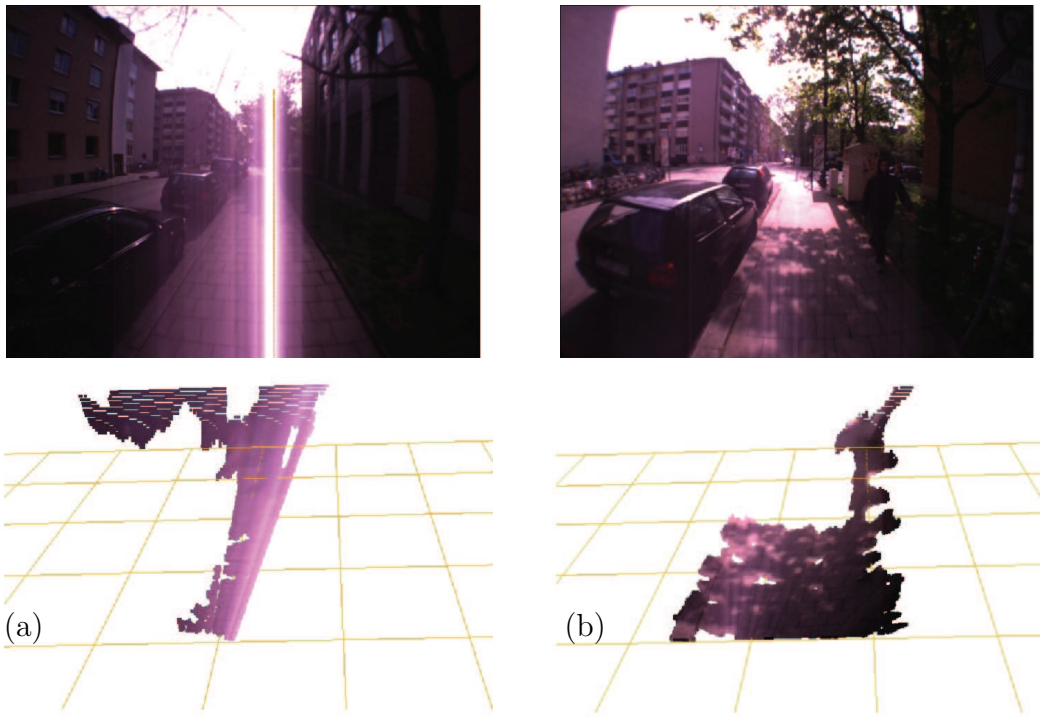


Fig. 4.16: Effect of bad light conditions: (a) the sun is shining directly in the camera and generating blooming and (b) abrupt transition between shadow and illuminated area.

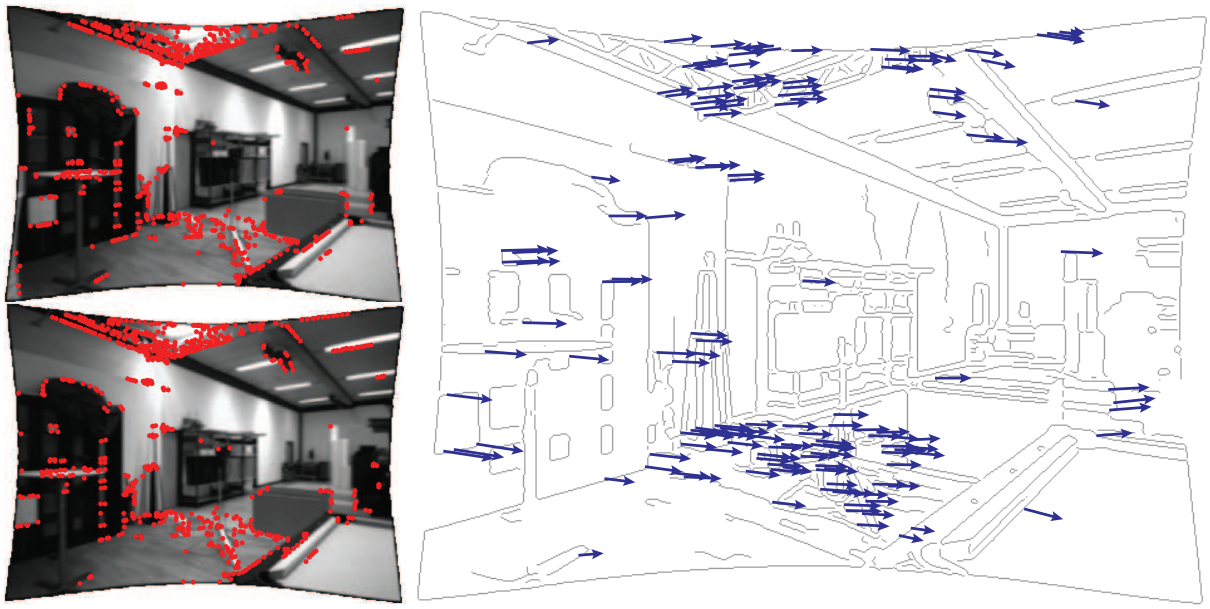


Fig. 4.17: Computation of the optical flow.

Some challenges arise due to sunlight for all applications in outdoor robotics. Two major challenges are depicted in Figures 4.16 (a) and (b). The first figure shows blooming, which occurs when the sun is shining directly into the camera. As a CCD-sensor is used, the only way to prevent blooming is to prevent the sun from shining into the camera. Hence, an artificial eyelid was developed, which can be lowered to cover the sun. Figure 4.16 (b) shows the problem, which occurs at hard transitions between a shadow and an illuminated area despite using the HSV color space. The algorithm is not able to detect the illuminated area as part of the sidewalk.
The presented algorithm is capable of running with a speed of 15 Hz at a resolution of 640x480 pixels. Both computation of the ground and the classification are computed in this time. Multicore processors are currently not supported, but by parallelizing the computation of the sidewalk, the back-projection, and the classification a speedup of approximately 200 % could be achieved. On the other hand, a frame rate of 5 Hz is adequate for the robot's speed. Hence, the remaining computational power can be used for other tasks.

4.5.3 Three-Dimensional Mapping

As three-dimensional mapping can be divided into two modules, the experimental result will be presented separately.

Ego-Motion Estimation

Figure 4.17 shows the computed optical flow for two consecutive images. The result of the actual ego-motion estimation is shown for four different scenes in Figure 4.18. Like in stereo-vision, a good trade-off between speed and quality has to be found. The sizes of the search windows N_{Ego} and N_{Search} have the largest influence on speed and quality, while the radius $r_{\min}$ of the evaluation area has a large influence on the quality. Both a small and a large $r_{\min}$ result in many faulty matches. A value of $r_{\min} = 20$ has shown the best results. The size of the search window N_{Ego} can be computed with a $\sigma = 0.2$ and $r_{\text{const}} = 10$. A larger σ will yield better results, but the computation time scales exponentially. The system achieved a median error of 43 mm and 0.014 rad per step with the selected parameters. However, Figure 4.17 shows some problems with fast rotations, where the corresponding corners are not located inside the search window. This error can be explained further by long shutter times, which are occurring often in an indoor scenario and are leading to blurry images. The algorithm computes a confidence of each measurement and discards those with a low confidence.
As the algorithm is required to be real-time capable, the computation time is of great importance. The ego-motion of two consecutive images can be computed in 34 ms. Hence, the stereo algorithm and the ego-motion estimation can be performed with a speed of 10 Hz.

Fusion of Depth Maps

As the robot's odometry achieves an accuracy of $1 - 4$ cm at a distance of 2 m, the displacement between different point clouds is small compared to their dimensions of $5 - 8$ m.

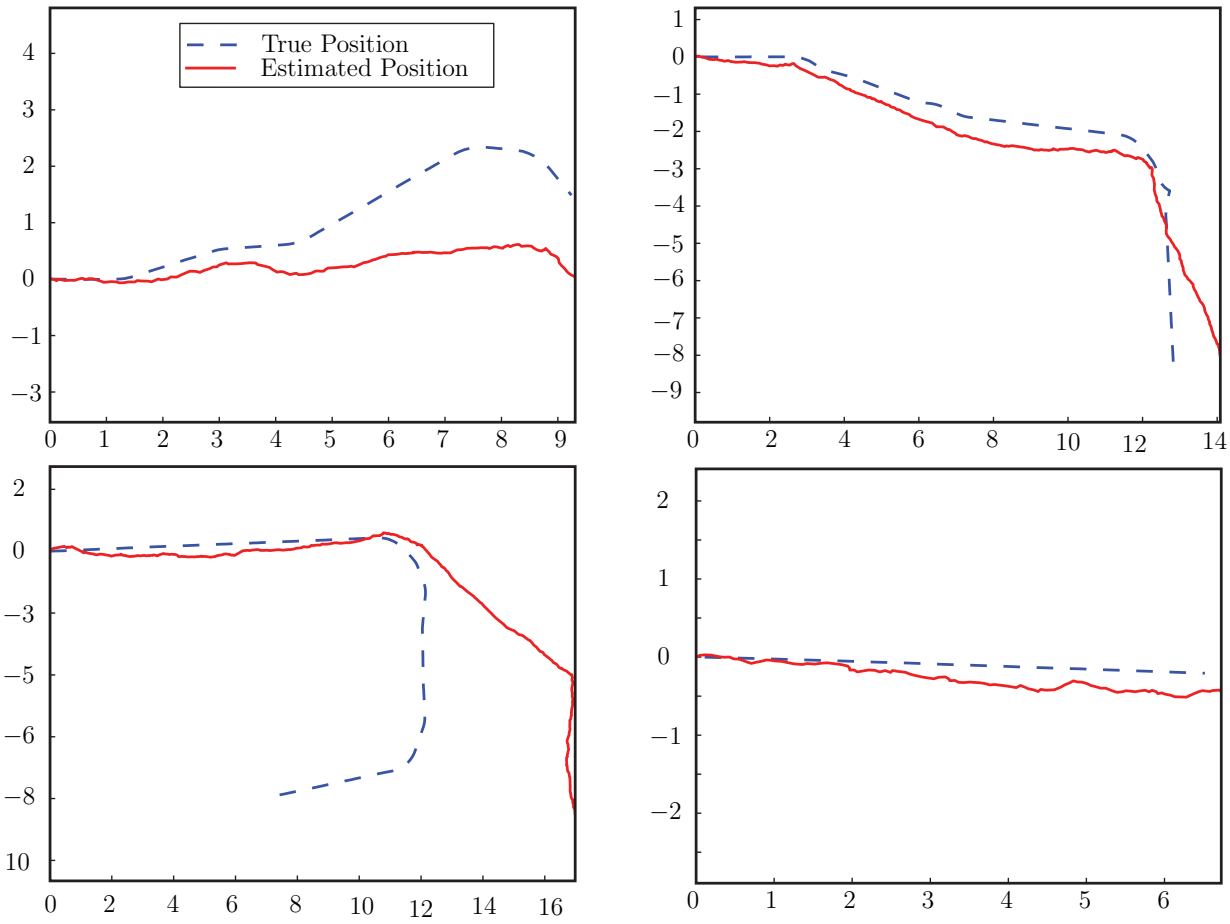


Fig. 4.18: Result of the ego-motion estimation. The solid line depicts the estimated positions and the dashed line the true position. All lengths are in m.

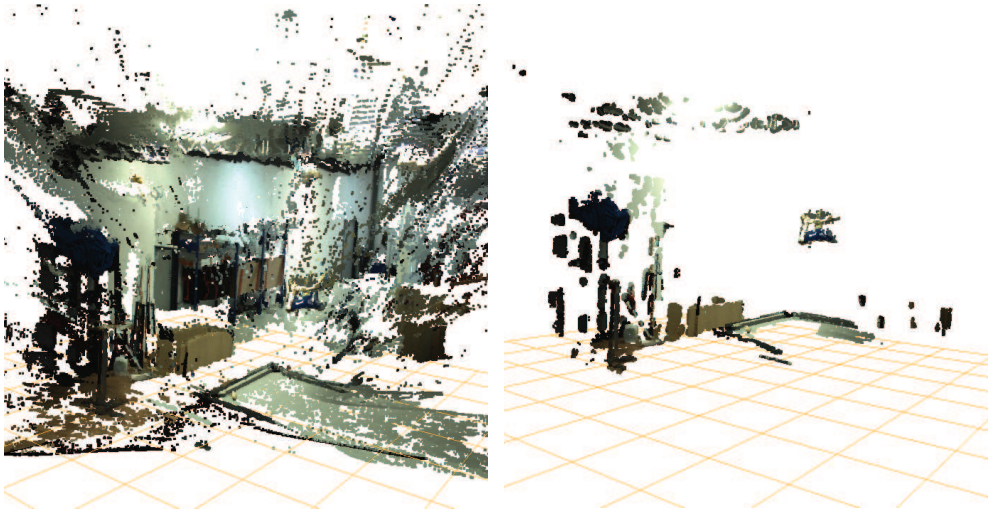


Fig. 4.19: Result of the pre-processing with the poposed parameter settings. The left figure shows the original point cloud and the right figure the result.

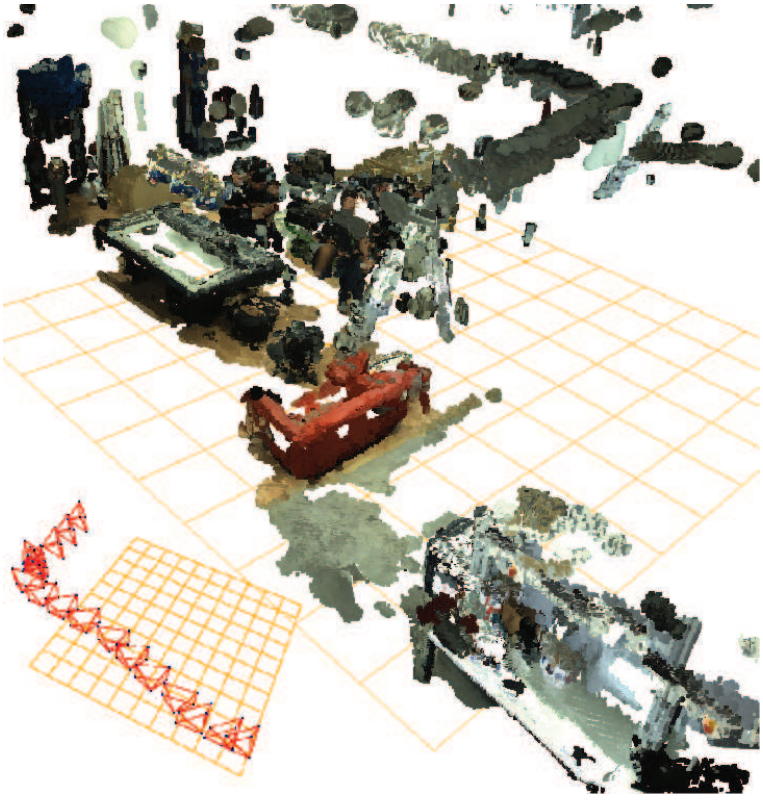


Fig. 4.20: Result of the median fusion algorithm and reconstruction of the camera position and orientation. The grid has a size of 10×10m..

As shown in Figure 4.19, around 50% of the points have been discarded in the preprocessing step. Like most stereo algorithms, only discrete disparity values can be computed, yielding a quantization error in the ground plane. As a result of the stringent parameter setting, large parts of the point clouds are removed and only those points with a high certainty remain. This effect is relativized by the high sampling rate. A parameter setting with a search radius $r = 20$ cm and a minimal number of neighbors of $N_{min} = 100$ has shown to deliver the best trade-off between quality and number of remaining points. When using a camera with a lower resolution and thus a lower density of points, the minimal number of points has to be decreased. The time needed for the noise and outlier reduction is around 50 ms for a point cloud with full resolution and 1.2 million points.
Figure 4.20 shows every fifth camera position and orientation of the reconstructed trajectory in a 10×10m grid. Furthermore, it shows the resulting point cloud, while Figure 4.21 shows a part of the reconstructed scene compared to the ground truth as measured by a laser rangefinder. The resulting point cloud is not only accurate and mostly free of outliers, but also free of redundant points, yielding a lower memory consumption (up to factor 13) compared to the original point clouds. In its current state of development, the algorithm is not able to deal with moving objects leading to multiple fragments in the reconstructed scene. This effect can be seen in Figure 4.20 behind the pool table, where a person was walking alongside with the robot and can be seen multiple times in the reconstructed scene. Besides achieving sufficient quality, the algorithm is supposed to be real-time capable in a

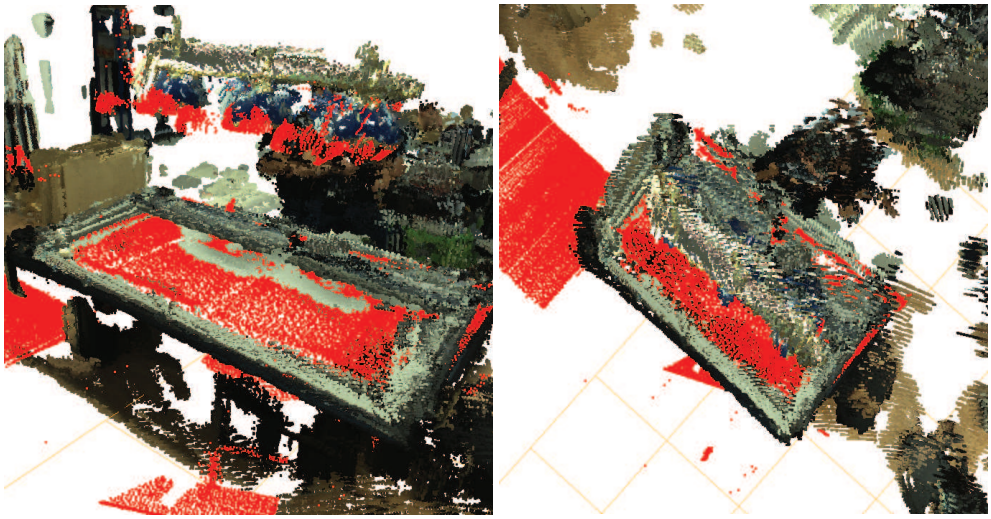


Fig. 4.21: Comparison between reconstruction from image data and ground truth obtained by a laser rangefinder (red areas).

long term vision. Hence, the computation speed can not be neglected. The time t needed for fusing two or more point cloud scales linearly with the number n of point clouds and can be estimated with $t = n \times 45$ ms. As it can be easily parallelized with the pre-processing, even higher computation speeds can be achieved by using two or more cores. These experiments have shown that the pre-processing and the median fusion algorithm is not the limiting factor for scene reconstruction in real-time. As most of the computational power is consumed by the stereo module, selecting a fast stereo algorithm is crucial for real-time capability of the presented module.

4.5.4 Sensor Fusion

The execution time for the genetic ICP algorithm is depending on the size of the point clouds. For a size of $n_L = n_C = 1000$ about 3000 generations are computed per second. Figures 4.22 (a) and 4.22 (b) show the scan of the left and the right laser rangefinder. The differences in the field of view of the two scanners can clearly be seen. The colored point cloud obtained from the vision data is shown in Figure 4.22 (c). Compared to the one obtained by laser scanners, the vision data is far less accurate, noisier, but covers a larger area of the environment. The spatial resolution is in the order of magnitude of 10 cm, compared to 0.1 cm for the laser rangefinders. The resulting merged point cloud can be seen in Figure 4.23 (b), while figure 4.23 (a) shows the certainty of the corresponding points. Green areas denote a high probability for a correct computation of a point, red areas denote a low probability.
Figure 4.24 (a) shows the median error compared to different sizes of the population. The maximal number of generations was set to 10000 generations in the first experiment and to 50000 generations in the second experiment. This experiment was conducted for two different datasets. To compute the median error, the genetic ICP algorithm was executed 20 times for each dataset, each size of population, and each maximal number of genera-

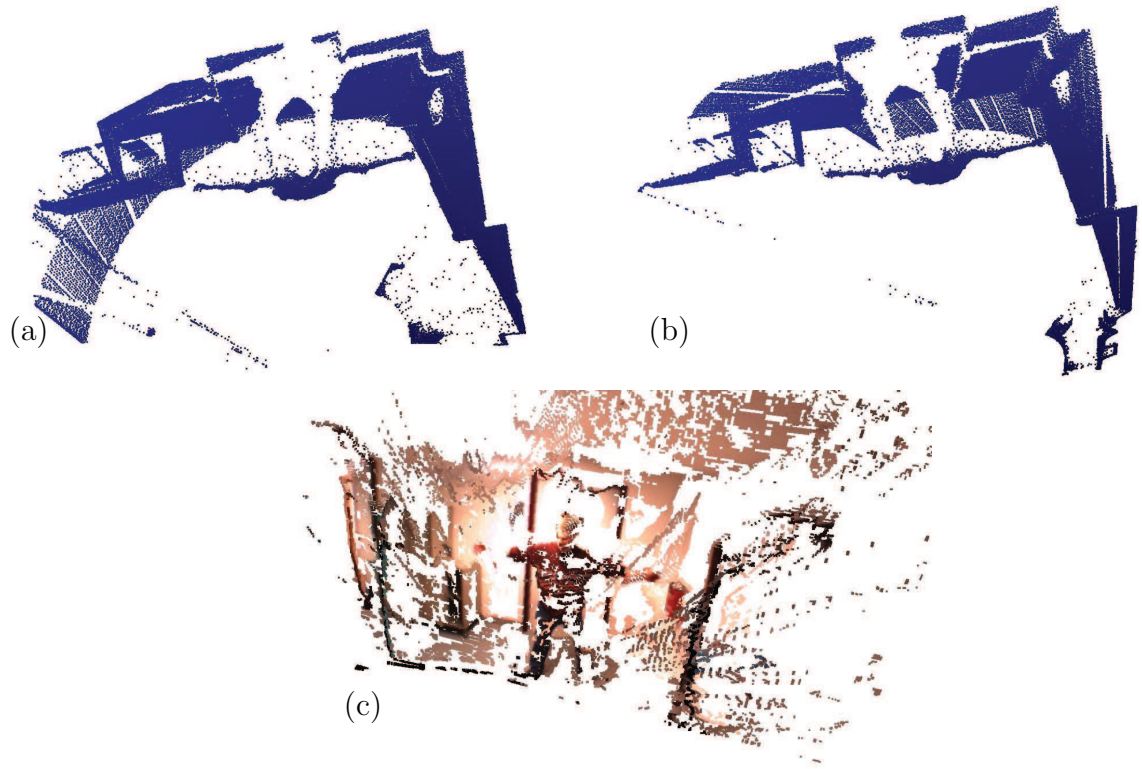


Fig. 4.22: Input for the fusion algorithm: laser scans of the (a) left and (b) right laser and (c) a point cloud obtained by the stereo module.

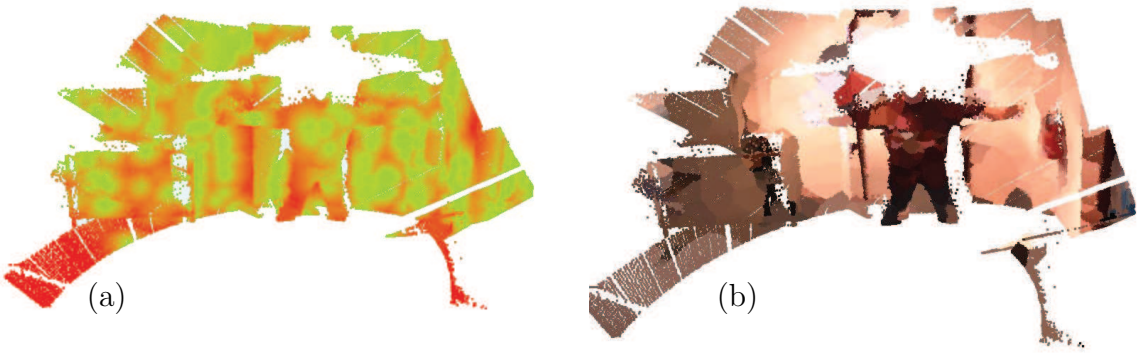


Fig. 4.23: Result of the genetic ICP algorithm: (a) probability of the match and the (b) colored point cloud.

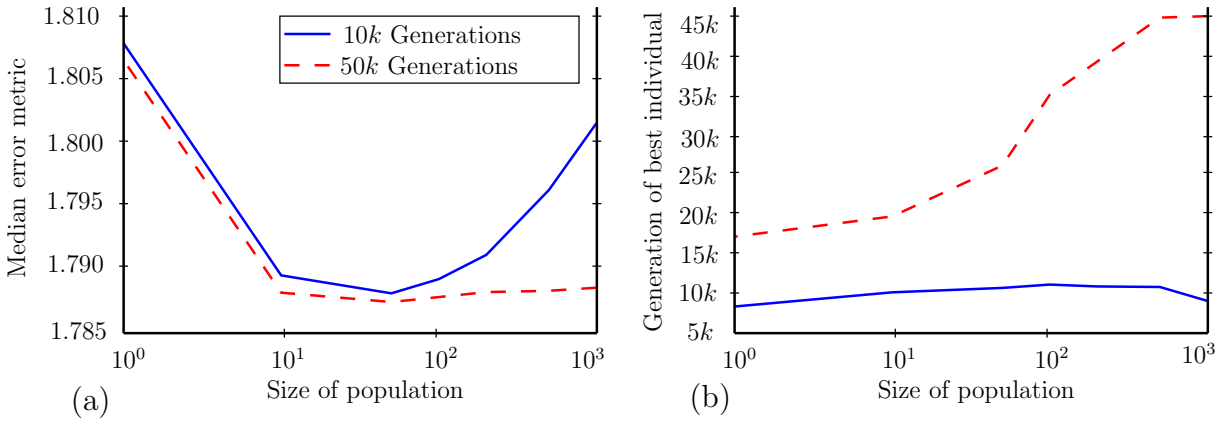


Fig. 4.24: Correlation between the size of the population and (a) the median error metric and (b) the generation of best individual.

tions. The generation that produced the best result is depicted in Figure 4.24 (a), again with different sizes of the population and for a maximal number of generations of 10000 and 50000, respectively. The median error was computed using Equation 4.18. As stated in Figure 4.24, a large number of individuals leads to a large number of generations necessary to achieve good results and a too small number of individuals leads to a relatively high medium error. When the number of generations was limited to 50000, the medium error was almost constant and the main differences have occurred in the number of generations needed to compute the best result. Consequently, the experiments have shown that the optimal number of individuals in the population is between 50 and 200. During the experiments with this population size, the best result was found in the first 20000 to 25000 generations, so the maximal number of generations should be set to this order of magnitude.

4.6 Discussion

This chapter presented algorithms for two-dimensional and three-dimensional mapping. Two-dimensional maps are required for path planning and are acquired by a texture-based approach. In contrast to existing algorithms, a memory is introduced, allowing the robot to learn those parts of the ground, where it is allowed to drive. However, this algorithm is not able to detect the ground robustly, when there are many obstacles ahead, the texture of the obstacles is identical to the ground, or the obstacles have an overhang.
Consequently, a modular system for accurate three-dimensional reconstruction of natural indoor and outdoor environments was presented. The system combines several modules, namely different stereo modules, pre- and post-processing modules, and the actual median fusion algorithm. Due to the modularity and well defined interfaces, each module can be exchanged by another module with similar functionality, a novelty compared to the state-of-the-art. For instance, a laser rangefinder could be used instead of the stereo module. Specifically, the pre-processing module including noise and outlier reduction and the median fusion algorithm have been presented in detail. The limiting bottleneck is the stereo

module, as stereo algorithms with a high resolution require vast amounts of computational power. By using a novel CUDA implementation, which combines the stereo algorithm with the ego-motion estimation, an adequate speed could be achieved. Novel confidence metrics have been introduced for both algorithms, eliminating invalid computations. On the other hand, the resolution is limited, so that future research might deal with algorithms running at higher resolutions. The ego-motion module was included to support or to replace the robot's odometry. This module presented good results, although some fast rotations cannot be recognized due to blurry images at high shutter times. In addition, a genetic ICP algorithm was presented, which is capable of aligning a colored point cloud obtained from a stereo vision module with a point cloud obtained from laser rangefinders. Before the genetic ICP algorithm can be computed, the laser rangefinders have to be calibrated to ensure a proper alignment of the point cloud. Furthermore, the current three-dimensional model is stored as a colored point cloud, which is an accurate, but very memory consuming representation. This problem can be solved by using more sophisticated methods to save the point clouds, like a textured polygon. Algorithms to convert a point cloud into a polygon grid have to be developed. Another interesting research topic is the detection and identification of moving objects.
Together with the object detection subsystem, the methods and algorithms presented in this chapter provide an ideal base for the semantic mapping system and thus a cognitive architecture. A suitable knowledge representation, which is capable of decision making is presented in the following chapter.

5 Semantic Maps

Robust object recognition and accurate three-dimensional mapping is useless without a convenient representation. Semantic networks provide an intuitive and easily accessible base to state information about objects and the attributes of the objects. Additionally, the position of the object has to be accessible from the semantic network and information about the objects have to be accessible from the metric map. So far, there exists no representation combining the advantages of semantic networks and metric maps. Although some different cognitive architectures have emerged during the last years, neither a standard architecture nor a standard representation has been established. A popular representation, which is trying to enhance metric maps, is the labeling of places. However, objects are not included in this representation. Moreover, a sound mathematic representation is essential for task planning.
A combination between semantic networks and metric maps is a well suited knowledge representation for a cognitive architecture. This leads to the introduction of so-called semantic maps providing both metric maps and semantic information, and thus combining the advantages of both representations. Compared to the state-of-the-art, the proposed semantic mapping system requires only one representation for task and path planning. Due to its modularity, it can work with arbitrary algorithms for mapping and object detection. In addition, a novel mathematic representation based on set theory is introduced, providing a state space for the objects. These states are used to derive methods and algorithms for the integration of new knowledge and for action planning. Simulations and experiments are used to validate the algorithms for the creation of semantic maps and for task planning. As the provided methods for perception and cognition and thus the corresponding processing steps are separated, semantic maps can be classified as a symbolic processing system.
This chapter starts with an introduction to semantic networks, followed by an overview of recent research advances in semantic representations. Next, Section 5.2 introduces the concept of semantic maps by presenting a sound mathematical representation. Specifically, detailed information about the implementation and integration of the object detection and mapping subsystems described in Chapter 3 and 4 will be given in Section 5.3, followed by simulation and experimental results. The chapter concludes with experimental results and a discussion, where the advantages and results of the different algorithms will be compared.

5.1 Introduction to Semantic Networks

A semantic network is a directed or undirected graph consisting of nodes and edges and is a well suited tool for knowledge representation. In particular, a node represents a concept and an edge represents relations between the concepts. A concept can be a concrete *object*, an *attribute* or an abstract *type* of an object. Typical relations are *is of type* or *has attribute*. Semantic networks have been introduced as a translation language between early computers and humans in 1956 by Richens [115] and Simmons et al. [135] in the 1960s, respectively. Nowadays, semantic networks are used for artificial intelligence, in

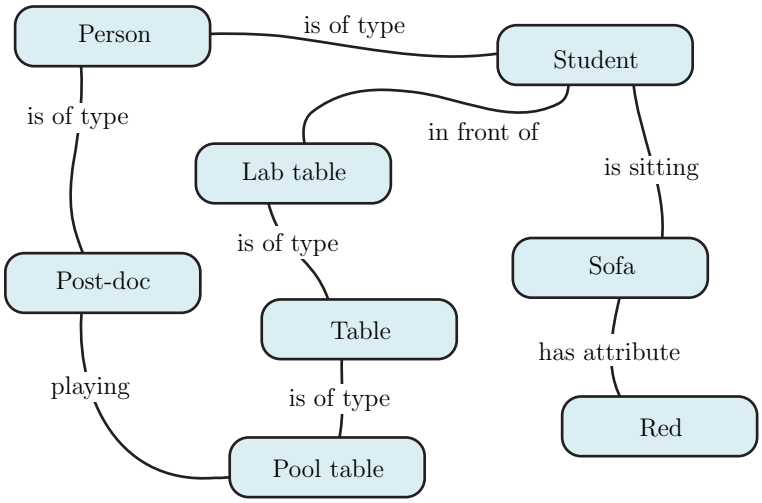


Fig. 5.1: Example for a semantic network of type existential graph. The network is describing a typical lab-scene with a pool table and a laboratory bench standing in front of a red sofa.

robotics, and in image processing systems for the interpretation of scenes and recognition and classification of places. As they have the ability to relate objects with a semantic scheme that provides further information, they are well suited for applications that rely on recognition and interpretation of natural scenes. The amount of program code can be reduced significantly by using semantic networks, as individual cases do not have to be handled separately.

5.1.1 Basic Types of Semantic Networks

All existing different types of semantic networks are based on a network consisting of nodes and edges. The types differ in type and number of different concepts, relations and in the formal definition. *Static* networks are created by a supervisor, while *dynamic* networks have the ability to grow and re-arrange and thus to learn.

Static Semantic Networks

Definitional networks are mainly based on the use of the relation *is of type*, but also less frequently on *is kind of* and *is part of*. Definitional networks use the relation *is of type* to be separated in different hierarchies.
Assertional Networks are mainly used for logical deduction. They can be separated into different subtypes with different quantifiers, e.g. existential graphs introduce a negation. Their edges represent the concrete relation between two nodes, e.g. *a camera is following an object*. Edges can be supplemented with arguments, e.g. the speed of the following movement. As they only provide real objects, they have a limited expressivity. By replacing

the relations with implications, other probabilistic networks like bayesian networks can be treated as special cases of assertional networks. Figure 5.1 shows an example of a static semantic network describing a typical lab scene.

Dynamic Semantic Networks

Executable networks have the ability to change the underlaying network either by exchanging and parsing messages, by executing procedures, or by separating and re-arranging graphs. *Learning networks* are able to learn new information. The simplest way is to transform the information into a semantic network and include it into the network. This can be enhanced by adapting weighting factors. The most complex way of learning is the re-arrangement of the whole semantic network, where fundamental changes in the structure of the network have to be made.

Hybrid Networks

A combination of features from the networks mentioned above is called a hybrid network. A hybrid network can exist of two or more separated networks working in close collaboration or can be a combination of two or more types into a single network.

5.1.2 Overview of current Semantic Knowledge Representations

In order to understand its environment, a robot needs a cognitive architecture. An overview of those cognitive systems can be found in [151]. Semantic representations provide an ideal knowledge base for these cognitive systems. In recent research, semantic information is used for task planning [39], where the robot uses predefined semantic information and is not able to learn [40]. Other approaches use a petri net to model and plan the tasks [27] or to model places by using objects [112]. Some algorithms are dealing with robot exploration based on semantic information of places. Stachniss et al. proposed algorithms for indoor robotics, which can distinguish between corridors and rooms [138], while other approaches [70] use topological knowledge about its environment for exploration. Nüchter et al. [101] introduced the robot Kurt3D [140], which is capable of building three-dimensional semantic maps. Rusu et al. [117] present a system that labels objects with semantic information in a kitchen.
Meger et al. [83] present a so-called semantic robot, which is able to drive through its environment, perform robust object detection, and label interesting places. Another system proposed by Mozos et al. [81] introduces a representation of knowledge based on different levels of abstraction, starting with a metric map up to a conceptual map. An overview of hybrid maps containing a metric map and the position of objects, can be found in [20]. As proposed by Galindo et al. [39, 40], a semantic network can be used for task planning and labeling of the map can be used for path planning.

5.2 Semantic Maps Approach

A semantic map is composed of an occupancy grid and a semantic network. The cells of the grid represent a metric map of the environment. Therefore, a probability is assigned to each cell. A high probability denotes that the cell is occupied and a low probability that the cell is free of obstacles. A novelty of the semantic maps approach are the links between cells of the occupancy grid and objects of the semantic network. As objects can be different types and can have a manifold of different attributes, this network is composed of nodes of the following four basic types:

- **object**-nodes, describing the objects itself,
- **type**-nodes, describing abstract types of objects,
- **attribute**-nodes, describing attributes of the objects or types, and
- **action**-nodes, specifying actions and relations between types of objects.

The following section introduces the novel mathematical description of a semantic map.

5.2.1 Mathematical Representation of Semantic Maps using Set Theory

A semantic network is an undirected graph and it can be described using set theory, providing a sound mathematical base. Therefore, the set $\boldsymbol{\mathcal{R}}$ is defined, containing all nodes of the semantic network, while the subset $\boldsymbol{\mathcal{O}}$ contains all nodes of type **object**, $\boldsymbol{\mathcal{A}}$ of type **attribute**, $\boldsymbol{\mathcal{T}}$ of type **type** and $\boldsymbol{\mathcal{C}}$ of type **action**:

$$\boldsymbol{\mathcal{R}} = \boldsymbol{\mathcal{O}} \cup \boldsymbol{\mathcal{A}} \cup \boldsymbol{\mathcal{T}} \cup \boldsymbol{\mathcal{C}}. \tag{5.1}$$

Every node $\mathcal{R}_i$ can be identified with a unique id i and it must be included in exactly one subset. Hence, the intersections of the subsets are empty:

$$\begin{aligned} \boldsymbol{\mathcal{O}} \cap \boldsymbol{\mathcal{A}} = \emptyset, \;\; \boldsymbol{\mathcal{O}} \cap \boldsymbol{\mathcal{T}} = \emptyset, \;\; \boldsymbol{\mathcal{O}} \cap \boldsymbol{\mathcal{C}} = \emptyset, \\ \boldsymbol{\mathcal{A}} \cap \boldsymbol{\mathcal{T}} = \emptyset, \;\; \boldsymbol{\mathcal{A}} \cap \boldsymbol{\mathcal{C}} = \emptyset \;\; \boldsymbol{\mathcal{T}} \cap \boldsymbol{\mathcal{C}} = \emptyset. \end{aligned} \tag{5.2}$$

$\dim(\boldsymbol{\mathcal{S}})$ denotes the number of nodes in the set $\boldsymbol{\mathcal{S}}$. Connections are written as a pair of nodes. For example, a connection between the two nodes $\mathcal{R}_i$ and $\mathcal{R}_j$ is denoted as $(\mathcal{R}_i, \mathcal{R}_j)$. However, not all types of nodes can be connected with each other. For the sake of clarity, different connection types are defined, depending on the types of the two nodes. Actions can be only defined, so they affect type-nodes and must not be connected to object-nodes directly. Table 5.1 shows the valid connections and introduces the connection types between the nodes.
The function $\Omega(\mathcal{R}_i)$ searches through all connection pairs and returns a set $\boldsymbol{\mathcal{S}}$ containing all nodes connected to $\mathcal{R}_i$. In order to obtain the nodes of a certain type connected to

Tab. 5.1: Valid connections between nodes

Node	Object	Attribute	Type	Action
Object	-	has_attribute	is_of_type	-
Attribute	has_attribute	-	has_attribute	affects
Type	is_of_type	has_attribute	-	has_action
Action	-	affects	has_action	-

a node, the function $\Omega^{\boldsymbol{\mathcal{A}}}(\mathcal{R}_i)$ can be used. It returns a set containing all nodes that are connected to $\mathcal{R}_i$ and are contained in the set $\boldsymbol{\mathcal{A}}$:

$$\Omega^{\boldsymbol{\mathcal{A}}}(\mathcal{R}_i) = \Omega(\mathcal{R}_i) \cap \boldsymbol{\mathcal{A}}. \tag{5.3}$$

The function Ω is also defined for a subset $\boldsymbol{\mathcal{S}}$ containing nodes as input:

$$\begin{aligned} \Omega^{\boldsymbol{\mathcal{A}}}(\boldsymbol{\mathcal{S}}) = \Omega^{\boldsymbol{\mathcal{A}}}(\mathcal{S}_1) \cup \Omega^{\boldsymbol{\mathcal{A}}}(\mathcal{S}_2) \ \ldots \ \cup \Omega^{\boldsymbol{\mathcal{A}}}(\mathcal{S}_n), \\ \text{with} \quad \boldsymbol{\mathcal{S}} = \{\mathcal{S}_1, \mathcal{S}_2, \ldots \mathcal{S}_n\}. \end{aligned} \tag{5.4}$$

As both objects-node and type-nodes can have attributes, an object has two types of attributes, direct ones and inherited ones. Hence, the get-connection function $\Psi^{\boldsymbol{\mathcal{A}}}(\mathcal{O}_i)$ returns all nodes contained in the set $\boldsymbol{\mathcal{A}}$, which are connected to the node directly or inherited via a type-node. Ψ is not only defined for object-nodes, but also for the cases defined in Equation 5.5a to 5.5p:

$$\Psi^{\boldsymbol{\mathcal{O}}}(\mathcal{O}_i) = \emptyset \tag{5.5a}$$
$$\Psi^{\boldsymbol{\mathcal{A}}}(\mathcal{O}_i) = \Omega^{\boldsymbol{\mathcal{A}}}(\mathcal{O}_i) \cup \Omega^{\boldsymbol{\mathcal{A}}}(\Omega^{\boldsymbol{\mathcal{T}}}(\mathcal{O}_i)) \tag{5.5b}$$
$$\Psi^{\boldsymbol{\mathcal{T}}}(\mathcal{O}_i) = \Omega^{\boldsymbol{\mathcal{T}}}(\mathcal{O}_i) \tag{5.5c}$$
$$\Psi^{\boldsymbol{\mathcal{C}}}(\mathcal{O}_i) = \Omega^{\boldsymbol{\mathcal{C}}}(\Omega^{\boldsymbol{\mathcal{T}}}(\mathcal{O}_i)) \tag{5.5d}$$
$$\Psi^{\boldsymbol{\mathcal{O}}}(\mathcal{A}_i) = \Omega^{\boldsymbol{\mathcal{O}}}(\mathcal{A}_i) \cup \Omega^{\boldsymbol{\mathcal{O}}}(\Omega^{\boldsymbol{\mathcal{T}}}(\mathcal{A}_i)) \tag{5.5e}$$
$$\Psi^{\boldsymbol{\mathcal{A}}}(\mathcal{A}_i) = \emptyset \tag{5.5f}$$
$$\Psi^{\boldsymbol{\mathcal{T}}}(\mathcal{A}_i) = \Omega^{\boldsymbol{\mathcal{T}}}(\mathcal{A}_i) \tag{5.5g}$$
$$\Psi^{\boldsymbol{\mathcal{C}}}(\mathcal{A}_i) = \Omega^{\boldsymbol{\mathcal{C}}}(\mathcal{A}_i) \tag{5.5h}$$
$$\Psi^{\boldsymbol{\mathcal{O}}}(\mathcal{T}_i) = \Omega^{\boldsymbol{\mathcal{O}}}(\mathcal{T}_i) \tag{5.5i}$$
$$\Psi^{\boldsymbol{\mathcal{A}}}(\mathcal{T}_i) = \Omega^{\boldsymbol{\mathcal{A}}}(\mathcal{T}_i) \tag{5.5j}$$
$$\Psi^{\boldsymbol{\mathcal{T}}}(\mathcal{T}_i) = \emptyset \tag{5.5k}$$
$$\Psi^{\boldsymbol{\mathcal{C}}}(\mathcal{T}_i) = \Omega^{\boldsymbol{\mathcal{C}}}(\mathcal{T}_i) \tag{5.5l}$$
$$\Psi^{\boldsymbol{\mathcal{O}}}(\mathcal{C}_i) = \Omega^{\boldsymbol{\mathcal{O}}}(\Omega^{\boldsymbol{\mathcal{T}}}(\mathcal{C}_i)) \tag{5.5m}$$
$$\Psi^{\boldsymbol{\mathcal{A}}}(\mathcal{C}_i) = \Omega^{\boldsymbol{\mathcal{A}}}(\mathcal{C}_i) \tag{5.5n}$$
$$\Psi^{\boldsymbol{\mathcal{T}}}(\mathcal{C}_i) = \Omega^{\boldsymbol{\mathcal{T}}}(\mathcal{C}_i) \tag{5.5o}$$
$$\Psi^{\boldsymbol{\mathcal{C}}}(\mathcal{C}_i) = \emptyset. \tag{5.5p}$$

When Ψ is computed for a non-defined case, it will return an empty set. An example of a simple semantic network composed of two object nodes of different type, some attributes, and an action is defined as follows:

$$\begin{aligned} \boldsymbol{\mathcal{O}} &= \{\mathcal{O}_1, \mathcal{O}_2\}\,, \quad \boldsymbol{\mathcal{T}} = \{\mathcal{T}_3, \mathcal{T}_4\}\,, \\ \boldsymbol{\mathcal{A}} &= \{\mathcal{A}_5, \mathcal{A}_6, \mathcal{A}_7\}\,, \quad \boldsymbol{\mathcal{C}} = \{\mathcal{C}_8\}\,, \\ \boldsymbol{\mathcal{R}} &= \{\mathcal{O}_1, \mathcal{O}_2, \mathcal{T}_3, \mathcal{T}_4, \mathcal{A}_5, \mathcal{A}_6, \mathcal{A}_7, \mathcal{C}_8\}\,, \end{aligned} \tag{5.6}$$

with the corresponding connections:

$$\begin{aligned} &(\mathcal{O}_1, \mathcal{T}_3),\ \ (\mathcal{O}_1, \mathcal{A}_7),\ (\mathcal{O}_2, \mathcal{T}_4),\ \ (\mathcal{T}_3, \mathcal{A}_5),\ \ (\mathcal{T}_4, \mathcal{A}_5), \\ &(\mathcal{T}_4, \mathcal{A}_6),\ \ (\mathcal{T}_3, \mathcal{C}_8),\ \ (\mathcal{T}_4, \mathcal{C}_8),\ \ (\mathcal{C}_8, \mathcal{A}_5). \end{aligned} \tag{5.7}$$

Figure 5.2 illustrates the semantic network and its subsets. The action $\mathcal{C}_8$ affects the

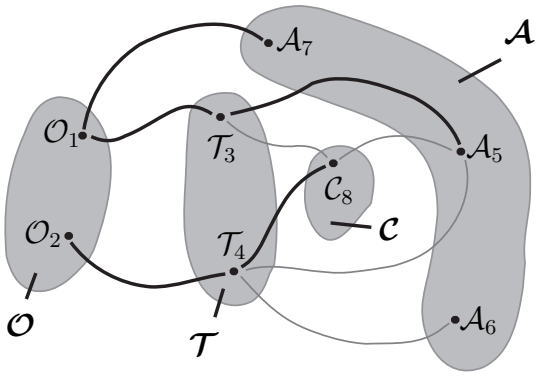


Fig. 5.2: Illustration of a semantic network and its subsets. The connections as described in Equation 5.8 are highlighted.

attribute $\mathcal{A}_5$ of both objects. The results of the get-connection function for the example given in Equation 5.6 can be stated as follows:

$$\begin{aligned} \Psi^{\boldsymbol{\mathcal{A}}}(\mathcal{O}_1) &= \{\mathcal{A}_5, \mathcal{A}_7\}\,, \\ \Psi^{\boldsymbol{\mathcal{C}}}(\mathcal{O}_2) &= \{\mathcal{C}_8\}\,. \end{aligned} \tag{5.8}$$

As defined in Equation 5.5b, $\Psi^{\boldsymbol{\mathcal{A}}}(\mathcal{O}_1)$ returns both the directly connected attribute $\mathcal{A}_7$ and the attribute $\mathcal{A}_5$, which was inherited via the type-node $\mathcal{T}_3$. Figure 5.2 highlights the corresponding connections.

5.2.2 Introducing States for Objects

To plan meaningful actions, states for the objects are introduced. Every attribute connected to an object, either directly or via a type-node, represents one variable of the object's state. Therefore $\chi_{\mathcal{O}_j}^{\mathcal{A}_i}$ denotes the state variable of the object-node $\mathcal{O}_j$ with respect to the attribute-node $\mathcal{A}_i$. Hence, the number of attribute-nodes connected (again directly

or indirectly) to an object represents the number of dimensions of its state. A state variable can be both discrete or continuous. Its type (e.g. its unit) must be defined unambiguously within the corresponding attribute-node. The state space X_j of a node $\mathcal{O}_j$ is defined as:

$$X_j = \left[\chi_{\mathcal{O}_j}^{\mathcal{A}_q}, \chi_{\mathcal{O}_j}^{\mathcal{A}_r}, \dots \chi_{\mathcal{O}_j}^{\mathcal{A}_s}\right]^T \text{ , with } \Psi^{\boldsymbol{\mathcal{A}}}(\mathcal{O}_j) = \{\mathcal{A}_q, \mathcal{A}_r, \dots \mathcal{A}_s\}\,, \tag{5.9}$$

These states can be used for the computation of the similarity of two objects (see Section 5.2.3). The states of the two objects from the given example can be written as:

$$X_1 = \left[\chi_{\mathcal{O}_1}^{\mathcal{A}_5}, \chi_{\mathcal{O}_1}^{\mathcal{A}_7}\right]^T, \quad X_2 = \left[\chi_{\mathcal{O}_2}^{\mathcal{A}_5}, \chi_{\mathcal{O}_2}^{\mathcal{A}_6}\right]^T. \tag{5.10}$$

The attribute-node $\mathcal{A}_5$ is connected to both objects and hence occurs in both states. Needless to say, $\chi_{\mathcal{O}_1}^{\mathcal{A}_5}$ and $\chi_{\mathcal{O}_2}^{\mathcal{A}_5}$ are independent and not affecting each other. As action-nodes can only be connected to type-nodes and the effect of an action to a state variable has to be specified, abstract state variables $\chi_{\mathcal{T}_j}^{\mathcal{A}_i}$ are introduced. Unlike a normal state variable, they are not connected to an object-node but to a type-node. Consequently, object-nodes have state variables while type-nodes have abstract state variables. An action affects the states of the object-nodes, it is connected to via a type-node. Furthermore, the action has to be connected directly with the corresponding attribute-node. This yields in a set of abstract state variables that serve as input for the action. When performing the action, these abstract state variables are updated. $\widehat{\chi}_{\mathcal{T}_j}^{\mathcal{A}_i}$ denotes an abstract state variable after it has been updated. Therefore, each action $\mathcal{C}_i$ provides a function Λ_i, which updates the abstract state variables:

$$\Lambda_i\left(\chi_{\mathcal{T}_m}^{\mathcal{A}_a}, \dots \chi_{\mathcal{T}_m}^{\mathcal{A}_b}, \dots \chi_{\mathcal{T}_n}^{\mathcal{A}_c}, \dots \chi_{\mathcal{T}_n}^{\mathcal{A}_d}\right) = \left(\widehat{\chi}_{\mathcal{T}_m}^{\mathcal{A}_a}, \dots \widehat{\chi}_{\mathcal{T}_m}^{\mathcal{A}_b}, \dots \widehat{\chi}_{\mathcal{T}_n}^{\mathcal{A}_c}, \dots \widehat{\chi}_{\mathcal{T}_n}^{\mathcal{A}_d}\right), \tag{5.11}$$

with

$$\begin{aligned}
\{\mathcal{T}_m, \dots \mathcal{T}_n\} &\in \Psi^{\boldsymbol{\mathcal{T}}}(\mathcal{C}_i),\\
\{\mathcal{A}_a, \dots \mathcal{A}_d\} &\in \Psi^{\boldsymbol{\mathcal{A}}}(\mathcal{C}_i),\\
\{\mathcal{A}_a, \dots \mathcal{A}_b\} &\in \Psi^{\boldsymbol{\mathcal{A}}}(\mathcal{T}_m),\\
\{\mathcal{A}_c, \dots \mathcal{A}_d\} &\in \Psi^{\boldsymbol{\mathcal{A}}}(\mathcal{T}_n).
\end{aligned}$$

Hence, an action can modify the abstract states of several arbitrary types-nodes and not all state variables of one type-node. The function Λ_8 for the action $\mathcal{C}_8$ as defined in the previous example in Equations 5.6 and 5.7 can be written as:

$$\Lambda_8\left(\chi_{\mathcal{T}_3}^{\mathcal{A}_5}, \chi_{\mathcal{T}_4}^{\mathcal{A}_5}\right) = \left(\widehat{\chi}_{\mathcal{T}_3}^{\mathcal{A}_5}, \widehat{\chi}_{\mathcal{T}_4}^{\mathcal{A}_5}\right). \tag{5.12}$$

As the action-node $\mathcal{C}_8$ is only connected to the attribute-node $\mathcal{A}_5$ and it is connected to two type-nodes, it only affects one state variable of the type-nodes it is connected to.

5.2.3 Similarity Probability of Objects

When the robot is moving in its environment, the SLAM algorithm will most likely generate errors, e.g. when performing a relaxation step. This results in the possible detection of the same object at different positions. Moving objects lead to the same effect. Consequently, a method to compute the probability of the similarity of two objects is introduced. In a first step, a similarity measurement $\mathrm{s}(\mathcal{O}_i, \mathcal{O}_j)$ between the two object nodes $\mathcal{O}_i$ and $\mathcal{O}_j$ with the corresponding states $X_i = [\chi_{\mathcal{O}_i}^{\mathcal{A}_a}, \dots \chi_{\mathcal{O}_i}^{\mathcal{A}_b}]^T$ and $X_j = [\chi_{\mathcal{O}_j}^{\mathcal{A}_c}, \dots \chi_{\mathcal{O}_j}^{\mathcal{A}_d}]^T$ is computed. This computation is only valid, when the two nodes are of the same type:

$$\mathrm{s}(\mathcal{O}_i, \mathcal{O}_j) = \begin{cases} d_0 + d_1, & \text{if } \Psi^{\mathcal{T}}(\mathcal{O}_i) = \Psi^{\mathcal{T}}(\mathcal{O}_j) \\ 0, & \text{if } \Psi^{\mathcal{T}}(\mathcal{O}_i) \neq \Psi^{\mathcal{T}}(\mathcal{O}_j) \end{cases} . \tag{5.13}$$

In detail, $\mathrm{s}(\mathcal{O}_i, \mathcal{O}_j)$ contains a normalized squared euclidean distance d_0 and a distance function d_1, which is used to estimate the similarity between the attributes of the objects. The euclidean distance function considers only those objects, with a distance of less than d_{Max}:

$$d_0 = \begin{cases} \frac{1}{d_{\mathrm{Max}}^2} d(\mathcal{O}_i, \mathcal{O}_j)^2, & \text{if } d(\mathcal{O}_i, \mathcal{O}_j)^2 < d_{\mathrm{Max}}^2 \\ 1, & \text{otherwise} \end{cases} . \tag{5.14}$$

To compute the similarity between the attributes of the objects, a distance function between two attributes of the same type is computed and weighted for each attribute of the state:

$$d_1 = \sum_{k=1}^{k=n} \alpha_k \, d(\chi_{\mathcal{O}_i}^{\mathcal{A}_k}, \chi_{\mathcal{O}_j}^{\mathcal{A}_k}), \tag{5.15}$$

with $n = \max(\dim(X_i), \dim(X_j))$. The weighting factor α_k has to be evaluated thoroughly for each state variable with respect to the importance of the attribute, so that more important attributes are assigned with a larger α_k. Furthermore, the weighting factors have to be normalized:

$$\sum_{k=1}^{k=n} \alpha_k = 1. \tag{5.16}$$

The distance function $d(\chi_{\mathcal{O}_i}^{\mathcal{A}_k}, \chi_{\mathcal{O}_j}^{\mathcal{A}_k})$ has to be normalized and can only be computed, if both state variables $\chi_{\mathcal{O}_i}^{\mathcal{A}_k}$ and $\chi_{\mathcal{O}_j}^{\mathcal{A}_k}$ are existing. If one of them is not existing the distance function will return $d(\chi_{\mathcal{O}_i}^{\mathcal{A}_k}, \chi_{\mathcal{O}_j}^{\mathcal{A}_k}) = 1$. Additionally, the distance function has to be defined independently for each type of attribute so that the result of the function varies between 0 for a large similarity and 1 for a large divergence. In most of the cases, the attribute can be described as a scalar and the distance function can be estimated by normalizing both states and computing an euclidean distance. On the contrary, there are some exceptions where a more complex computation is required, for example if an attribute describes color information:

$$d(\chi_{\mathcal{O}_i}^{\mathcal{A}_k}, \chi_{\mathcal{O}_j}^{\mathcal{A}_k}) = (\frac{1}{3}((r_i - r_j)^2 + (g_i - g_j)^2 + (b_i - b_j)^2))^{\frac{1}{2}} , \tag{5.17}$$

where r denotes the red color channel of the attribute, g the green channel and b the blue channel. All color channels have to be normalized. If the attribute is of type boolean, the distance function can be computed as:

$$d(\chi_{\mathcal{O}_i}^{\mathcal{A}_k}, \chi_{\mathcal{O}_j}^{\mathcal{A}_k}) = \begin{cases} 0, & \text{if } \chi_{\mathcal{O}_i}^{\mathcal{A}_k} = \chi_{\mathcal{O}_j}^{\mathcal{A}_k} \\ 1, & \text{otherwise} \end{cases} . \tag{5.18}$$

After all distance functions have been computed, the similarity estimation has to be converted into a probability:

$$p(\mathcal{O}_i, \mathcal{O}_j) = 1 - \frac{1}{2}\mathrm{s}(\mathcal{O}_i, \mathcal{O}_j), \tag{5.19}$$

where a large $p(\mathcal{O}_i, \mathcal{O}_j)$ indicates a high probability of similarity.

5.2.4 Action Planning using Object States

At first a reduced abstract type state Y_j is introduced, containing only those states of the type $\mathcal{T}_j$, which are of interest for the user:

$$Y_j = \left[\chi_{\mathcal{T}_j}^{\mathcal{A}_r}, \dots \chi_{\mathcal{T}_j}^{\mathcal{A}_s}\right]^T . \tag{5.20}$$

Y_j must contain at least one element. A set containing arbitrary reduced abstract states is called $\mathbf{Y}$. Now, the inverse action function Λ_i^{-1} can be used to search for suitable actions. Λ_i^{-1} uses the desired state variables $\widehat{\chi}_{\mathcal{O}_i}^{\mathcal{A}_j}$ as input and delivers those states $\chi_{\mathcal{O}_m}^{\mathcal{A}_n}$, which are required to perform the action. Additionally, Λ_i^{-1} returns an estimation of the necessary costs σ of the corresponding action:

$$\Lambda_i^{-1}\left(\widehat{\chi}_{\mathcal{T}_m}^{\mathcal{A}_a}, \dots \widehat{\chi}_{\mathcal{T}_m}^{\mathcal{A}_b}, \dots \widehat{\chi}_{\mathcal{T}_n}^{\mathcal{A}_c}, \dots \widehat{\chi}_{\mathcal{T}_n}^{\mathcal{A}_d}\right) = \left(\sigma, \chi_{\mathcal{T}_m}^{\mathcal{A}_a}, \dots \chi_{\mathcal{T}_m}^{\mathcal{A}_b}, \dots \chi_{\mathcal{T}_n}^{\mathcal{A}_c}, \dots \chi_{\mathcal{T}_n}^{\mathcal{A}_d}\right). \tag{5.21}$$

If a certain state cannot be achieved, Λ_i^{-1} returns a cost of $\sigma = -1$. Therefore the search function Φ is introduced, which uses the inverse action function to find the best suited action. Input of the function is a set of desired state variables and output is the estimated cost to perform the action, as well as a set $\mathbf{C}$, containing the found actions:

$$\Phi(\widehat{\chi}_{\mathcal{O}_i}^{\mathcal{A}_n}, \dots \widehat{\chi}_{\mathcal{O}_j}^{\mathcal{A}_m}) = (\sigma, \mathcal{C}_i^{\mathcal{O}_m}, \dots \mathcal{C}_j^{\mathcal{O}_n}), \tag{5.22}$$

while $\mathcal{C}_i^{\mathcal{O}_m}$ denotes that action $\mathcal{C}_i$ is performed with object $\mathcal{O}_m$. Actions affecting two or more objects $\mathcal{O}_m$ and $\mathcal{O}_n$ are denoted as $\mathcal{C}_i^{\mathcal{O}_{m,n}}$. Algorithm 5.1 describes the search function. Internally, Φ uses Λ_i^{-1} to find all possible actions that lead to the desired state. As Λ_i^{-1} returns a set of abstract states, a suitable object-node has to be found. Therefore Φ computes the action-cost for all object-nodes of the corresponding type. The driving costs towards the object are also taken into account. If an action requires a certain state value and no object in this state was found, Φ calls itself recursively to compute the actions required to achieve the states returned by the function Λ_i^{-1}. Φ will select the object and action with the lowest total costs. Finally the lowest instance of Φ will return a list of actions and corresponding object-nodes. A cost of $\sigma = -1$ will be returned, if no action

Algorithm 5.1 Recursive search-function $\Phi(\widehat{Y}) = (\sigma, \mathbf{C})$

$\mathbf{C} = \emptyset$, $\sigma = 0$
for all $\widehat{\chi}_{\mathcal{O}_i}^{\mathcal{A}_n} \in \widehat{Y}$ **do**
  if $\widehat{\chi}_{\mathcal{O}_i}^{\mathcal{A}_n} \neq \chi_{\mathcal{O}_i}^{\mathcal{A}_n}$ **then**
    for all Λ_l^{-1} with $\mathcal{C}_l \in \Psi^{\mathcal{C}}(\mathcal{O}_i)$ **do**
      $\mathbf{C}_l = \emptyset$
      Compute $\Lambda_l^{-1}\left(\widehat{\chi}_{\Psi^{\boldsymbol{\tau}}(\mathcal{O}_i)}^{\mathcal{A}_n}\right) = (\sigma_l, Y)$
      if $\sigma_l \neq -1$ **then**
        for all $\widehat{\chi}_{\mathcal{T}_j}^{\mathcal{A}_m} \in Y$ **do**
          for all $\mathcal{O}_l \in \Psi^{\mathcal{O}}(\mathcal{T}_j)$ **do**
            $\mathbf{C}_u = \emptyset$, $\sigma_u = 0$
            Compute $\Phi(\widehat{\chi}_{\mathcal{O}_l}^{\mathcal{A}_m}) = (\sigma_t, \mathbf{C}_t)$
            if $\sigma_t \neq -1$ **then**
              Compute driving cost σ_d to object $\mathcal{O}_l$
              $\sigma_u = \sigma_d + \sigma_t$
              $\mathbf{C}_u = \mathbf{C}_t$
            end if
            Store σ_u and $\mathbf{C}_u$ into memory
          end for
          Select $\mathbf{C}_u$ with best σ_u
          $\sigma_l = \sigma_l + \sigma_u$
          $\mathbf{C}_l = \mathbf{C}_u \cap \mathbf{C}_l$
        end for
        Store σ_l and $\mathbf{C}_l$ into memory
      end if
    end for
    Select $\mathbf{C}_l$ with best σ_l
    if $\sigma_l = -1$ **then**
      return $(-1, \emptyset)$
    end if
    Compute driving cost σ_d to object $\mathcal{O}_i$
    $\sigma = \sigma + \sigma_d + \sigma_l$
    $\mathbf{C} = \mathbf{C}_l \cap \mathbf{C}$
  end if
end for
return $(\sigma, \mathbf{C})$

can be found, and $\sigma = 0$, if all objects are already in the desired states.
To illustrate the search-function, the example defined in Equation 5.6 and 5.7 is extended with the action $\mathcal{C}_9$, which has the following connections: $(\mathcal{T}_4, \mathcal{C}_9)$ and $(\mathcal{C}_9, \mathcal{A}_5)$. $\mathcal{C}_9$ has the following action function:

$$\Lambda_9\left(\chi_{\mathcal{T}_4}^{\mathcal{A}_5}\right) = \left(\widehat{\chi}_{\mathcal{T}_4}^{\mathcal{A}_5}\right). \tag{5.23}$$

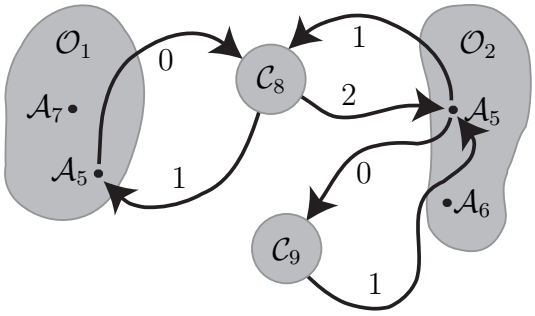


Fig. 5.3: Illustration of objects, actions and the corresponding state changes in an action-centered view.

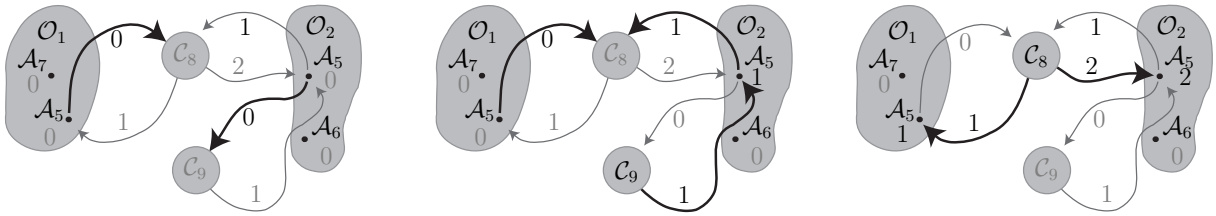


Fig. 5.4: Illustration of action planning in three steps (from left to right) in an action-centered view. Black lines denote possible actions.

Let $\mathcal{O}_1$ and $\mathcal{O}_2$ have the following initial states:

$$X_1 = \left[\chi_{\mathcal{O}_1}^{\mathcal{A}_5}, \chi_{\mathcal{O}_1}^{\mathcal{A}_7}\right]^T = [0,0]^T, \quad X_2 = \left[\chi_{\mathcal{O}_2}^{\mathcal{A}_5}, \chi_{\mathcal{O}_2}^{\mathcal{A}_6}\right]^T = [0,0]^T. \tag{5.24}$$

The action functions have the following numerical values:

$$\begin{aligned} \Lambda_8\left(\chi_{\mathcal{T}_3}^{\mathcal{A}_5}, \chi_{\mathcal{T}_4}^{\mathcal{A}_5}\right) = \Lambda_8(0,1) = \left(\widehat{\chi}_{\mathcal{T}_3}^{\mathcal{A}_5}, \widehat{\chi}_{\mathcal{T}_4}^{\mathcal{A}_5}\right) = (1,2), \\ \Lambda_9\left(\chi_{\mathcal{T}_4}^{\mathcal{A}_5}\right) = \Lambda_9(0) = \left(\widehat{\chi}_{\mathcal{T}_4}^{\mathcal{A}_5}\right) = 1. \end{aligned} \tag{5.25}$$

Figure 5.3 shows the actions of the example and the corresponding state changes in an action-centered view. The desired object state for $\mathcal{O}_1$ is:

$$\widehat{X}_1 = \left[\widehat{\chi}_{\mathcal{O}_1}^{\mathcal{A}_5}, \widehat{\chi}_{\mathcal{O}_1}^{\mathcal{A}_7}\right]^T = [1,0]^T. \tag{5.26}$$

The result of the action planning is shown in Figure 5.4. In the first step, the function $\Phi'(\widehat{\chi}_{\mathcal{O}_1}^{\mathcal{A}_5} = 1, \widehat{\chi}_{\mathcal{O}_1}^{\mathcal{A}_7} = 0)$ is executed. This function finds Λ_8 as solution, with an input of $\chi_{\mathcal{T}_3}^{\mathcal{A}_5} = 0$ and $\chi_{\mathcal{T}_4}^{\mathcal{A}_5} = 1$. In detail, the two objects $\mathcal{O}_1$ and $\mathcal{O}_2$ are selected to perform the action. However, the second requirement is not fulfilled, so another instance will be called: $\Phi''(\widehat{\chi}_{\mathcal{O}_2}^{\mathcal{A}_5} = 1)$. This instance will return the action $\mathcal{C}_9$ executed on object $\mathcal{O}_2$. Now, both requirements are fulfilled and Φ' will now return $\mathcal{C}_8^{\mathcal{O}_{1,2}}$. Consequently, the following actions have to be performed: $(\mathcal{C}_9^{\mathcal{O}_2}, \mathcal{C}_8^{\mathcal{O}_{1,2}})$.

A sound and powerful mathematical description of semantic maps was introduced in this section. This description serves as a base for further algorithms, like the computation of a similarity probability of two objects or the action planning using the newly introduced object states. In contrast to existing algorithms, only one representation is sufficient for both action planning and path planning. Equally important, the knowledge representation has to be integrated with the presented perception system. The next section describes, how the two systems can be integrated and provides further details about the implementation.

5.3 Implementation

A framework containing a simulator, the semantic processing, and different robot clients has been implemented to verify and test the algorithms. The semantic processing serves as an interface combining two- and three-dimensional mapping with object detection. Both semantic processing and the simulator act as a server. To ensure connectivity, the simulator and the semantic processing have the same interface. Communication between the server and the client is either using the UDP network protocol when server and robot are distributed, or the D-Bus protocol when server and client are running on the same machine.

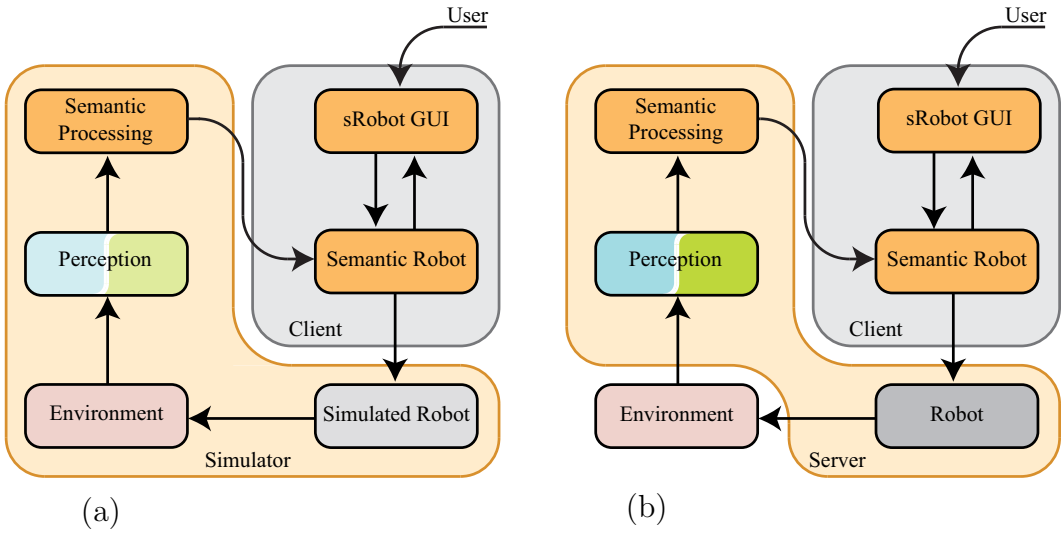


Fig. 5.5: Architecture of (a) the simulator and the semantic processing in (b) a real world environment.

Figure 5.5 shows the difference between the simulated and the real environment. The semantic robot client can either be attached to a real or to a simulated robot, both having the same interface. Both servers, the simulator and the semantic processing, are equipped with an optional GUI providing information about the current processing steps. Another GUI is provided for the semantic robot, offering different views of the semantic map, methods to modify the underlying semantic network, and functions to control the robot and initiate actions.

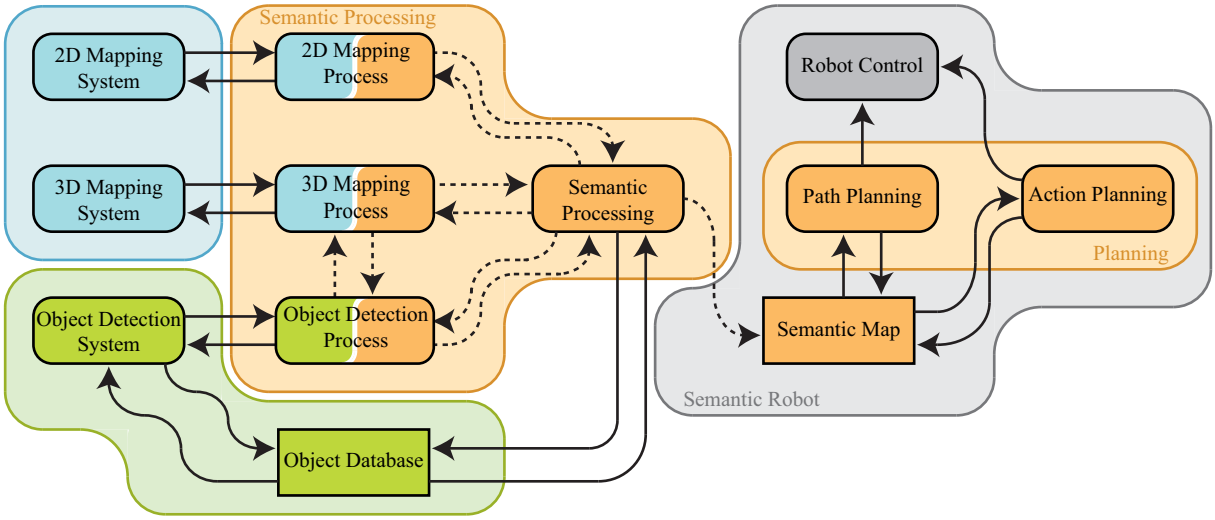


Fig. 5.6: Architecture of the semantic mapping system. Solid lines indicate conventional inter-thread communication while dashed lines indicate communication via D-Bus or a network protocol like UDP. The object database can be opened in several instances.

5.3.1 Simulator

Robots are simulated in a two-dimensional environment with different types of objects. Arbitrary attributes can be assigned to each object. Each robot is simulated as a non-holonomic system and the client can control the driving- and rotational speed. Furthermore, the server transmits range data of a simulated laser rangefinder to each connected client. If in the client's field of view, the position of objects and their attributes are also transmitted. Clients can perform simple actions on objects, which change attributes or the place of the objects. The simulated environment consists of a bitmap representing the map and a xml-file, where information about the initial position of the clients, as well as the position, type, and attributes of the objects and possible actions are defined. Not only the movement of the robot and the environment is simulated, but also the perception system of the robot. Consequently, the simulator provides an ideal base for testing the semantic mapping subsystem.

5.3.2 Integration of Semantic Maps with Mapping and Object Detection

Figure 5.6 shows the architecture of a complete semantic mapping system, including a two and a three-dimensional mapping system and an object detection system. As the three systems run independently and with different rates, they are separated in a total of four different processes: two-dimensional mapping, three-dimensional mapping for the detection of obstacles, object detection, and semantic processing. The semantic processing runs with the highest priority, followed by the two-dimensional mapping. The three-dimensional mapping and object detection process have the same, and lowest, priority. Furthermore, the semantic processing serves as a clock-generator, controlling the timing of the other pro-

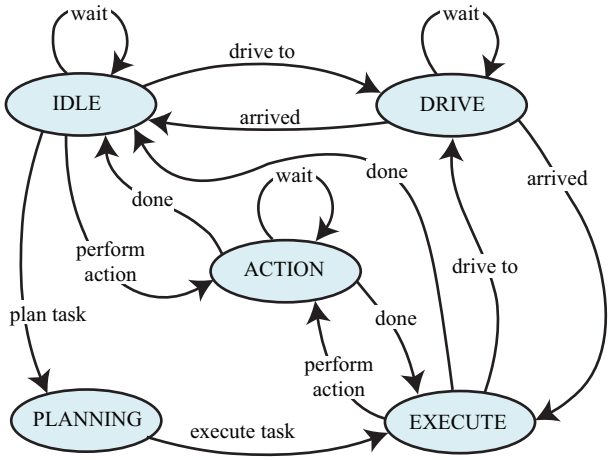


Fig. 5.7: Finite State Machine of a semantic robot with the states 'idle', 'drive', 'action', 'planning', and 'execute'.

cesses. It can start or stop the other processes, manage and distribute the data acquired from the robot, and process the resulting data received from the processes. To ensure the encapsulation, each process is running in its own exclusive memory space. Hence, memory conflicts can be avoided and unexpected behavior of one process will not affect the other ones. The communication between the processes is realized with the D-Bus protocol and shared memory blocks.
The three-dimensional object detection module returns the type of the detected object and its position in three dimensions, which can easily be included into the semantic map. However, the two-dimensional object detection will return the type of the object and its position and dimensions in the input image. Consequently, the position has to be transformed into the robot system, before it can be included into the semantic map. Using stereo triangulation is the most eligible method. To compute the disparity that is needed for stereo triangulation, the position of the object needs to be known in both images of the stereo image pair. On the other hand, a complete disparity map is already being computed in the three-dimensional mapping module. To avoid unnecessary computations, this disparity map can be used: the object detection process sends the position and dimensions of the objects to the three-dimensional mapping process, which then computes the median disparity in the region of the image, where the object was found. This median disparity is then used to compute the position of the object. For the insertion of the object into the semantic map, the z-coordinate of the objects position can be neglected.

5.3.3 Semantic Robot Client

A robot client needs functions to connect to a server, send driving commands, and to receive and interpret the data sent by the server. By using the simulated laser data, the robot can easily build a two-dimensional map. Furthermore it is able to understand the maps created from the two and three-dimensional mapping modules. The so called *semantic robot* is composed out of these basic components and is providing an

implementation of a semantic map as described in Section 5.2. For this purpose, the robot processes the object data received from the server and creates new nodes and links in the semantic network, when a new object was found or updates the network, when attributes of an object have changed. Predefined information can be used by loading a semantic network, which is containing different types of objects, possible attributes, and actions. This predefined semantic network will then be updated with the new information received by the server.
Figure 5.7 shows the finite state machine of the semantic robot client. Received information like range data or information about new objects is processed in all states. In order to assign a new action, the desired state of an object can be given to the robot. Now the finite state machine will switch to the state PLANNING, which computes the search function Φ. After planning a suitable list of actions, it will start to EXECUTE the plan. During execution, the robot will switch between the states ACTION and DRIVE until the object is in the desired state. The execution time and thus the driving time is used to estimate the costs σ for the search function.

Besides the mathematical background, details about the implementation of a semantic mapping system have been given in this section. The implementation framework includes a simulator, which is used to verify the cognition system. Additionally, a semantic robot has been presented, which uses semantic mapping and can either be used with a simulator or a real robot. In the next section, the complete semantic mapping system will by verified by using simulations and real world experiments.

5.4 Simulation Results

The simulator was used to test and verify the presented semantic mapping architecture. Several objects have been placed in the simulated environment, amongst others a cup and two objects of type bottle. The cup had the attribute *level* with the value *empty*, and the bottles the attributes *level* and *closed* with the corresponding values *full* and *true* for one bottle and *false* for the other one, respectively.
After driving and exploring the scenario, the robot had built the semantic map as shown in Figure 5.8. Furthermore, the actions *open* and *fill* have been defined in a predefined semantic network. The depicted semantic map shows not only the objects and their attributes, but also their positions. Every object has a link to its corresponding cell in the metric map and the cells have links to the nodes of the network vice versa. Figure 5.9 shows the state changes of the defined actions.
To test the planning algorithm, the robot was assigned with the task to find a cup with the state *level:full*. Therefore, the robot had two possibilities to perform the action *fill*. One was to use the bottle with state *open:true* and the other one was to open the other bottle by performing action *open* before performing action *fill*. Depending on the initial positions of the robot and the objects, the semantic robot chose the alternative with the lowest overall cost. When the distance between the robot and the bottle with state *closed:true* is smaller than the other distance, so that the driving costs to the more distant bottle are larger than the costs for the action *open*, the robot will the select the nearer bottle.

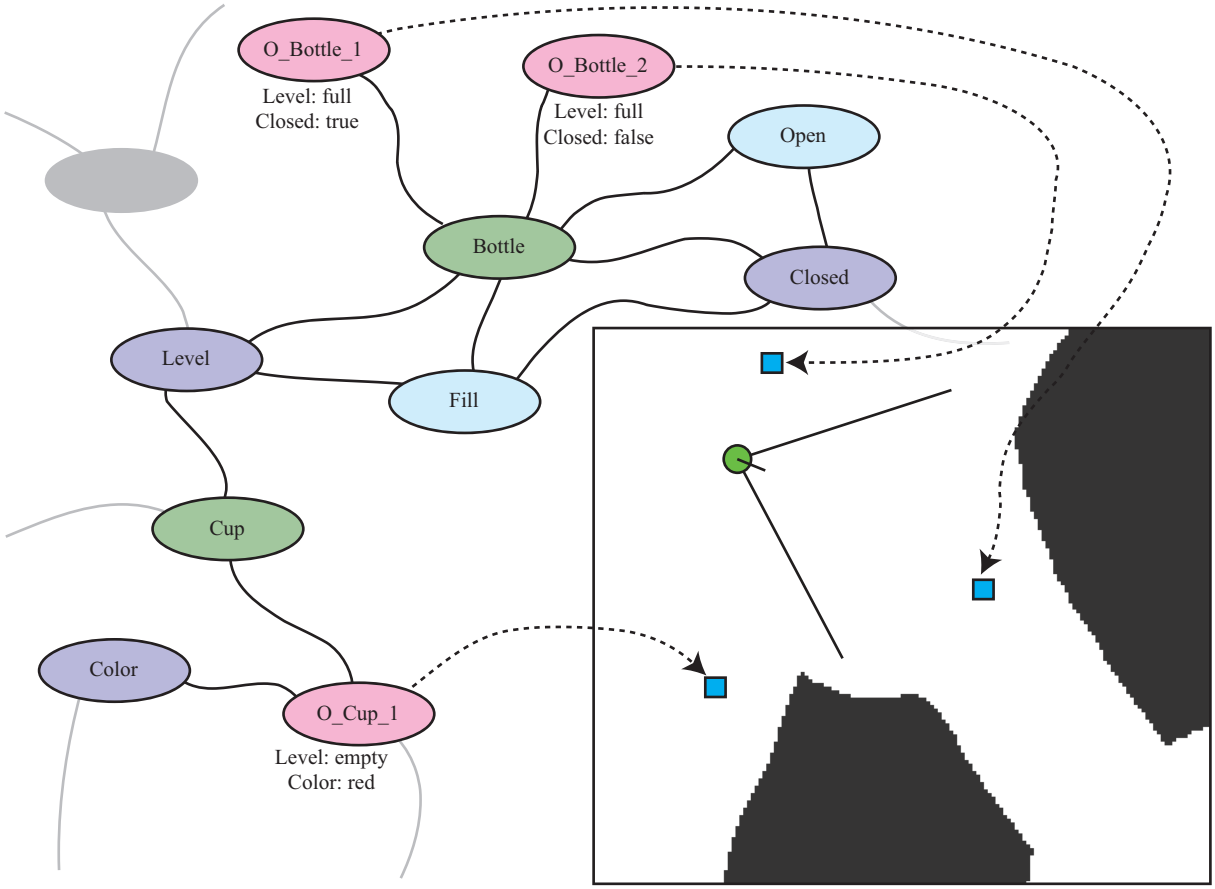


Fig. 5.8: Part of semantic map in an object-based view. The dashed lines illustrate connections between the semantic network and the metric map.

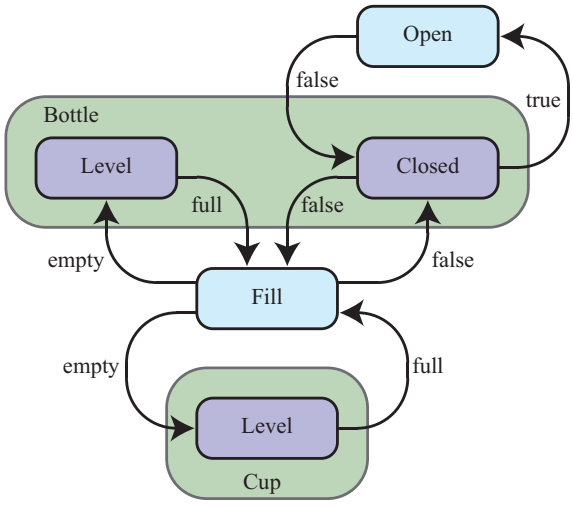


Fig. 5.9: State changes of the actions in an action-based view.

5.5 Experimental Results

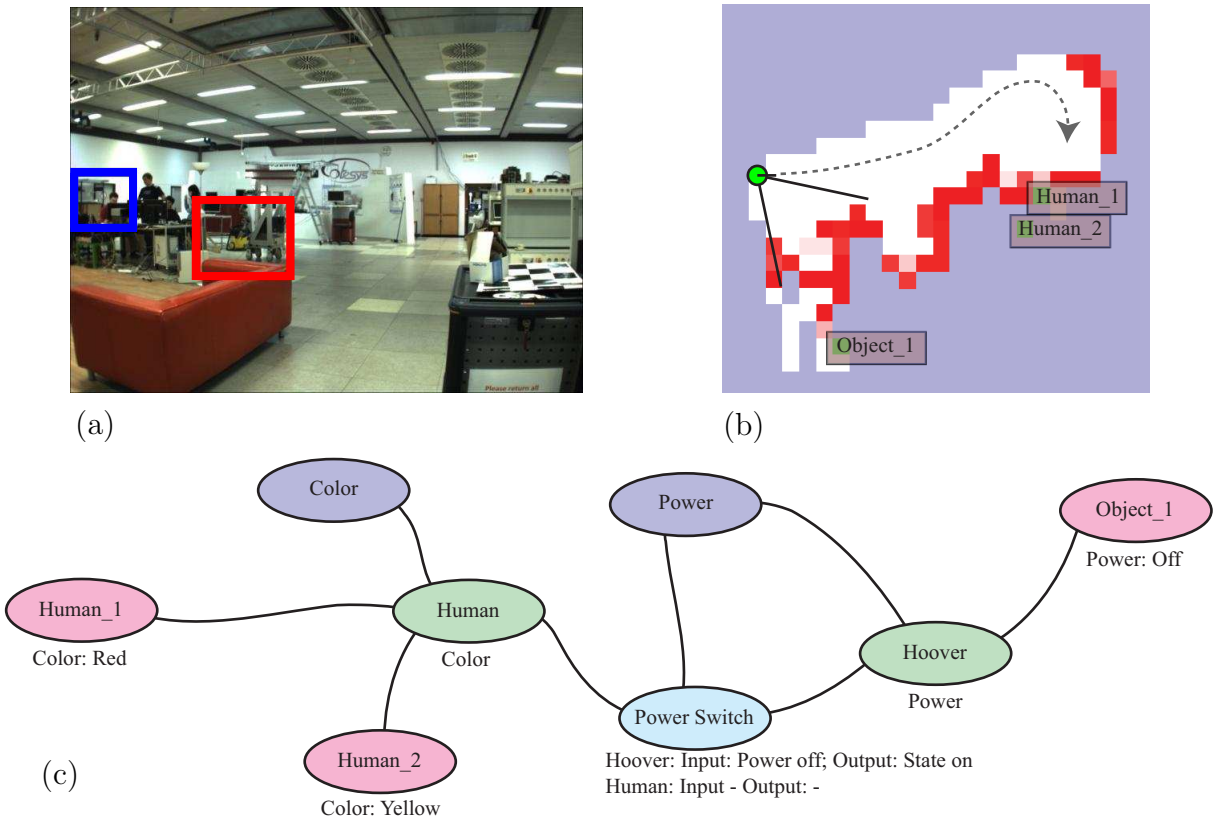


Fig. 5.10: Integration of mapping with object detection and two type of objects, humans and a vacuum cleaner. (a) shows an image scene taken at the start point, while (b) shows the metric map with the positions of the objects. The dashed line illustrates the path of the robot. (c) shows the corresponding semantic network.

Like the previous experiments, the following investigations have been conducted using the vision processing PC on the ACE robot, which was equipped with an AMD Phenom Quad-Core CPU running at 2.5 GHz, 4 GB of physical memory, and two GeForce 9800 GX2 cards and hence four Cuda enabled devices. Unless stated otherwise, only one Cuda device has been used for a module.
The camera was not aligned with the robots heading. In specific, it was rotated around the robots z-axis with an angle of 45°, thus covering a more interesting field of view. A bumblebee X3 camera with a field of view of 66° was used for this experiment. All three modules were executed on one PC, while one Cuda device was used for stereo processing and two of the other devices for object detection. Although three-dimensional mapping was only used for obstacle detection, the result three-dimensional map was stored for further use. The two-dimensional mapping was performed with a frame rate of 5 Hz, the three-dimensional mapping with 2 Hz, and the object detection with 7 Hz. A block size of 30 cm was used for the occupancy grid of the metric map. To enhance accuracy, the robot's odometry module was used to plot the position.

The object database contained two objects, a hoover and the upper part of a human body. Figure 5.10 (a) shows a part of the scene. The blue and red rectangle indicate the result of the object detection algorithm that was able to detect the vacuum cleaner and one of the humans. The two object-type nodes, human and hoover, the attribute-type nodes *color* and *power*, and the action-type node *power switch* have been added manually before the experiment was started. Figure 5.10 (b) shows the map-centered view of the semantic map generated on a short trip of ACE and Figure 5.10 (c) shows the object-centered view of the same semantic map. Three aspects of the system have been tested in the experiment: the two-dimensional mapping, the object detection, and the integration of the detected objects into the semantic map. In particular, the similarity condition of the semantic map could be investigated. Due to the camera distortion and uncertainties in the transformation of the objects position from image to robot coordinates, duplicate objects can occur easily. On the other hand, the scene contains two different humans and the system should be able to distinguish between them. In its current state of development, the two-dimensional object detection algorithm is not able to detect attributes of the objects. Consequently, the attribute *color* of the two objects of type human had to be entered manually, to be able to test the similarity probability. Furthermore, the threshold probability of similarity and the maximal distance have to be adjusted carefully. Summing up, the presented robot is able to distinguish between different objects, when they have different attributes.
Another experiment is presented in Figure 5.11. The experimental setup contained three chairs with the same attribute but with a large distance between them. Again, the attributes have been edited manually. As the distance between the objects was larger than $d_{\text{Max}} = 2$ m, the algorithm was able to distinguish between them. As shown in the third image of Figure 5.11 (a), two chairs with a small distance could not be distinguished and were hence detected as one object, namely Object_3.

5.6 Discussion

This chapter introduced a novel mathematical framework used to derive algorithms, which can be used to create a map of the environment containing semantic information. This so called semantic map consists of a metric map containing the necessary information for navigation and a semantic network, containing information about objects and their attributes. Algorithms for the computation of the similarity probability of two objects have been presented. Both the metric map and the semantic network are linked with each other. Through the introduction of state spaces for objects, actions can be planned and objects can be manipulated. Consequently, the system is capable of planning actions that change the state and thus attributes of objects. In addition, the state spaces can be used to compute a similarity probability of two objects, so the semantic mapping system is capable of dealing with uncertain measurements. Another main difference between the state-of-the-art and the semantic mapping system is that only one representation is sufficient to model the environment and thus for path and action planning.

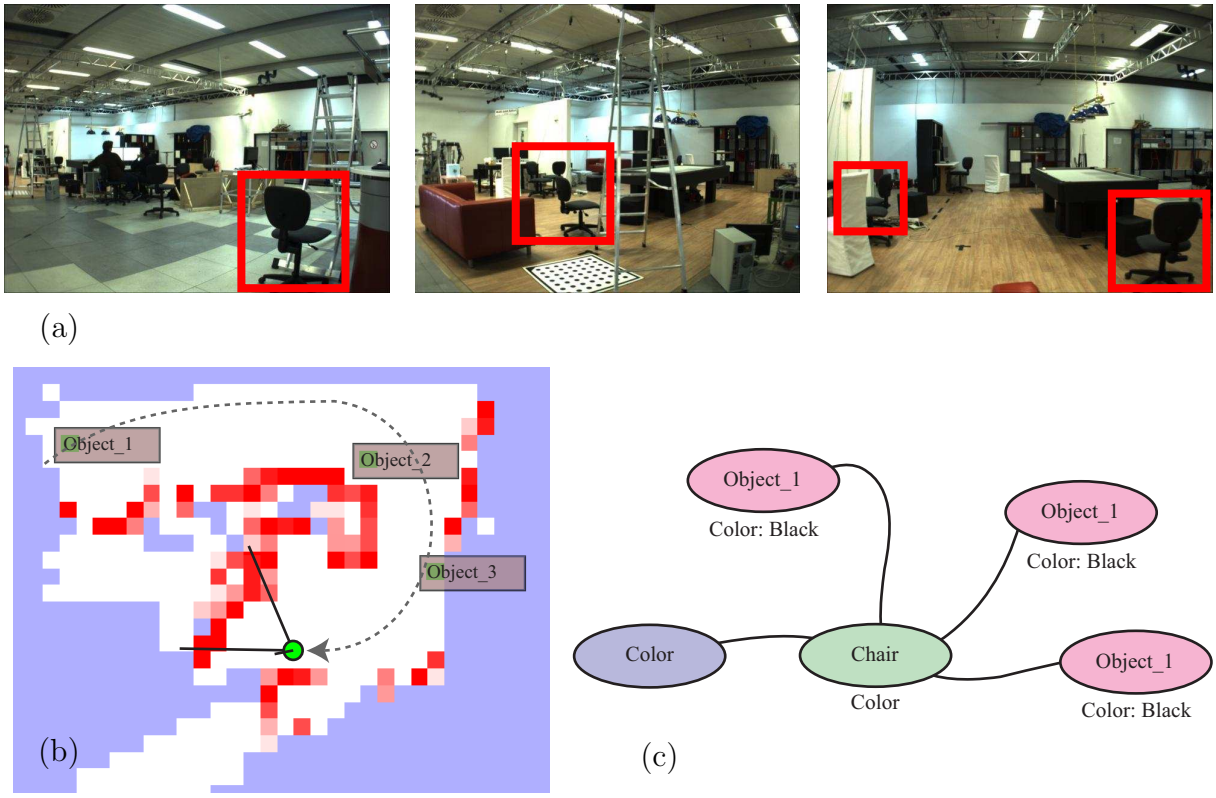


Fig. 5.11: Another example for the integration of mapping with object detection with multiple objects of one type, a chair. (a) shows an image scene taken at the start point, while (b) shows the metric map with the positions of the objects. The dashed line illustrates the path of the robot. (c) shows the corresponding semantic network.

An implementation of a semantic map was presented, containing a simulator, which was used to test the functionality of the presented algorithms. Furthermore, the implementation contains a complete framework for semantic mapping, composed out of a two- and three-dimensional mapping module, an object detection module, a semantic processing system controlling the input module, and a semantic robot containing the actual implementation of the semantic map. Due to standardized interfaces, the semantic processing system can be replaced by the simulator and each of the modules can be exchanged by a compatible one. New modules can be included without fundamental changes of the architecture.

Both simulation and experimental validations have been conducted to verify the functionality. The presented system was able to build an accurate semantic map containing useful information and the system was able to plan and perform complex actions in the simulation. The proposed methods can be used to increase the cognitive understanding of mobile robots working in close collaboration with humans. However, the current object detection algorithm is limited in the detection of the object's attributes. Consequently, the action planning could not be verified in a real case experiment. The robust detection of attributes and the experimental validation are subject to future research directions. Another aspect in future works is the introduction of a new kind of actions capable of identifying the type

of an object. Furthermore, the semantic mapping system is only capable of adding new nodes to the semantic network and of modifying existing connections. Methods to remove nodes of disappeared objects are also subject to future research.
As they are combining vision-based mapping and object detection with a sound knowledge representation and decision making system, semantic maps mark a sophisticated cognitive architecture.

6 Conclusion

6.1 Concluding Remarks

This thesis led through the development of a robot with powerful cognitive abilities, which are formed by a perception and a cognition part. Sophisticated cognitive abilities require a suitable knowledge representation and powerful algorithms for object detection, mapping, and decision making. The main approaches of this thesis are summarized below, together with the most important results.

ACE, a sophisticated mobile robot has been co-developed in the scope of this thesis and was used as hardware platform during the implementation process and for the experimental validation. The robot is equipped with a camera head, two stereo cameras with different resolution and different field of view, a touchscreen and loudspeakers for interaction, and two PCs for computation. A modular software architecture was designed, so that almost all modules of the perception and cognition system can be used isolated or can be replaced by another module with similar functionality. Therefore, a stringent hierarchy for the processes and unambiguous communication protocols have been developed. Consequently, the whole system can either be executed on one machine or can be distributed between several PCs. To guarantee the real-time capability of the system, a scheduler controlling the submodules and modules has been implemented in the semantic processing module.

As both perception parts utilize two- and three-dimensional input data, three-dimensional input data has to be generated. Moreover, robust object detection and mapping require input data in real-time and at a high resolution. Additionally, the ego-motion of the robot has to be known for mapping. Consequently, a sophisticated stereo and ego-motion estimation algorithm was developed using a novel implementation on Cuda, Nvidia's stream computing hardware. As stereo reconstruction and ego-motion estimation are computed simultaneously, this algorithm is able of performing the reconstruction of disparity images at a resolution of 640×480 with 150 disparities together with the ego-motion estimation at a frame rate of 10 Hz. Although it is not able to compute a full disparity map, it exceeds most existing algorithms regarding speed and number of computed disparities.

The stereo algorithm computes a colored point cloud, which is then used by one of the presented object detection algorithms, namely the novel human body-pose estimation, providing valuable information for interaction. The algorithm tries to fit a human model into the segmented point cloud, starting with the head. It can be extended to detect other skeleton based types of objects. As the fitting process starts with the head, the whole algorithm is constrained by the algorithm used to detect the start point. In addition to the three-dimensional algorithm, an algorithm using two-dimensional images was introduced. Like the stereo algorithm, it is implemented on Cuda, to ensure its real-time capability. A novel cascade of three types of histograms, namely color histograms, histograms of oriented gradients, and color co-occurrence histograms is used. A search window is moved over the whole image in every step of the cascade and a histogram is computed for every search window and then intersected with the reference histograms. Through careful selection of

the thresholds, the algorithm is able to deal with large occlusions. Another improvement compared to the state-of-the-art is the low number of necessary training images, allowing the algorithm to be used for online learning. The computation time for a whole cascade ranges between 40 ms and 140 ms per image, depending on the number of objects in an image. Hence, the algorithm is real-time capable. However, the unpredictability of the computational time may lead to problems with the scheduler.
Of equal importance are the two vision-based mapping systems. The two-dimensional system is based on a texture-based approach, where the texture of the image is compared to a reference texture of the ground. By introducing a texture memory containing valid matches, the algorithm is able to adopt to a changing ground and to recognize previously learned grounds. A metric map of the environment is created by using a back-projection algorithm. Although this algorithm runs with a frame rate of 15 Hz, it is not able to detect all obstacles. Sometimes there is an overhang or obstacles are of the same color as the ground. This leads to the requirement for a three-dimensional mapping system. The median fusion algorithm was implemented in a novel modular manner, so that each module can be replaced by another module of similar functionality. Even laser rangefinders can be used instead of the stereo camera or the robot's odometry instead of the ego-motion estimation, respectively. A genetic ICP algorithm was introduced, which is able to fuse a point cloud obtained by a laser rangefinder with a point cloud obtained by the stereo system. Consequently, the advantages of both can be utilized and a colored point cloud with a high quality can be generated. The resulting three-dimensional representation of the environment requires large amounts of memory, but can be used to detect obstacles or to plan complex actions for the manipulation of the environment.
Object detection and mapping have to be combined to a sound knowledge representation and a decision making system has to be available to form cognitive abilities. Therefore, semantic maps have been introduced, combining a metric map with a semantic network. Cells of the map can link to nodes of the network and vice versa. A sound mathematical background has been developed, revealing several classes of nodes: objects, types, attributes, and actions. By introducing a novel state space for every object, which is formed by the type of the object and its attributes, actions can be planned to manipulate the object. As the action planning is executed recursively, some dependencies in object states can be resolved. However, sudden changes of the object's state may lead to unpredictable behavior of the current implementation. The semantic processing module includes several submodules, which have been developed to construct and to utilize the semantic map. A path planner, a scheduler to ensure real-time capability, and a GUI to process user inputs and to display the current state of the system have been included. A simulator emulating a robot and the environment was used during the implementation process to verify the cognitive architecture and to test the decision making and action planning modules. Additional experiments have been performed to test the applicability. Parts of the perception process, the human body pose estimation, the two-dimensional mapping, and the obstacle detection in three dimensions have already been tested in an extensive outdoor experiment, when the ACE robot found its way to the Marienplatz, a central square in Munich.

By developing new algorithms and enhancing known ones, the research presented in this thesis contributes to advance the state-of-the-art in the development of cognitive architectures. Due to the modular design of the semantic mapping system, new algorithms can be included easily. Consequently, semantic mapping is a sustainable architecture for future research. Experiments have proven the functionality of semantic mapping, showing that the methods and results presented in this thesis serve as a stepping stone on the way to more cognitive and social robots.

6.2 Suggestions for Future Work

Constantly increasing computational power allows more complex software architectures, more sophisticated perception algorithms, and the arbitrary storage of large amounts of data. New sensors allow even more accurate representations of the environment using the full computational capacity. Due to its modularity and well defined interface, the presented architecture is an ideal base for future developments in the research field of cognitive abilities for mobile robots.
A sophisticated perception system is inevitable. Several aspects of the presented object detection research can be enhanced. As mentioned before, the body pose estimation algorithm can be used to detect other objects. Furthermore, the developed object detection cascade can be extended to detect certain attributes of objects, like the objects color. Therefore the color histogram step needs to be replaced with a more complex algorithm. With a large increase of available computational power, color co-occurrence histograms can even be extended to use three-dimensional images as input. Vision-based mapping is the other important part of perception. On the other hand, the quality of the resulting map is limited by the quality of the ego-motion estimation. Therefore, algorithms capable of dealing with difficult light conditions have to be developed. Colored three-dimensional point clouds as representation of the map provide valuable information - but require large amounts of memory. Hence, future research might deal with memory efficient representation based on a textured polygon grid and object models. Reliable algorithms for the computation of this grid have to be developed. Another interesting area of research is the combination of object detection and the computation of the grid.
As it is a relatively new research area in the field of mobile robotics, cognition holds the most interesting challenges. In a first step, the presented architecture can be enhanced with the mentioned suggestions, where the detection of attributes has the highest potential. Other research may deal with complex action and path planning algorithms using all three spatial dimensions. The increasing computational power allows the integration of more complex biologically inspired methods, like large-scale neuronal networks.
Modeling human-like behavior is a major challenge for future research and will increase the fascination for robotics. In a long term vision the cognitive abilities of robots will reach a point, where robots can interact naturally and have an amazing ability to understand the environment. However, they still won't possess a soul.

A System Overview of ACE

This chapter gives a short introduction to the software and hardware architecture of the ACE (autonomous city explorer) robot. Detailed information about the robots mission can be found in [12], while [74] and [92] provide information about the navigational subsystem and the vision system, respectively. The interaction system is introduced in [13].

A.1 Hardware

Figure A.1 shows the ACE robot and its main components. The robot is based on a differential wheel platform developed by BlueBotics SA[1]. An encapsulated PowerPC performs the low level control of the platform, providing basic navigational features and is connected to a laser rangefinder for navigation and obstacle avoidance. ACE has a length of 78 cm, a width of 56 cm, a height of 178 cm and a wight of approximately 160 kg. Its maximal speed is $1.4\frac{\mathrm{m}}{\mathrm{s}}$ and the maximal acceleration is $1.35\frac{\mathrm{m}}{\mathrm{s}^2}$.
Two linux PCs are located in the main chassis, one for navigation and interaction and the other PC for vision processing. This PC is equipped with two NVIDIA CUDA capable graphic cards, allowing massive parallel computing. The PCs are powered by eight lithium polymer batteries with a capacity of 1000 Wh each, thus achieving a total operation time of over six hours. Furthermore, the robot is equipped with a second, tilted laser rangefinder for traversability assessment. A loud speaker and a touch screen are used for the interaction with pedestrians. Another touchscreen and a keyboard for maintenance are mounted on the back of the robot and are hidden by a panel.
The right part of figure A.1 shows the sophisticated camera head with its two stereo cameras. A Bumblebee X2 camera by Point Grey Research Inc.[2] with a focal length of 2 mm and a field of view of 97°, which is mounted in a fixed pose, is used for the detection of the sidewalks and for human body pose estimation. Another Bumblebee X3 camera with a focal length of 3.8 mm and thus a small field of view of 66° is mounted on a pan/tilt platform and is used for human tracking and object detection. During interaction, this camera is orientated towards the human to enhance the acceptance. Both cameras are equipped with a servo driven artificial eyelid, which was developed to fulfill two tasks: improve interaction by blinking at the human and preventing the sun from shining directly into the camera, to avoid undesirable effects like blooming. The camera head can be accessed via a CAN and a USB interface.

A.2 Software Architecture

As seen in figure A.2, the software architecture of ACE can be separated into four main modules: *navigation*, *interaction*, *vision*, and *core*. The core modules provides basic func-

[1] `http://www.bluebotics.com`
[2] `http://www.ptgrey.com`

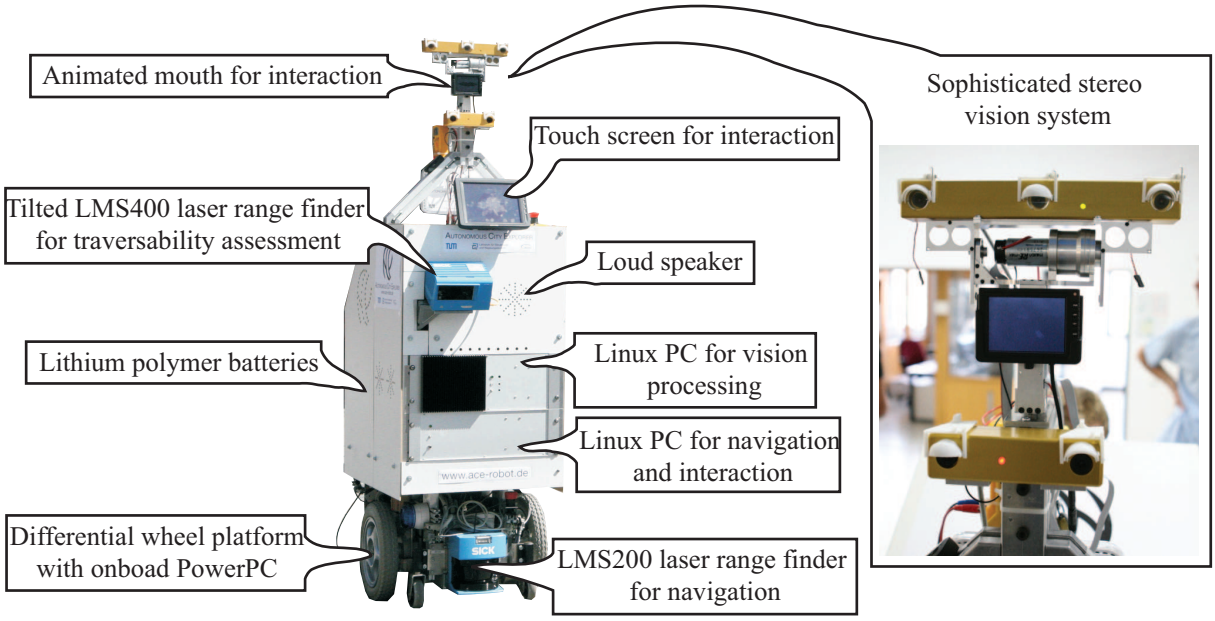


Fig. A.1: The hardware of the ACE-Robot and the sophisticated camera head with two stereo cameras and the animated mouth during a blinking process.

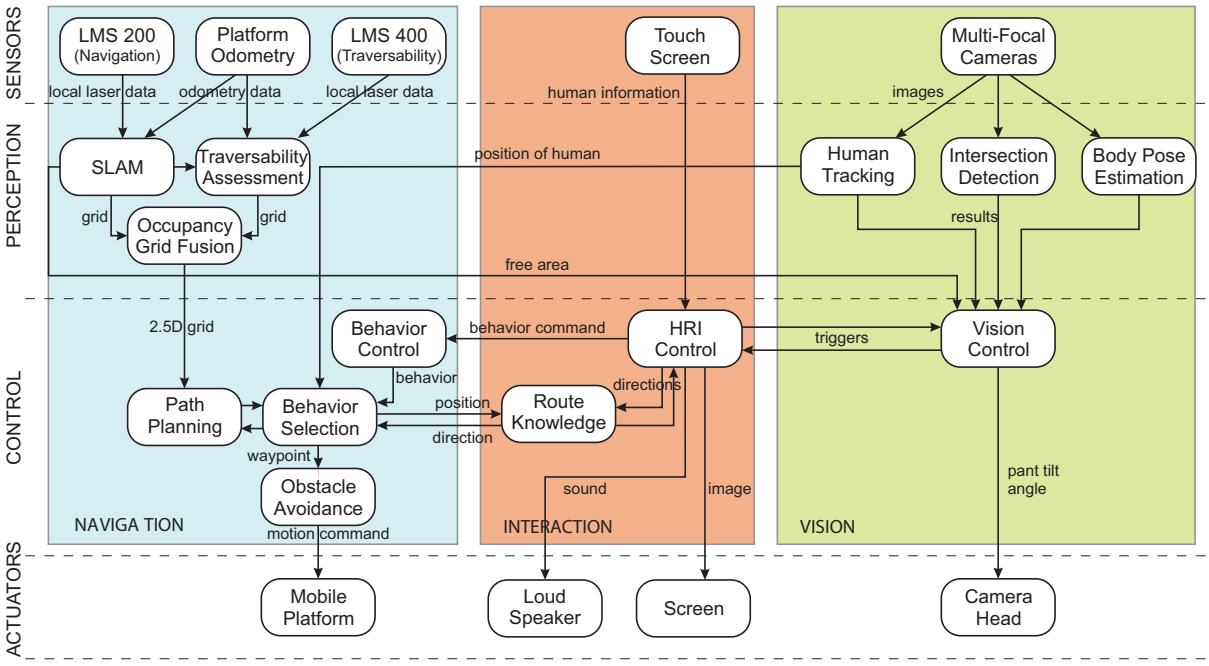


Fig. A.2: The software architecture of the ACE-Robot. Modules are indicated by boxes, subsystems by vertical shaded regions and layers are separated by horizontal dashed lines. [12]

tionality like a communication protocol or fundamental data structures. The navigation module uses one of the laser rangefinders for traversability assessment, while the other one is used for simultaneous mapping and localization (SLAM). This yields a two-dimensional map, which can be used for navigation. Different navigational behaviors are available. Possible commands are *drive to x y* or *rotate by p* and *drive straight ahead for x m*.
The vision module controls the camera head and processes the data recorded by the cameras. It possesses different states. When the tracking mode is active, the vision module accesses the driving commands of the navigation module directly. Other modes include the detection of human body poses and a search mode that sends triggers, when an intersection or other important objects are detected. Both the navigation and vision module are controlled by the interaction module, which controls the speaker and the touchscreen and uses input from the user, the navigation and the vision module to select the robots navigational behavior and compute position commands.
Some of the methods presented in this thesis have been used for the ACE-project, namely the stereo module, the human body pose estimation, the two-dimensional mapping and the two-dimensional object detection in the intersection detection module.

B Introduction to CUDA

While conventional CPUs are able to compute many different operations on different types of data and provide up to four arithmetic logic units (ALUs), the main task of graphics cards is to perform graphics rendering based on polygons. As each polygon can be computed separately, graphics cards provide many ALUs and are only able to compute few different operations on few types of data. However, simple operations are computed in a vast amount of parallel threads. By providing easy access to parallel computing, the introduction of CUDA established a whole new branch in general-purpose computing on graphics processing units (GPGPU). Writing programs for CUDA is quite different from classical programming. Thus, an overview of the architecture of CUDA and the developed programs is provided in this chapter.

B.1 Overview of CUDA

CUDA comes with an API based on the language C and provides a complete framework for the development. CUDA programs can be separated into two parts, the main loop is executed on the CPU, also called *host*, while the actual threads are executed on the GPU, also called *device*. As a thread is a relatively simple function, more complex algorithms have to be split into several execution steps. Up to 512 threads compose a block, while a grid holds an arbitrary number of blocks. When starting a thread, the size of the grid and the number of threads in each block have to be defined. All threads contained in one block will be executed in parallel, while the different blocks can also be computed consecutively. The CUDA API provides built in variables to get the current position of the thread in the block and in the grid, so that the thread knows the memory it is supposed to process.

Tab. B.1: Comparison of the different memory types

Type	Size	Latency	Device Access	Host Access
Register	$<$ 1 KB (per Thread)	1 cycle	read & write	-
Local	$\sim$ 64 KB (per Thread)	$\sim$ 500 cycles	read & write	-
Shared	16 KB (per Block)	1 cycle	read & write	-
Constant	64 KB	Cached	read	write
Texture	$>$ 256 MB	Cached	read	read & write
Global	$>$ 256 MB	$\sim$ 500 cycles	read & write	read & write

Different memory types for different purposes are provided, ranging from small and fast to large and slow memories. Table B.1 shows and compares these different types. Registers provide the fastest type of memory. However, only a very small number is available per thread. If a thread requires more memory, the slow local memory has to be used. Registers

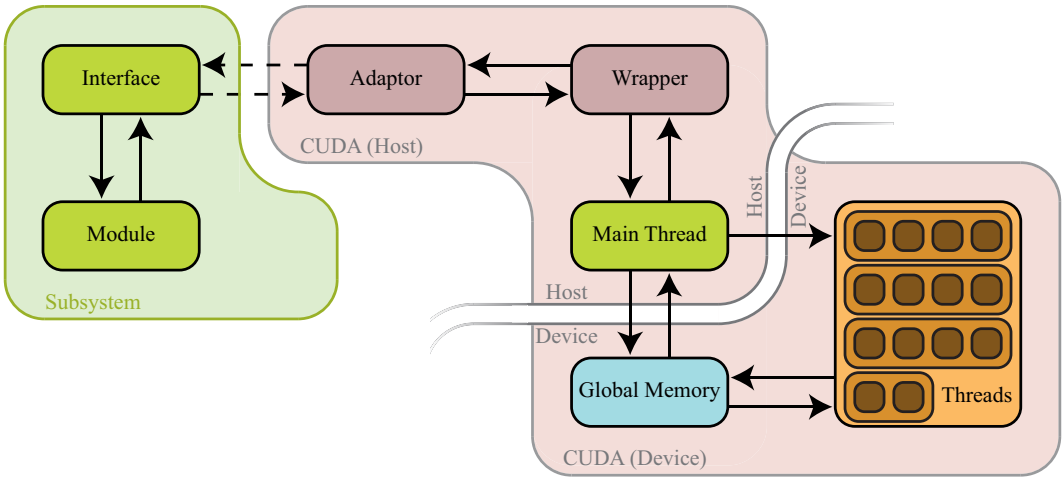


Fig. B.1: Architecture of the CUDA interface. Dashed lines indicate a communication protocol based on UDP or D-Bus.

and local memory can only be accesses by a single thread, while global memory can be accesses by all threads and functions executed on the host. Hence, global memory is an ideal base for the transfer of data. Shared memory can be used for the communication between the threads in one block. However, simultaneous writing is not allowed, leading to threads blocking other threads and thus a longer execution time. Constant and texture memory provide a large amount of memory that is cached, and thus fast to access.

B.2 Integration of a CUDA Application

All programs using the CUDA technology have to be built with NVIDIA's toolchain, complicating the integration into another framework. Therefore, all modules using CUDA have been split into two parts: an interface and an own process. While the interface conducts the communication with the other modules in a subsystem, the process conducts the actual computation and communicates via a UDP network connection or D-Bus with the interface, depending on whether the process is run on the same system or is distributed. Figure B.1 shows the architecture of the interface and the CUDA process, which can be separated into further submodules. The adaptor communicates with the interface, while the wrapper pre-processes the the data for the device. The main thread controls the actual threads and distributes the data to the memory.

C Computation of the Camera Motion

The motion of the camera can be computed directly by examining the disparity without transforming the disparity into cartesian coordinates [30]:

$$\begin{pmatrix} \mathbf{w} \\ 1 \end{pmatrix} \simeq \Gamma \cdot \begin{pmatrix} {}_c\mathbf{m} \\ 1 \end{pmatrix}, \tag{C.1}$$

with

$$\mathbf{w} = \begin{pmatrix} x \\ y \\ \delta \end{pmatrix} = \begin{pmatrix} p_x \cdot (u_0 - i) \\ p_y \cdot (v_0 - j) \\ d \cdot p_x \end{pmatrix}, \quad \mathbf{m} = \begin{pmatrix} {}_cX \\ {}_cY \\ {}_cZ \end{pmatrix}, \quad \Gamma = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 0 & fB \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{C.2}$$

p_x and p_y denote the pixel size, while u_0 and v_0 denote the center of the camera, B the stereo basis, and f the focal length. The ego-motion of the camera in disparity space can be described as:

$$\begin{pmatrix} \hat{\mathbf{w}} \\ 1 \end{pmatrix} = \Gamma \cdot \mathbf{T} \cdot \Gamma^{-1} \cdot \begin{pmatrix} \mathbf{w} \\ 1 \end{pmatrix}, \tag{C.3}$$

with the transformation matrix T, which contains the rotation and translation of the robot. $\mathbf{w}$ contains the pixel and disparity data of the previous image, while $\hat{\mathbf{w}}$ contains the disparity data of the current image. The computation can be simplified by using the small-angle approximation:

$$T = \left(\begin{array}{ccc|c} \cos(\alpha) & 0 & \sin(\alpha) & {}_ct_x \\ 0 & 1 & 0 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) & {}_ct_z \\ \hline 0 & 0 & 0 & 1 \end{array}\right) = \left(\begin{array}{ccc|c} 1 & 0 & \alpha & {}_ct_x \\ 0 & 1 & 0 & 0 \\ -\alpha & 0 & 1 & {}_ct_z \\ \hline 0 & 0 & 0 & 1 \end{array}\right), \tag{C.4}$$

while α denotes the rotation around the camera's y axis and ${}_ct_x$ and ${}_ct_z$ the translation of the camera. Equation C.3 can be solved by assuming the transformation error $\mathbf{e}$ to be $\mathbf{e} = 0$:

$$-(x + \frac{f^2}{\hat{x}})\alpha + \frac{\delta}{B}{}_ct_z - \frac{f\delta}{B\hat{x}}{}_ct_x = f(\frac{x}{\hat{x}} - 1), \tag{C.5a}$$

$$-\alpha x - \frac{\delta}{B}{}_ct_z = f(\frac{x}{\hat{x}} - 1), \tag{C.5b}$$

$$-\alpha x - \frac{\delta}{B}{}_ct_z = f(\frac{x}{\hat{x}} - 1). \tag{C.5c}$$

By using the matrix vector notation, the movement for all n corners can be described as:

$$\begin{pmatrix} -(x_1 + \frac{f^2}{\hat{x}_1}) & -\frac{f\delta_1}{B\hat{x}_1} & \frac{\delta_1}{B} \\ -x_1 & 0 & \frac{\delta_1}{B} \\ -x_1 & 0 & \frac{\delta_1}{B} \\ \vdots & \vdots & \vdots \\ -(x_n + \frac{f^2}{\hat{x}_n}) & -\frac{f\delta_n}{B\hat{x}_n} & \frac{\delta_n}{B} \\ -x_n & 0 & \frac{\delta_n}{B} \\ -x_n & 0 & \frac{\delta_n}{B} \end{pmatrix} \cdot \begin{pmatrix} \alpha \\ {}_c t_x \\ {}_c t_z \end{pmatrix} = \begin{pmatrix} f(\frac{x_1}{\hat{x}_1} - 1) \\ f(\frac{y_1}{\hat{y}_1} - 1) \\ f(\frac{\delta_1}{\hat{\delta}_1} - 1) \\ \vdots \\ f(\frac{x_n}{\hat{x}_n} - 1) \\ f(\frac{y_n}{\hat{y}_n} - 1) \\ f(\frac{\delta_n}{\hat{\delta}_n} - 1) \end{pmatrix}, \tag{C.6}$$

$$\mathbf{D} \cdot \begin{pmatrix} \alpha \\ {}_c t_x \\ {}_c t_z \end{pmatrix} = \mathbf{p}. \tag{C.7}$$

Now, Equation C.7 can be solved by using the method of least squares:

$$\begin{pmatrix} \alpha \\ {}_c t_x \\ {}_c t_z \end{pmatrix} = (\mathbf{D}^T \cdot \mathbf{D})^{-1} \cdot \mathbf{D}^T \cdot \mathbf{p}. \tag{C.8}$$

As this computes the movements of the objects relative to the camera, the signs of α, ${}_c t_x$, and ${}_c t_z$ have to be changed. Furthermore, the movement of the camera ${}_c\mathbf{m}$ has to be transformed into the movement of the robot ${}_r\mathbf{m}$:

$${}_r\mathbf{m} = {}^r T_c \cdot {}_c\, \mathbf{m}, \tag{C.9}$$

while ${}^r T_c$ denotes the position and orientation of the camera relative to the robot.

List of Figures

1.1 Illustration of perception 2
1.2 Cognitive architecture simulating a biological neuronal network 4
1.3 Architecture of a symbolic processing system 5
1.4 The Robot iCub 6
1.5 Illustration of a symbolic processing system 9

2.1 Approximate trajectory of ACE with four stations 12
2.2 System architecture of a semantic mapping system 13
2.3 Hierarchy of the software architecture 14

3.1 Geometry of a stereo system 23
3.2 Comparison of the different aggregation methods 25
3.3 Result of the reduction algorithm with different parameters 28
3.4 Example of the Segmentation Process 30
3.5 Reduced human model with 15 links 33
3.6 Illustration of the link fitting algorithm 36
3.7 Traffic signs used for training of the SIFT algorithm 38
3.8 Illustration of the haar-like-cascade 39
3.9 Architecture of the Object Detection Cascade. 40
3.10 Computation of a color histogram in a reduced color space 41
3.11 Computation of a histogram of oriented gradients 41
3.12 Computation of a color co-occurrence histogram 42
3.13 Intersection of two histograms 44
3.14 Result of the stereo processing system 47
3.15 Correlation between window size, miss-mismatches, and computation time 48
3.16 Computed body pose 48
3.17 Results of the human body pose estimation. 49
3.18 Correlation between stage, miss-rate, and false positives 50
3.19 Result of the object detection cascade 51
3.20 Result of the intersect with different histogram types 52
3.21 Correlation between detected occlusions, miss-rate, and false positives . . . 53

4.1 Architecture of the two-dimensional mapping module 59
4.2 Processing steps of the ground detection algorithm 60
4.3 Histogram during a texture analysis 60
4.4 Extrinsic and intrinsic camera parameters used for the back projection . . 62
4.5 Illustration of the different coordinate systems 63
4.6 Reconstruction of a crossroad 64
4.7 Software architecture of the scene reconstruction module 65
4.8 Processing steps of the scene reconstruction module 66
4.9 Illustration of the median fusion algorithm with the two error models . . . 70

4.10 Obstacle detection in 3D Maps 71
4.11 The different coordinate systems needed for calibration 73
4.12 Setup of a calibration-scene 73
4.13 Scan of the calibration-scene 74
4.14 Plot of the distance metric 78
4.15 Result of the mapping algorithm over a long distance 79
4.16 Effect of bad light conditions 80
4.17 Computation of the optical flow 80
4.18 Result of the ego-motion estimation 82
4.19 Result of the pre-processing with the used parameter settings 82
4.20 Result of the median fusion algorithm 83
4.21 Comparison between reconstruction from image data and ground truth . . 84
4.22 Input for the fusion algorithm 85
4.23 Result of the genetic ICP algorithm 85
4.24 Correlation of parameters of the genetic ICP algorithm 86

5.1 Example for a semantic network 90
5.2 Illustration of a semantic network and its subsets 94
5.3 Illustration of objects, actions and the corresponding state changes 99
5.4 Illustration of action planning 99
5.5 Architecture of the simulator and the semantic processing 100
5.6 Architecture of the semantic mapping system 101
5.7 Finite State Machine of a semantic robot 102
5.8 Part of semantic map in an object-based view 104
5.9 State changes of the actions in an action-based view. 104
5.10 Result of the integrated semantic mapping system 105
5.11 Result of the integrated semantic mapping system 107

A.1 The hardware of the ACE-Robot 114
A.2 The software architecture of the ACE-Robot 114

B.1 Architecture of the CUDA interface 118

Bibliography

[1] S. Adalıand L. Pigaty. The darpa advanced logistics project. *Annals of Mathematics and Artificial Intelligence*, 37:409–452, 2003.

[2] A. Akbarzadeh, J.-M. Frahm, P. Mordohai, B. Clipp, C. Engels, D. Gallup, P. Merrell, M. Phelps, S. Sinha, B. Talton, L. Wang, Q. Yang, H. Stewenius, R. Yang, G. Welch, H. Towles, D. Nister, and M. Pollefeys. Towards urban 3d reconstruction from video. In *Proceedings of the International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT)*, pages 1–8, 2006.

[3] B. Allen, B. Curless, and Z. Popovi. The space of human body shapes: reconstruction and parameterization from range scans. In *Proceedings of the International Conference on Computer Graphics and Interactive Techniques (ACM SIGGRAPH)*, pages 587–594, 2003.

[4] J. R. Anderson, M. Byrne, S. Douglass, C. Lebiere, and Y. Qin. An integrated theory of the mind. *Psychological Review*, 111(4):1036–1050, 2004.

[5] S. Arya, D. Mount, N. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. In *Proceedings of the annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 573–582, 1994.

[6] I. Asimov. *I, Robot*. Gnome Press, 1950.

[7] K. Atul, S. Cristian, and M. Dimitris. Semi-supervised hierarchical models for 3d human pose reconstruction. In *Proceedings of the Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, 2007.

[8] H. Bang, S. Lee, D. Yu, and I. Suh. Robust object recognition using a color co-occurrence histogram and the spatial relations of image patches. *Artificial Life and Robotics*, 13(2):488–492, 2009.

[9] Y. Bar-Yam. *Dynamics of Complex Systems*. Westview Press, 2003.

[10] J. L. Barron, D. J. Fleet, and S. S. Beauchemin. Performance of optical flow techniques. *International Journal of Computer Vision*, 12(1):43–77, 1994.

[11] A. Bauer, K. Klasing, G. Lidoris, Q. Mühlbauer, F. Rohrmüller, S. Sosnowski, T. Xu, K. Kühnlenz, D. Wollherr, and M. Buss. The autonomous city explorer project (video). In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2009.

[12] A. Bauer, K. Klasing, G. Lidoris, Q. Mühlbauer, F. Rohrmüller, S. Sosnowski, T. Xu, K. Kühnlenz, D. Wollherr, and M. Buss. The autonomous city explorer: Towards natural human-robot interaction in urban environments. *International Journal of Social Robotics*, 1(2):127–140, 2009.

[13] A. Bauer, D. Wollherr, and M. Buss. Information retrieval system for human-robot communication: asking for directions. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 1522–1527, 2009.

[14] P. J. Besl and H. D. Mckay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[15] P. Biber, H. Andreasson, T. Duckett, and A. Schilling. 3d modeling of indoor environments by a mobile robot with a laser scanner and panoramic camera. In *International Conference on Intelligent Robots and Systems (IROS)*, 2004.

[16] B. Boulay, F. Bremond, and M. Thonnat. Posture recognition with a 3d human model. In *Proceedings of the IEEE International Symposium on Imaging for Crime Detection and Prevention (ICDP)*, pages 135–138, 7-8 June 2005.

[17] N. Bourbakis and R. Andel. Fusing laser and image data for 3d perceived space representations. In *Proceedings of the International Conference on Tools with Artificial Intelligence*, page 50, 1997.

[18] R. Bowden, T.A. Mitchell, and M. Sarhadi. Reconstructing 3d pose and motion from a single camera view. In *Proceedings of the British Machine Vision Conference*, pages 904–913, 1998.

[19] C. Bregler and J. Malik. Tracking people with twists and exponential maps. In *Proceedings of the Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8–15, 23-25 Jun 1998.

[20] P. Buschka and R. Saffiotti. Some notes on the use of hybrid maps for mobile robots. In *Proceedings of the International Conference on Intelligent Autonomous Systems (IAS)*, pages 547–556, 2004.

[21] J. Bückner, S. Müller, M. Pahl, and O. Stahlhut. Semantic interpretation of remote sensing data. In *International Archives of Photogramming and Remote Sensing*, pages 62–66, 2002.

[22] K. Čapek. *R.U.R. (Rossum's Universal Robots)*. Penguin Classics, 2004.

[23] P. Chang and J. Krumm. Object recognition with color cooccurrence histogram. In *Proceedings of the Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 1999.

[24] A. Chariot and R. Keriven. Gpu-boosted online image matching. In *Proceedings of the International Conference on Pattern Recognition (ICPR)*, pages 1–4, 2008.

[25] Y.D. Chen, J. Ni, and S.M. Wu. Dynamic calibration and compensation of a 3d laser radar scanning system. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 652–658, 1993.

[26] H.I. Christensen, A. Sloman, G-J. Kruijff, and J. Wyatt. *Cognitive Systems: Final Report of the CoSy Project*. Springer Verlag, 2009.

[27] H. Costelha and R. Lima. Modelling, analysis and execution of robotic tasks using petri nets. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 1187–1190, 2007.

[28] J. Coughlan and H. Shen. Terrain analysis for blind wheelchair users: Computer vision algorithms for finding curbs and other negative obstacles. In *Proceedings of the Conference & Workshop on Assistive Technologies for People with Vision & Hearing Impairments*, 2007.

[29] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proceedings of the Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 886–893, 2005.

[30] D. Demirdjian and T. Darrell. Motion estimation from disparity images. In *Proceedings of the International Conference on Computer Vision*, pages 213–218, 2001.

[31] A. Elgammal, A. Elgammal, and Chan-Su Lee. Inferring 3d body pose from silhouettes using activity manifold learning. In *Proceedings of the Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 681–688, 2004.

[32] C. Estrada, J. Neira, and J.D. Tardos. Hierarchical slam: Real-time accurate mapping of large environments. *IEEE Transactions on Robotics and Automation*, 21(4):588–596, 2005.

[33] E. B. Fernandez and X. Yuan. Semantic analysis patterns. In *International Conference on Conceptual Modeling / the Entity Relationship Approach*, pages 183–195, 2000.

[34] J.F. Ferreira, P. Bessiére, K. Mekhnacha, J. Lobo, J. Dias, and C. Laugier. Bayesian models for multimodal perception of 3d structure and motion. In *International Conference on Cognitive Systems (CogSys)*, 2008.

[35] S. Franklin. A foundational architecture for artificial general intelligence. In *Proceeding of the Conference on Advances in Artificial General Intelligence: Concepts, Architectures and Algorithms*, pages 36–54, 2007.

[36] C. Früh and A. Zakhor. An automated method for large-scale, ground-based city model acquisition. *International Journal on Computer Vision*, 60(1):5–24, 2004.

[37] P. Fua. Reconstructing complex surfaces from multiple stereo views. In *Proceedings of the Fifth International Conference on Computer Vision*, pages 1078–1085, 1995.

[38] T. Funkhouser and P. Shilane. Partial matching of 3d shapes with priority-driven search. In *Proceedings of the fourth Eurographics Symposium on Geometry Processing*, pages 131–142, 2006.

[39] C. Galindo, J. A. Fernández-Madrigal, J. González, and A. Saffiotti. Using semantic information for improving efficiency of robot task planning. In *Proceedings of the ICRA Workshop on Semantic Information in Robotics*, pages 955–966, 2007.

[40] C. Galindo, J. A. Fernández-Madrigal, J. González, and A. Saffiotti. Robot task planning using semantic maps. *Robotics and Autonomous Systems*, 56(11):955–966, 2008.

[41] D. M. Gavrila and L. S. Davis. 3-d model-based tracking of humans in action: a multi-view approach. In *Proceedings of the Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 73–81, 1996.

[42] J. Giarratano and G. Riley. *Expert Systems: Principles and Programming*. Brooks/Cole Publishing Co., 1989.

[43] M. Goesele, B. Curless, and S. M. Seitz. Multi-view stereo revisited. In *Proceedings of the Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2402–2409, 2006.

[44] M. Gong, R. Yang, L. Wang, and M. Gong. A performance study on different cost aggregation approaches used in real-time stereo matching. *International Journal of Computer Vision*, 75(2):283–296, 2007.

[45] G. Granlund. A Cognitive Vision Architecture Integrating Neural Networks with Symbolic Processing. *Künstliche Intelligenz*, (2):18–24, 2005. ISSN 0933-1875.

[46] O. Grau. A scene analysis system for the generation of 3-d models, 1997.

[47] M. Habbecke and L. Kobbelt. A surface-growing approach to multi-view stereo reconstruction. In *Proceedings of the Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, 2007.

[48] R. M. Haralick, K. Shanmugam, and I. Dinstein. Textural features for image classification. In *IEEE Transactions on Systems, Man and Cybernetics*, volume 3, pages 610–621, 1973.

[49] C. Havasi, R. Speer, and J. Alonso. Conceptnet 3: a flexible, multilingual semantic network for common sense knowledge. In *Recent Advances in Natural Language Processing*, 2007.

[50] B. Heisele, T. Serre, and T. Poggio. A component-based framework for face detection and identification. *International Journal of Computer Vision*, 74(2):167–181, 2007.

[51] S. Hilsenbeck. Rekonstruktion der Umgebung aus einer Sequenz von Stereobildern, Studienarbeit an der Technischen Universität München, 2009.

[52] M. Hiroshi and N. Shree K. Visual learning and recognition of 3-d objects from appearance. *International Journal on Computer Vision*, 14(1):5–24, 1995.

[53] A. Hogue, A. German, J. Zacher, and M. Jenkin. Underwater 3d mapping: Experiences and lessons learned. In *The 3rd Canadian Conference onComputer and Robot Vision*, pages 24–24, 2006.

[54] J. Hopfield. *Feynman and computation: exploring the limits of computers*, chapter Neural networks and physical systems with emergent collective computational abilities, pages 7–19. Perseus Books, 1999.

[55] B. K. P. Horn. Extended gaussian images. *Proceedings of the IEEE*, 72:1671–1686, 1984.

[56] B. K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America*, 4(4):629–642, 1987.

[57] J. Horn, A. Bachmann, and Thao Dang. Stereo vision based ego-motion estimation with sensor supported subset validation. In *IEEE on Intelligent Vehicles Symposium*, pages 741–748, 2007.

[58] N. R. Howe, M.E. Leventon, and W. T. Freeman. Bayesian reconstruction of 3d human motion from single-camera video. *Advances in Neural Information Processing Systems*, 12:820–826, 2000.

[59] G. Jin, S. Lee, J. K. Hahn, S. Bielamowicz, R. Mittal, and R. Walsh. 3d surface reconstruction and registration for image guided medialization laryngoplasty. *Lecture Notes in Computer Science*, 4291:761–770, 2006.

[60] A. E. Johnson and S. B. Kang. Registration and integration of textured 3-d data. In *Proceedings of the International Conference on Recent Advances in 3-D Digital Imaging and Modeling*, page 234, 1997.

[61] T. Jost and H. Hugli. A multi-resolution scheme icp algorithm for fast shape registration. *First International Symposium on 3D Data Processing Visualization and Transmission*, pages 540–543, 2002.

[62] M. Kaess and F. Dellaert. A Markov chain Monte Carlo approach to closing the loop in SLAM. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 645–650, 2005.

[63] P. Karp. The design space of frame knowledge representation systems. Technical Report 520, SRI International AI Center, 333 Ravenswood Ave, Menlo Park, CA 94025, Oct 1992.

[64] K. Kayama, I. Eguchi, and S. Igi. Detection of sidewalk border using camera on low-speed buggy. In *Proceedings of the 25th conference on Proceedings of the 25th IASTED International Multi-Conference*, pages 238–243, 2007.

[65] A. Klaus, M. Sormann, and K. Karner. Segment-based stereo matching using belief propagation and a self-adapting dissimilarity measure. In *Proceedings of the 18th International Conference on Pattern Recognition*, pages 15–18, 2006.

[66] S. Knoop, S. Vacek, and R. Dillmann. Sensor fusion for 3d human body tracking with an articulated 3d body model. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 1686–1691, 2006.

[67] R. Koch. 3-d scene modeling from stereoscopic image sequences. In *Proceedings European Workshop on Combined Real and Synthetic Image Processing for Broadcast and Video Production*, 1994.

[68] K. Krawiec, D. Howard, and M. Zhang. Overview of object detection and image analysis by means of genetic programming techniques. *Frontiers in the Convergence of Bioscience and Information Technologies*, pages 779–784, 2007.

[69] S. Krishnan, R. Lee, J. B. Moore, and S. Venkatasubramanian. Global registration of multiple 3d point sets via optimization-on-a-manifold. In *Proceedings of the third Eurographics symposium on Geometry processing*, page 187, 2005.

[70] B. Kuipers and Y. T. Byun. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Journal of Robotics and Autonomous Systems*, 8:47–63, 1993.

[71] M. Lalonde, D. Byrns, L. Gagnon, N. Teasdale, and D. Laurendeau. Real-time eye blink detection with gpu-based sift tracking. In *Proceedings of the Fourth Canadian Conference on Computer and Robot Vision*, pages 481–487, 2007.

[72] C. Langis, M. Greenspan, and G. Godin. The parallel iterative closest point algorithm. *Third International Conference on 3-D Digital Imaging and Modeling*, 00:195, 2001.

[73] J. Lehman, J. Laird, and P. Rosenbloom. A gentle introduction to soar, an architecture for human cognition. In *In S. Sternberg & D. Scarborough (Eds), Invitation to Cognitive Science*, 1996.

[74] G. Lidoris, F. Rohrmüller, D. Wollherr, and M. Buss. The autonomous city explorer (ace) project: mobile robot navigation in highly populated urban environments. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 2238–2244, 2009.

[75] R. Lienhart and J. Maydt. An extended set of haar-like features for rapid object detection. In *Proceedings of the International Conference on Image Processing*, pages 900–903, 2002.

[76] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 2004.

[77] B.D. Lucas. Generalized image matching by the method of differences. In *Robotics Institute, Carnegie Mellon University*, 1984.

[78] Y. Ma, Z. Wang, M. Bazakos, and W. Au. 3d scene modeling using sensor fusion with laser range finder and image sensor. In *Proceedings of the 34th Applied Imagery and Pattern Recognition Workshop (AIPR)*, pages 224–229, 2005.

[79] A. Makadia, A. Patterson, and K. Daniilidis. Fully automatic registration of 3d point clouds. In *Proceedings of the Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1297–1304, 2006.

[80] Cornelius Malerczyk. 3d-reconstruction of soccer scenes. In *International Conference on 3DTV*, pages 1–4, 2007.

[81] O. Martínez Mozos, P. Jensfelt, H. Zender, G. J. Kruijff, and Burgard W. From labels to semantics: An integrated system for conceptual spatial representations of indoor environments for mobile robots. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2007.

[82] L. Matthies, Y. Xiong, R. Hogg, D. Zhu, A. Rankin, B. Kennedy, M. Hebert, R. Maclachlan, C. Won, and T. Frost. A portable, autonomous, urban reconnaissance robot. *Robotics and Autonomous Systems*, 40:163–172, 2002.

[83] D. Meger, P. E. Forssén, K. Lai, S. Helmer, S. McCann, T. Southey, M. Baumann, J. J. Little, and D. G. Lowe. Curious george: An attentive semantic robot. *Robotics and Autonomous Systems*, 56(6):503–511, 2008.

[84] C. Mei and P. Rives. Calibration between a central catadioptric camera and a laser range finder for robotic applications. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 532–537, 2006.

[85] P. Merrell, A. Akbarzadeh, Liang Wang, P. Mordohai, J.-M. Frahm, Ruigang Yang, D. Nister, and M. Pollefeys. Real-time visibility-based fusion of depth maps. In *International Conference on Computer Vision*, pages 1–8, 2007.

[86] G. Metta, L. Craighero, L. Fadiga, A. Ijspeert, K. Rosander, G. Sandini, D. Vernon, and C. von Hofsten. A roadmap for the development of cognitive capabilities in humanoid robots. Deliverable D2.1, European Commission FP6 Projection IST-004370 RobotCub, 2009.

[87] G. Minglun and Y. Ruigang. Image-gradient-guided real-time stereo on graphics hardware. In *Proceedings of the Fifth International Conference on 3-D Digital Imaging and Modeling*, pages 548–555, 2005.

[88] N. J. Mitra, N. Gelfand, H. Pottmann, and L. Guibas. Registration of point cloud data from a geometric optimization perspective. In *Eurographics Symposium on Geometry Processing*, pages 23–32, 2004.

[89] A. Mittal, L. Zhao, and L. S. Davis. Human body pose estimation using silhouette shape analysis. In *Proceedings of the IEEE Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 263–270, 2003.

[90] Q. Mühlbauer, K. Kühnlenz, and M. Buss. Fusing laser and vision data with a genetic icp algorithm. In *Proceedings of the International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 2008.

[91] Q. Mühlbauer, K. Kühnlenz, and M. Buss. A model-based algorithm to estimate body poses using stereo vision. In *Proceedings of the International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 285–290, 2008.

[92] Q. Mühlbauer, S. Sosnowski, T. Xu, T. Zhang, K. Kühnlenz, and M. Buss. Navigation through urban environments by visual perception and interaction. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 1907–1913, 2009.

[93] Q. Mühlbauer, T. Xu, A. Bauer, K. Klasing, G. Lidoris, F. Rohrmüller, S. Sosnowski, K. Kühnlenz, D. Wollherr, and M. Buss. Navigation by natural human-robot interaction - wenn roboter nach dem weg fragen. *at - Automatisierungstechnik*, 58(1):639–646, 2010.

[94] L. Mun Wai and N. Ram. Body part detection for human pose estimation and tracking. In *Proceedings of the Workshop on Motion and Video Computing (WMVC)*, pages 23–23, 2007.

[95] S. Negahdaripour and B. Horn. Passive navigation. *Computer Vision, Graphics, and Image Processing*, 21:3–20, 1983.

[96] J. Neira, J. Tardos, J. Horn, and G. Schmidt. Fusing range and intensity images for mobile robot localization. *Transactions on Robotics and Automation*, 15(1):76–84, 1999.

[97] P. Newman, D. Cole, and K. Ho. Outdoor slam using visual appearance and laser ranging. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 1180–1187, 2006.

[98] D. Nister. *Automatic dense reconstruction from uncalibrated video sequences.* PhD thesis, Royal Institute of Technology KTH, Stockholm, Sweden, 2001.

[99] D. Nister. Automatic passive recovery of 3d from images and video. In *International Symposium on 3D Data Processing Visualization and Transmission*, pages 438–445, 2004.

[100] I. Nourbakhsh, C. Kunz, and T. Willeke. The mobot museum robot installations: a five year experiment. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 3636–3641, 2003.

[101] A. Nüchter and J. Hertzberg. Towards semantic maps for mobile robots. *Robotics and Autonomous Systems*, 56(11):915–926, 2008.

[102] A. Nüchter, H. Surmann, K. Lingemann, and J. Hertzberg. Semantic scene analysis of scanned 3D indoor environments. In *Proceedings of the Eighth International Fall Workshop on Vision, Modeling, and Visualization*, pages 215–222, 2003.

[103] A. Nüchter, O. Wulf, K. Lingemann, J. Hertzberg, B. Wagner, and H. Surmann. 3d mapping with semantic knowledge. In *RoboCup International Symposium*, pages 335–346, 2005.

[104] A. S. Ogale and Y. Aloimonos. A roadmap to the integration of early visual modules. *International Journal of Computer Vision*, 72(1):9–25, 2007.

[105] V. Parameswaran and R. Chellappa. View independent human body pose estimation from a single perspective image. In *Proceedings of the Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16–22, 2004.

[106] M. Park, M. Choi, and S. Shin. Human motion reconstruction from inter-frame feature correspondences of a single video stream using a motion library. In *Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation (SCA)*, pages 113–120, 2002.

[107] S. Pellegrini and L. Iochhi. Human posture tracking and classification through stereo vision and 3d model matching. *Journal on Video and Image Processing*, 2007.

[108] M. Pollefeys, L. Van Gool, M. Vergauwen, F. Verbiest, K. Cornelis, J. Tops, and R. Koch. Visual modeling with a hand-held camera. *International Journal of Computer Vision*, 59(3):207–232, 2004.

[109] Jean Ponce, Svetlana Lazebnik, Fredrick Rothganger, and Cordelia Schmid. Toward true 3d object recognition. In *In Reconnaisance de Formes et Intelligence Artificelle*, 2004.

[110] A. Pronobis, P. Jensfelt, K. Sjöö, H. Zender, G. Kruijff, O. Mozos, and W. Burgard. Semantic modelling of space. In *Cognitive Systems: Final Report of the CoSy Project*. Springer Verlag, 2009.

[111] A. Pronobis, O. Martinez Mozos, B. Caputo, and P. Jensfelt. Multi-modal semantic place classification. *The International Journal of Robotics Research (IJRR)*, 29(2-3):298–320, 2010.

[112] A. Ranganathan and F. Dellaert. Semantic modeling of places using objects. In *Proceedings of the International Conference on Robotics: Science and Systems Conference*, 2007.

[113] C. Rasmussen. Combining laser range, color, and texture cues for autonomous road following, 2002.

[114] Signe Redfield, Michael Nechyba, John G. Harris, and Antonio A. Arroyo. Efficient object recognition using color, 2001.

[115] R. H. Richens. Preprogramming for mechanical translation. *Mechanical Translation*, 3(1), 1956. Discontinued Journal.

[116] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the icp algorithm. In *Proceedings of the Third International Conference on 3D Digital Imaging and Modeling*, pages 145–152, 2001.

[117] R. Rusu, Z. Marton, N. Blodow, M. Dolha, and M. Beetz. Towards 3d point cloud based object maps for household environments. *Robotics and Autonomous Systems Journal (Special Issue on Semantic Knowledge)*, 56(11):927–941, 2008.

[118] Y. Sagawa, M. Shimosaka, T. Mori, and T. Sato. Fast online human pose estimation via 3d voxel data. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 1034–1040, 2007.

[119] G. Sandini, G. Metta, D. Vernon, P. Dario, R. Pfeifer, C. von Hofsten, L. Fadiga, K. Dautenhahn, C. Nehaniv, J. Santos-Victor, J. Grey, A. Billard, A. Ijspeert, F. Becchi, and D. Caldwell. *Robotic open-architecture technology for cognition, understanding and behaviours (ROBOT-CUB).* IST Project Funded under European Commission 6th FWP (Sixth Framework Programme) IST-2002-2.3.2.4 Cognitive systems Project Reference: 004370, 2009.

[120] T. Sato, M. Kanbara, N. Yokoya, and H. Takemura. Dense 3-d reconstruction of an outdoor scene by hundreds-baseline stereo using a hand-held video camera. In *International Journal of Computer Vision*, pages 119–129, 2002.

[121] D. Scharstein, R. Szeliski, and R. Zabih. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. In *International Journal of Computer Vision*, 2001.

[122] T. Schenk and B. Csatho. Fusing imagery and 3d point clouds for reconstructing visible surfaces of urban scenes. *IEEE GRSS/ISPRS Joint Workshop on Remote Sensing and Data Fusion over Urban Areas*, 2007.

[123] R. Schnabel, R. Wessel, R. Wahl, and R. Klein. Shape recognition in 3d point-clouds. In *The 16-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, 2008.

[124] H. Schneiderman. *A Statistical Approach to 3D Object Detection Applied to Faces and Cars.* PhD thesis, 2000.

[125] R. Schraft, B. Graf, A. Traub, and D. John. A mobile robot platform for assistance and entertainment. In *Industrial Robot Journal*, 2001.

[126] F. Schubert, T. Spexard, M Hanheide, and S. Wachsmuth. Active vision-based localization for robots in a home-tour scenario. In *International Conference on Computer Vision Systems (ICVS)*, 2007.

[127] C. Schutz, T. Jost, and H. Hugli. Multi-feature matching algorithm for free-form 3d surface registration. In *Proceedings of the 14th International Conference on Pattern Recognition*, page 982, 1998.

[128] S. Se and P. Jasiobedzki. Stereo-vision based 3d modeling and localization for unmanned vehicles. In *Special Issue on Field Robotics and Intelligent Systems*, pages 47–58, 2008.

[129] S. Se, H. Ng, P. Jasiobedzki, and T. Moyung. Vision based modeling and localization for planetary exploration rovers. In *55th International Astronautical Congress 2004*, 2004.

[130] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Proceedings of the Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 519–528, 2006.

[131] J. S. Seng and T. J. Norrie. Sidewalk following using color histograms. *Journal of Computing Sciences in College*, 23(6):172–180, 2008.

[132] M. Shanahan. A cognitive architecture that combines internal simulation with a global workspace. *Consciousness and Cognition*, 15:433–449, 2006.

[133] J. Shi and C. Tomasi. Good features to track. In *Proceedings of the Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 593–600, 1994.

[134] M. Shiomi, T. Kanda, H. Ishiguro, and N. Hagita. Interactive humanoid robots for a science museum. In *Proceedings of the 1st ACM SIGCHI SIGART conference on Human-robot interaction.*, 2006.

[135] R. F. Simmons. Semantic networks: their computation and use for understanding english sentences. *Computer Models of Thought and Language*, 1973.

[136] S.K. Singh, D.S. Chauhan, M. Vatsa, and R. Singh. A robust skin color based face detection algorithm. In *Tamkang Journal of Science and Engineering*, pages 227–234, 2003.

[137] K. Sjöö, H. Zender, P. Jensfelt, G. Kruijff, A. Pronobis, N. Hawes, and M. Brenner. The explorer system. In Henrik I. Christensen, Aaron Sloman, Geert-Jan M. Kruijff, and Jeremy Wyatt, editors, *Cognitive Systems: Final Report of the CoSy Project*. Springer Verlag, 2009.

[138] C. Stachniss, O. Martínez Mozos, and W. Burgard. Speeding-up multi-robot exploration by considering semantic place information. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2006.

[139] Michael Suppa, Simon Kielhofer, Jörg Langwald, Franz Hacker, Klaus H. Strobl, and Gerd Hirzinger. The 3d-modeller: A multi-purpose vision platform. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 781–787. IEEE, 2007.

[140] H. Surmann, A. Nüchter, and J. Hertzberg. An autonomous mobile robot with a 3d laser range finder for 3d exploration and digitalization of indoor environments. *Journal on Robotics and Autonomous Systems*, 45:181–198, 2003.

[141] M. J. Swain and D. H. Ballard. Color indexing. *International Journal of Computer Vision*, pages 11–32, 1991.

[142] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney. Stanley: The robot that won the darpa grand challenge: Research articles. *Journal on Robotic Systems*, 23(9):661–692, 2006.

[143] T. Tian, C. Tomasi, and D. Heeger. Comparison of approaches to egomotion computation. In *Proceedings of the Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 315–320, 1996.

[144] C. Tomasi and J. Shi. Direction of heading from image deformations. In *Proceedings of the Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 422–427, 1993.

[145] I. Ulrich and I. Nourbakhsh. Appearance-based obstacle detection with monocular color vision. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 866–871, 2000.

[146] R. Unnikrishnan and M. Hebert. Fast extrinsic calibration of a laser rangefinder to a camera. Technical Report CMU-RI-TR-05-09, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, July 2005.

[147] S. Vacek, C. Schimmel, and R. Dillmann. Road-marking analysis for autonomous vehicle guidance. In *Proceedings of the 3rd European Conference on Mobile Robots*, 2007.

[148] K. Valavanis. *Advances in Unmanned Aerial Vehicles: State of the Art and the Road to Autonomy*. Springer Publishing Company, Incorporated, 2007.

[149] S. Vasudevan, S. Gächter, and M. Berger. Cognitive maps for mobile robots - an object based approach. *Robotics and Autonomous Systems: From Sensors to Human Spatial Concepts*, 55(5):359–371, 2007.

[150] D. Vernon. Cognitive vision: The case for embodied perception. *Image Vision Computation*, 26(1):127–140, 2008.

[151] D. Vernon, G. Metta, and G. Sandini. A survey of artificial cognitive systems: Implications for the autonomous development of mental capabilities in computational agents. *IEEE Transactions on Evolutionary Computation*, 11(2):151–180, 2007.

[152] P. Viola and M. Jones. Robust real-time object detection. In *Proceedings of International Workshop on Statistical and Computational Theories of Vision*, 2001.

[153] C. Vogl. Schätzung der Eigenbewegung mittels Stereobildverarbeitung, Diplomarbeit an der Technischen Universität München, 2010.

[154] L. Wang, M. Liao, M. Gong, R. Yang, and D. Nister. High-quality real-time stereo using adaptive cost aggregation and dynamic programming. In *Proceedings of the Third International Symposium on 3D Data Processing, Visualization, and Transmission*, pages 798–805, 2006.

[155] G. Wei and G. Hirzinger. Active self calibration of hand-mounted laser range finders, 1998.

[156] G. Wen, D. Zhu, S. Xia, and Z. Wang. Total least squares fitting of point sets in m-d. In *Proceedings of the Computer Graphics International*, pages 82–86, 2005.

[157] A. Wu, D. Xu, X. Yang, and J. Zheng. Generic solution for image object recognition based on vision cognition theory. In *Fuzzy System Knowledge Discovery*, pages 1265–1275, 2005.

[158] P. Yan and K. W. Bowyer. A fast algorithm for icp-based 3d shape biometrics. *Computer Vision Image Understanding*, 107(3):195–202, 2007.

[159] H. Yang and S. Lee. Reconstructing 3d human body pose from stereo image sequences using hierarchical human body model learning. In *Proceedings of the 18th International Conference on Pattern Recognition (ICPR)*, pages 1004–1007, 2006.

[160] C. Yiu Ip and R. Gupta. Retrieving matching cad models by using partial 3d point clouds. 2008.

[161] A. Zatari and G. Dodds. Practical stereo vision and multi-laser scanning in object face detection and orientation determination. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 746–751, 1997.

[162] Q. Zhang and R. Pless. Extrinsic calibration of a camera and laser range finder (improves camera calibration). In *International Conference on Intelligent Robots and Systems (IROS)*, pages 2301–2306, 2004.

[163] W. Zhang, H. Deng, T. G. Dietterich, and E. N. Mortensen. A hierarchical object recognition system based on multi-scale principal curvature regions. In *Proceedings of the 18th International Conference on Pattern Recognition*, pages 778–782, 2006.