Robust Computer Vision for domestic robot applications

Dissertation zur Erlangung des Grades
eines Doktors der Naturwissenschaften (Dr. rer. nat.)
am Fachbereich Mathematik und Informatik
der Freien Universität Berlin

von

Sven Olufs

Berlin
Spring 2014

Erstgutachter: Prof. Dr. Raúl Rojas

Zweitgutachter: Prof. Dr. Davide Scaramuzza

Mündlichen Prüfung: 26. September 2014

I declare that the work in this dissertation was carried out in accordance with the requirements of the *Promotionsordnung* of the Department of Mathematics and Computer Science of Free University Berlin and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, this work is my own work. Work done in collaboration with, or with the assistance of others, is indicated as such. I have identified all material in this dissertation which is not my own work through appropriate referencing and acknowledgement. Where I have quoted from the work of others, I have included the source in the bibliography. Any views expressed in the dissertation are those of the author.

.....	.....
Sven Olufs	Date

Abstract

In this thesis we show novel techniques for the robust estimation and segmentation of indoor room structure using common 2.5d sensors, namely AIT Stereo Vision and Microsoft's Kinect. The underlying concept of this work is the so-called Manhattan world assumption i.e. the frequently observed dominance of three mutually orthogonal vanishing directions in man-made environments. Our work emphasizes on processing speed and robustness over high-quality segmentation. Many indoor environments can be considered Manhattan-like if the furniture is aligned to the walls and the room is rectangular within limits.

Our methods works in three steps: First we estimate the Manhattan world, extract features and fuse them together in a segmentation. The estimation uses three different techniques i.e. 2D vision using vanishing point detection, 3D vision using minimum entropy in histograms and normal vector MSAC estimation. All methods work efficiently and independently from each other and are robust to noise and occlusion. The feature extraction is based on the used estimators and uses geometric constrained line and plane detection. Lines are extracted using histograms and gabor filters while planes are extracted using mean shift clustering and connected component RANSAC estimators. All estimates are fused using a traditional particle filter for a coherent sensor data representation. In a last step we apply multi-label graph segmentation and extract the room structure.

We also present applications like geometric constrained visual odometry and mapping. We show that our method is robust and accurate in realistic environments using our own created database. This work can be applied for indoor robot navigation, object recognition and holistic scene understanding. Our approach is not limited to AIT Stereo Vision and Microsoft's Kinect and can be used with any 2.5d sensor like for example in Google's Project Tango.

... for Ping Ping

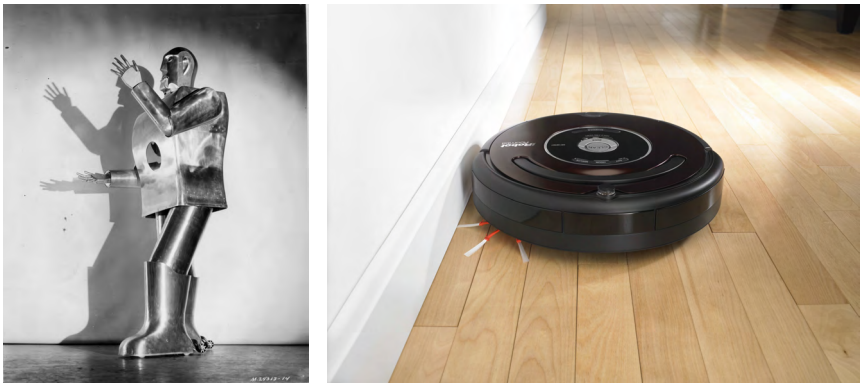
Contents

List of Figures

List of Tables

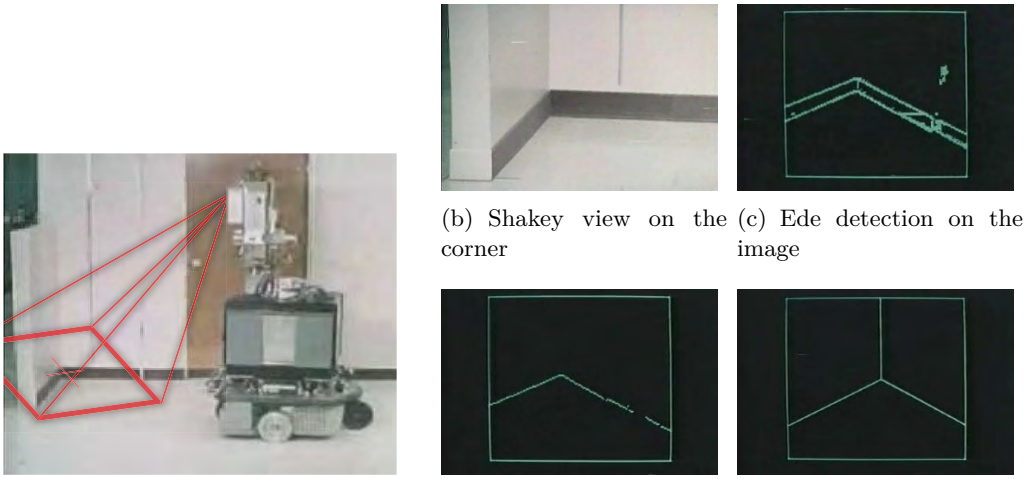
1 Introduction

Since the dawn of modern robotics (e.g. 1966) it has been predicted that robots will become part of our daily life within the next 50 years. While service robotics has become more and more common in the industry domain, they are still rare in the home robotics domain. Almost 50 years later it turned out that this prediction was partially correct. For instance, there are robots that do vacuum cleaning like iRobot's "Roomba" (see fig. 1.1(b)). While this kind of robot (and its clones) has become quite affordable for the mass market [1], they are still not found in every household. Actually some researchers believe that the "a robot in every house" prediction will probably not be true even in 100 years (Spring 2014). One reason for this partial distribution within homes may be the limited nature of the robot itself. It is only capable to clean a flat ground, but the majority of the people expect the robot to clean everywhere on top of cupboards etc.) in the household [2, 3] like a maid. The same holds true for the advanced generation of Roomba-like robots, like Samsung's "Navibot" with more sophisticated sensors which enables path planning for cleaning. Ray et al [2] notice that it is not necessary for a robot to be a humanoid to be accepted, but it might



(a) "Elektro" (1939) (b) "Rumba" (2010)

Figure 1.1: Examples of robots in households for vacuum cleaning



(a) Shakey looks at corner

(b) Shakey view on the corner

(c) Ede detection on the image

(d) Plane detection via grouping

(e) Projected estimated pose using the map

Figure 1.2: Example of Shakeys Vision system to estimate his position using walls. The red solid rectangle (1.2(a)) shows the field of view of the robot. Edges are grouped into line segments and the ground plane is estimated. A hough transformation is executed to estimate the position of the robot using a priori CADbased map. All pictures are taken from "Shakeys" original video presentation [4] from 1960

be useful since our households are made for humans. While humanoid robots become more and more advanced they are still a subject for research and are not yet suitable for a mass market of not sophisticated users.

One early example[1] of a humanoid robot at home was "Elektro" (see fig. 1.1(a)) by Westinghouse 1939. Westinghouse was known for its variety of electrical appliances for households. While the robot had a humanoid design, it did not lift up its legs for walking. Instead it used little wheels in the sole of foot. "Elektro's" capabilities were very limited, because of its is pure rigid mechanical construction. It was not more capable of "tricks" than other mechanical machines from the 18th century and before. He was demonstrated with a Westinghouse vacuum cleaner for household applications, but was remote controlled with commands like "forward" or "backwards" from a human operator and not with a behavior . Please note that the selling point of "Elektro" was that *he is the first robot that smokes* and that he was not advertised[2] as a maid-like robot. One reason why we today have robots like "Rumba" and not "Elektro" for vacuum cleaning is probably the simple rule in engineering *design follows function.*

[1] See here for more examples: http://einestages.spiegel.de/static/topicalbumgallery/3737/_ich_habe_ein_ausgezeichnetes_gehirn.html

[2] Only one robot was ever sold to a museum in Texas U.S.A.

The first robot with a behavior in the "Brooksian" [5] sense is probably "Shakey" (see fig. 1.2(b) on page 2 [4], 1966–1972). Please note that Shakey used a pure planning-based behavior approach and not a reactive approach like Brooks [5] (1991). Developed at the Artificial Intelligence Center of Stanford Research Institute (today called SRI International), it was one of the first robot on wheels using differential drive. The researchers considered the construction of a humanoid robot to be too difficult for the control loop, so the wheeled robot concept was born that is still used in today's e.g. Mobile robot's "Pioneer" robot product line or even the "PR-2" from Willow garage. "Shakey" was able to move to certain a priori known places while avoiding obstacles and pushing objects on a planar ground. A control program written in a LISPvariant was used to program the robot and let "Shakey" get to certain places (with obstacle avoidance) or pushing objects. The researchers assumed that the robot always knows its position and the objects using two 1D range finders and a pan-tilt video camera to detect walls and objects on the ground (see fig. 1.2(b) & 1.2). While "Shakey" ([4], 1966–1972) was equipped with relatively simple sensors from the today's (Spring 2014) point of view, one finding is still true [6–8]: *robust perception is the key* to all kind of e.g. low level control to high level behavior of the robot. Some researchers argue [9, 10] that today's modern sensors like Microsoft's Kinect are more precise sensors than from the one used in the late 60s; therefore they enable more sophisticated control and behavior.

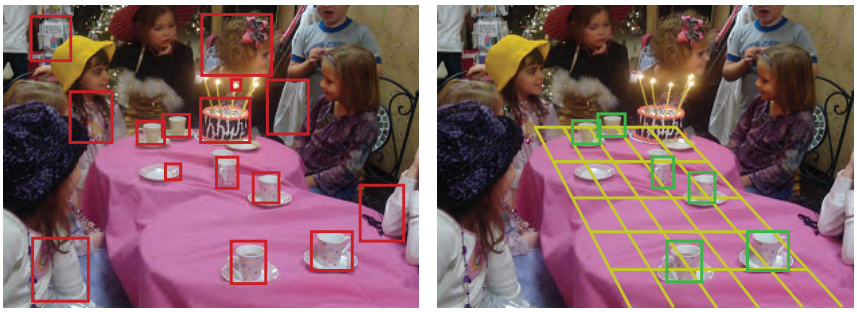
Recent demographic developments in Europe, South Korea and Japan have shown that there is a need for sophisticated control and behavior robots in everybody's home due to the *elderly society* phenomenon. Such robots can be used for the remote surveillance of elderly in the case of an emergency or to help them in their daily life. For instance, the MOVEMENT (2004-2007) [11] EU Project used a robot to literally move either a disabled person (sitting on a special chair) or objects (e.g. prepared and marked tables) by "lifting them up". The intension was to let the robot be a part of the environment like furniture, but to still be perceived as an artificial object from the point of view from the end users. One of the reasons was the pure technology design and the lack of natural interaction with the robot. The robot was controlled using a tabled PC, while commands were executed using a button like a normal Microsoft Windows PC. Another concept was used in the HOBBIT (2011-2014) [12] project introducing *mutual care* of end user and robot. The hypothesis was that the human can develop a social binding to the robot and therefore care for it like a pet. For instance the robot

looks after the human, while the human has to charge the robot. To enable such a relationship it must be able to understand the human concepts like ”rooms” and ”usage of objects” for behavior and perception. One key issue is that the **perception of man-made structure** is not able to be perceived by a robot rather than perceiving it as a set of loose planes and point clouds (from the point of view of a 3D sensor).

Recent demographic developments in Europe, South Korea and Japan have shown that there is a need for sophisticated control and behavior robots with in everybody's home due to the *elderly society* phenomenon. Such robots can be used for the remote surveillance of elderly in the case of an emergency or to help them in their daily life. For instance, the MOVEMENT (2004-2007) [11] EU Project used a robot to literally move either a disabled person (sitting on a special chair) or objects (e.g. prepared and marked tables) by ”lifting them up”. The intension was to let the robot be a part of the environment like furniture, but to still be perceived as an artificial object from the point of view from the end users. One of the reasons was the pure technology design and the lack of natural interaction with the robot. The robot was controlled using a tabled PC, while commands were executed using a button like a normal Microsoft Windows PC. Another concept was used in the HOBBIT (2011-2014) [12] project introducing *mutual care* of end user and robot. The hypothesis was that the human can develop a social binding to the robot and therefore care for it like a pet. For instance the robot looks after the human, while the human has to charge the robot. To enable such a relationship it must be able to understand the human concepts like ”rooms” and ”usage of objects” for behavior and perception. One key issue is that the **perception of man-made structure** is not able to be perceived by a robot rather than perceiving it as a set of loose planes and point clouds (from the point of view of a 3D sensor).

In order to ”bring a robot in everyone's house” [13] it must meet several requirements; for instance it must be helpful, reliable, cost efficient and safe. Some researchers believe [2, 3, 12] hat most end-users (not military or industrial) would accept a ”95%-safe” robot rather than a ”100%-safe” one if its significantly (e.g. $5-50\times$) cheaper assuming they are both equally useful. Very often sensors are a big part of the cost for a robot besides the costs of motors, on-board computers without considering costs for general development of the robot and its maintenance. For instance the SICK Laser Range Scanner LSM-200 is very often used as a 2D laser scanner for robotics. It was probably one of the most used sensors in the 90s for robots, but at a cost of approx.

$5000 (originally $8000) it is a relatively expansive sensor compared to other sensors like stereo vision. Today many cheaper laser sensors are available like the "Hokuyo URG-04LX-UG01" series with approx. $1000 per unit, but they are limited in the aspects of readability of distance measurement and range. Both sensors can be used for domestic robotics as the RoboCup@Home[3] competitions have shown.

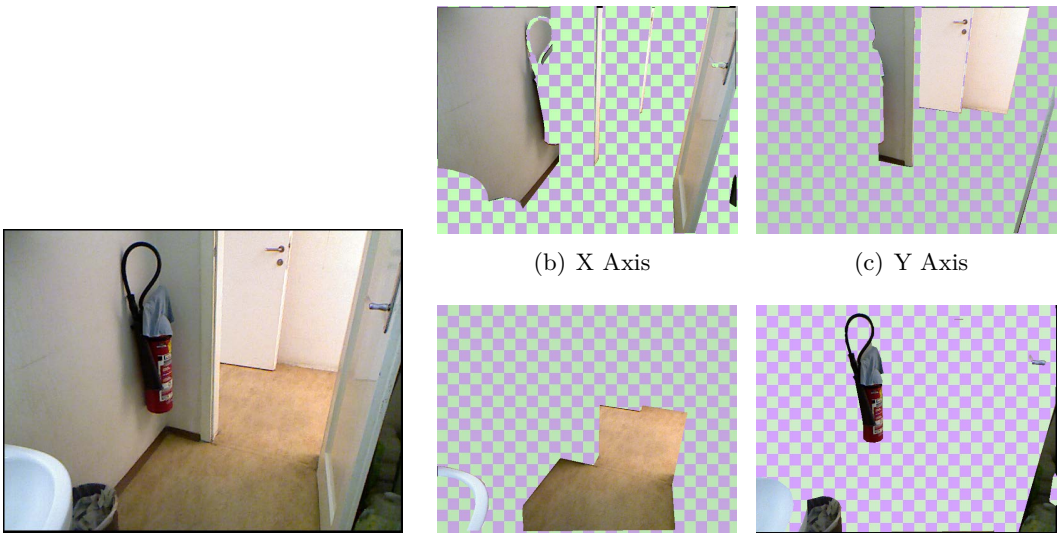


(a) Isolated mug detector (b) Holistic scene understanding

Figure 1.3: Sample for Holistic Scene understanding. The plain object detector is combined with geometry reasoning for global scene reasoning. Images taken from Bao et al. [14]

Within the home robotics domain we face the problem of having a relatively small area (e.g. $100m^2$ of the owners flat), clutter and visually weak structured environments. From the robotics point of view, one of the key problems is to estimate semantics from the visually weak structured environments. The semantics of the room structure is required for efficient user interaction [12], mapping [15, 16] and navigation [17] and object recognition [14]. Semantic information such as wall, ground, table surface, or door assists in all these tasks. We start from the observation that the structure of many rooms looks the same, e.g. they have a rectangular shape, due to the limited sensing capabilities of today's robotics. To cope with these environments, the use of 2.5d sensors has become quite popular in the last decade. For instance, the use of tilting 3D laser scanners, stereo vision or the Swissranger SR-3000 have been used, just to name a few. With the recent release of Microsoft's Kinect structured light sensor, the popularity of 2.5d sensors gained a boost. In a nutshell, 2.5d is simply a 2D image with an optimal depth value per pixel. With the Microsoft's Kinect, 2.5d is also called RGB–D, with the "-D" for depth. In spring 2014, Google announced the ŤGoogle TangoŤ project which is actually a smart-phone with an built-in Kinect-

[3] http://www.robocupathome.org, http://www.robocup.org



(a) RGB Image (b) X Axis (c) Y Axis (d) Z Axis (e) non Axis

Figure 1.4: Sample for structure aligned to Manhattan structure

like sensor setup. Google's idea is to bring in-door understand on a phone which is similar to out work e.g. for games or an interactive furniture catalog. The exact capabilities or goals of the project were not known at the time of submission of this thesis.

The *aim of this thesis* is to create a perception system that **detects indoor room structure with cost efficient sensors** to enable high level behavior on todays (Spring 2014) robots. The idea is to segment the room into semantic parts to reduce the amount of data for object recognition, behavior and to localize the robot. Within this work we exploit a certain property of **man-made structure**: the so called **Manhattan system geometry** [18–21]. It refers to the frequently observed dominance of three mutually orthogonal vanishing directions. See figure 1.4 and chapter 2.1 on page 13 for details. In this thesis we use two popular cost-efficient 2.5d sensors for indoor room perception, namely Microsoft's Kinect and AITs Stereo Camera (see chapter 2). Figure 1.3 gives an example as to how a simple 2D mug detector (1.3(a)) can significantly improve when geometry knowledge is incorporated. A 3D mug detector was not used due to the limitation[4] of depth perception of small objects. In fig. 1.3(b) all mugs are rejected that do not fit to the plane model (shown as grid) i.e. regarding scale and position on the plane. This kind of strategy is often referred to as *Holistic scene understanding* e.g. Bao's et al. [14] approach. So far techniques like holistic scene

[4]the resolution of a 2D image can be in a higher resolution than the depth data with some sensors

understanding use very often pre-labeled geometry data or simple features like planes. Within this work we present a system that enables a robust recognition of geometry within a home. A sample is shown in figure 1.4(e): The fire extinguisher is not aligned to any Manhattan structure since it is a round object. In combination with a fire extinguisher detector we can reject all false positives since the object must be aligned with either the X Axis or Y Axis in a certain height[5] and does not belong to any XYZ Axis.

1.1 Concept

The idea of this thesis is simple: combine the benefits of 2D and 3D sensor processing and fuse the results into a unified framework. The overall assumption is that both 2D and 3D data is complementary to each other. For instance, 2D can "perceive" where "3D" can not and vice versa: For instance, the detection of planes is easier with 3D data than 2D and vice versa with line detection. Another aspect is to increase the certainty when both 2D and 3D detect something similar within the same area: For instance, detecting homogeneous areas using texture/color features with 2D and normal vectors of point clouds with 3D vision.

As shown in figure 1.5, both 2D and 3D data are processed independently of each other. First, the Manhattan system(s) i.e. one global and many optional local, are estimated. 3D vision uses two techniques to estimate the system: one using normal vectors [22] and one using minimum entropy in histograms [19]. The 2D vision uses traditional vision vanishing point estimation [23] i.e. parallel lines that seem to meet in one point. After the estimate is obtained, a refinement based on local histogram is applied. All (three) independently estimated systems are fused [24] using probabilistic tracking methods and the robot's odometry. After that, a constrained visual odometry [24] is applied to estimate the robot's motion within the system. An optional step is the use of a geometric constrained spatial-temporal map using constrained ICP techniques [22] and refined by using matching techniques [25].
The fused estimates are projected back to the 2D and 3D processing. For 3D vision, Manhattan oriented planes [26] are extracted using clustering and RANSAC while the 2D processing extract exact lines segments [27] based on the estimated Manhattan system. An over-segmentation is applied on both 2D, 3D and line features [28] to

[5] according to the EU safety "EN 3" regulation

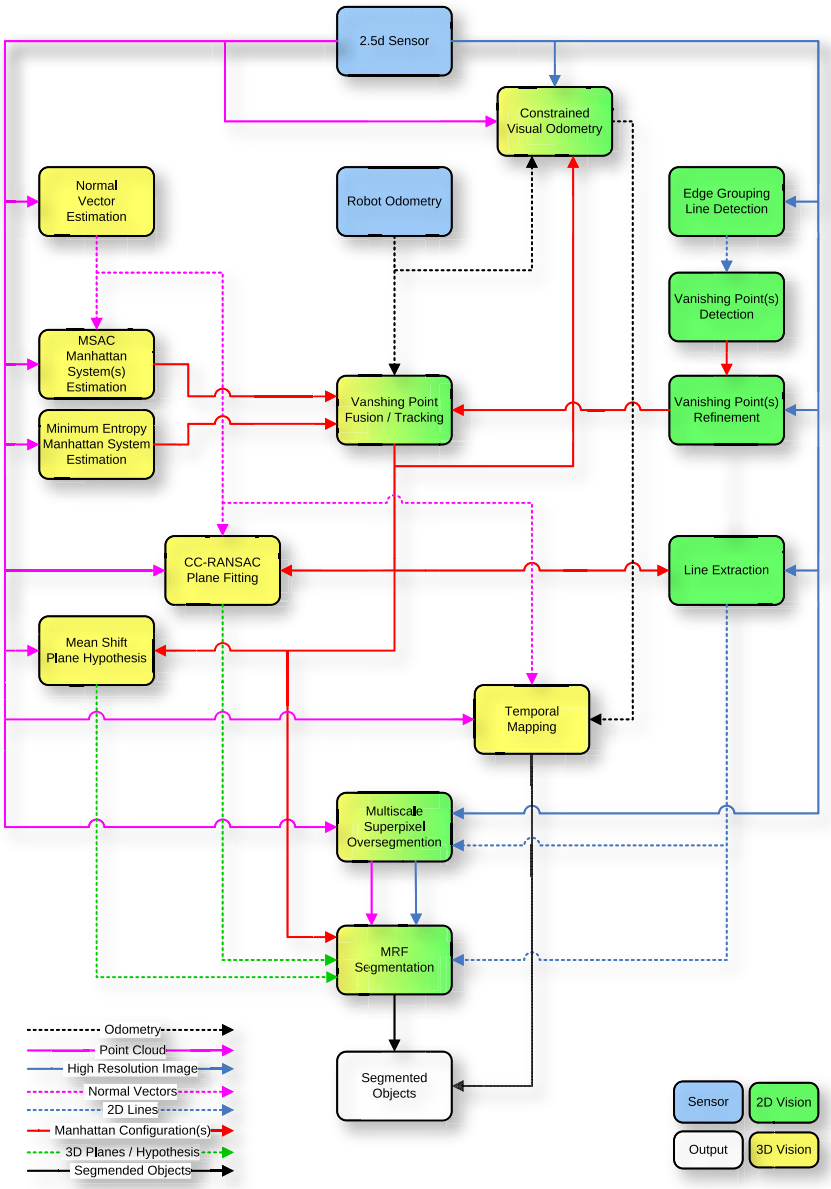


Figure 1.5: Simplified concept as flow diagram: The 2D and 3D data are independently processed first to estimate the Manhattan system geometry configurations. The estimates are fused in the tracking system using the robots odometry and projected back to the 2D and 3D processors. Features like planes and lines are extracted, which are then aligned with the Manhattan system geometry system(s). The final segmentation is completed using an over-segmentation on both 2D and 3D data and the extracted features in a multi-label graph cut

transform the data into a graph like structure with different image patches in different feature representations. The graph is segmented using a multi-label graph cut [29]. This minimizing the number of labels by fusing them into semantic parts.

This work is inspired by the pioneering work of SRI's "Shakey" robot vision (see fig. 1.2). The robot uses its pan-tilt camera to estimate its position using walls. While the sensor processing seems to be primitive on the first view; it is stunning how much this method is similar to recent developments in computer vision [18, 21, 30] and robotics [31–33]. We adapted the idea that a robot can localize itself using *just* room structure, but in a far less laboratory-like environment than "Shakey".

1.2 Organization of the Thesis

This thesis is organized into conceptual/topical chapters for better readability. For instance, we discuss the 2D and 3D perception separately, although there are many links between them. There are 3 major parts: *3D sensor processing* (left part of fig. 1.5, *2D sensor processing* (right part of figure 1.5) and *sensor fusion* (middle part of fig. 1.5). The entire thesis is organized as follows.

- **Manhattan, Hardware and Environment**: In this chapter we explain the concept of Manhattan system geometry including a brief history. The basic sensors and the used robot is introduced. We also present the test environment and our self-made evaluation database.
- **2D Sensor Processing**: The Manhattan system geometry is estimated using 2D features i.e. lines. The lines are used to estimate Vanishing Points using the Gaussian sphere technique and robust MSAC estimators. In a last step we extract lines which are oriented to the Manhattan system geometry.
- **3D Sensor Processing**: In this chapter we present a parametric and non-parametric technique to estimate Manhattan system geometry form ordered point clouds. In a second step, we extract plane features that are oriented to the Manhattan system geometry using two techniques.
- **2D/3D Fusion and Applications**: Here we show how trackers are used to fuse the 2D and 3D data into a coherent Manhattan system geometry configuration. In the last step, all data is put together using multi-label graph cuts. We also

show applications like Manhattan system geometry constrained visual odometry and constrained map building using ICP.

- **Conclusion**: Finally, we conclude the thesis in this chapter. A discussion about the scientific contributions as well as the limits of the approach is also given. It concludes with giving an outlook on possible extensions.

1.3 Contributed Publications

The following selected papers led to this PhD thesis in chronological order:

- [34] P. Einramhof, S. Olufs, M. Vincze. "Experimental evaluation of state of the art 3d-sensors for mobile robot navigation". *In 31st AAPR/OAGM Workshop*, Schloss Krumbach, Austria, 2007
- [35] M. Vincze, S. Olufs, P. Einramhof, Wildenauer. "Roboternavigation in BÃijros und Wohnungen (In German)". *Elektrotechnik und Informationstechnik (e&i)*, 1-2(1-2):25 âĂŞ 32, 2008
- [36] S. Olufs and M. Vincze. "An Efficient Area-based Observation Model for Monte-Carlo Robot Localization". *IROS*, St. Louis, U.S.A., 2009
- [37] S. Olufs and M. Vincze. "An Intuitive Inexpensive Interface for Robots using the Nintendo Wii Remote". *IROS*, St. Louis, U.S.A., 2009
- [38] K. Ambrosch, M. Humenberger, S. Olufs. "Embedded Stereo Vision". In Ahmed Nabil Belbachir, editor, *Smart Cameras*, chapter 8. Springer, London, first edition, 2010
- [13] M. Vincze, W. Wohlkinger, S. Olufs, P. Einramhof, and R. Schwarz. "Towards Bringing Robots into Homes". *ISR & ROBOTIK 2010*, Munich, Germany, 2010
- [27] R. Schwarz, S. Olufs, M. Vincze. "Merging line segments in 3D using mean shift algorithm in man-made environments". *In AAPR/OAGM Workshop*, Zwettel, Austria, 2010
- [39] S. Olufs and M. Vincze. "Robust Room-Structure estimation in Manhattan-like Environments from dense 2.5D range data". *In IEEE ICRA 2010 Workshop on Semantic Mapping and Autonomous Knowledge Acquisition*, Taipei, Taiwan, 2010
- [19] S. Olufs, M. Vincze. "Room-Structure estimation in Manhattan-like Environments from dense 2.5D range data using minimum Entropy and Histograms". *In IEEE WACV*, Hawaii, U.S.A., 2011.
- [28] S. Olufs, M. Vincze. "Robust Single View Room Structure Segmentation in Manhattan-like Environments from Stereo Vision". *In IEEE ICRA*, Shanghai, China, 2011
- [25] S. Olufs and M. Vincze. "Efficient Semantic mapping of Man-made environments using Kinect". *In IEEE ICRA Workshop on Active Semantic Perception and Object Search in the Real World (ASP-AVS- 11)*, San Francisco, U.S.A., 2011
- [26] S. Olufs, M. Vincze. "Real time Manhattan-like Structure segmentation from Kinect with constrained 1D CC-RANSAC". *In IEEE SSRR*, Kyoto, Japan, 2011
- [40] S. Olufs, P. G. PlÃűger, M. Vincze. "Probabilistic Shape Vision for Embedded Systems (Best Paper Award)". *In IEEE URAI*, Incheon, South Korea, 2011

- [22] S. Olufs, M. Vincze. "Towards efficient Semantic Real time mapping of Man-made environments using Microsoft's Kinect". *In IEEE ROBIO*, Puket, Thailand, 2011
- [29] S. Olufs, M. Vincze. "Towards robust Room Structure Segmentation in Manhattan-like Environments from dense 2.5D data (Best Paper Award)". *In IEEE ICCAS*, Seoul, Korea, 2011
- [24] S. Olufs, M. Vincze. "Visual IMU in Manhattan-like Enviroments from 2.5D data". *In AROB*, Beppu, Japan, 2012
- [41] M. Vincze, W. Wohlkinger, A. A. Buchaca, S. Olufs, P. Einramhof, K. Zhou, E. Potapova, D. Fischinger, M. Zillich. "Roboternavigation in BÃijros und Wohnungen (in german)". *Elektrotechnik und Informationstechnik (e&i)*, 1(129):42 âĂŞ 52, 2012
- [42] S. Olufs, M. Vincze. "Semantic Segmentation in Manhattan-like Environments from 2.5D data". *In IEEE AROB*, Daejeon, South Korea, 2013
- [43] M. Vincze, S. Olufs, W. Wohlkinger, P. Einramhof, R.Schwarz, K. M. Varadarajan. "A Situated Approach to Scene Understanding and Object Categorization for Domestic Robots". *Journal of Intelligent & Robotic Systems*, in press

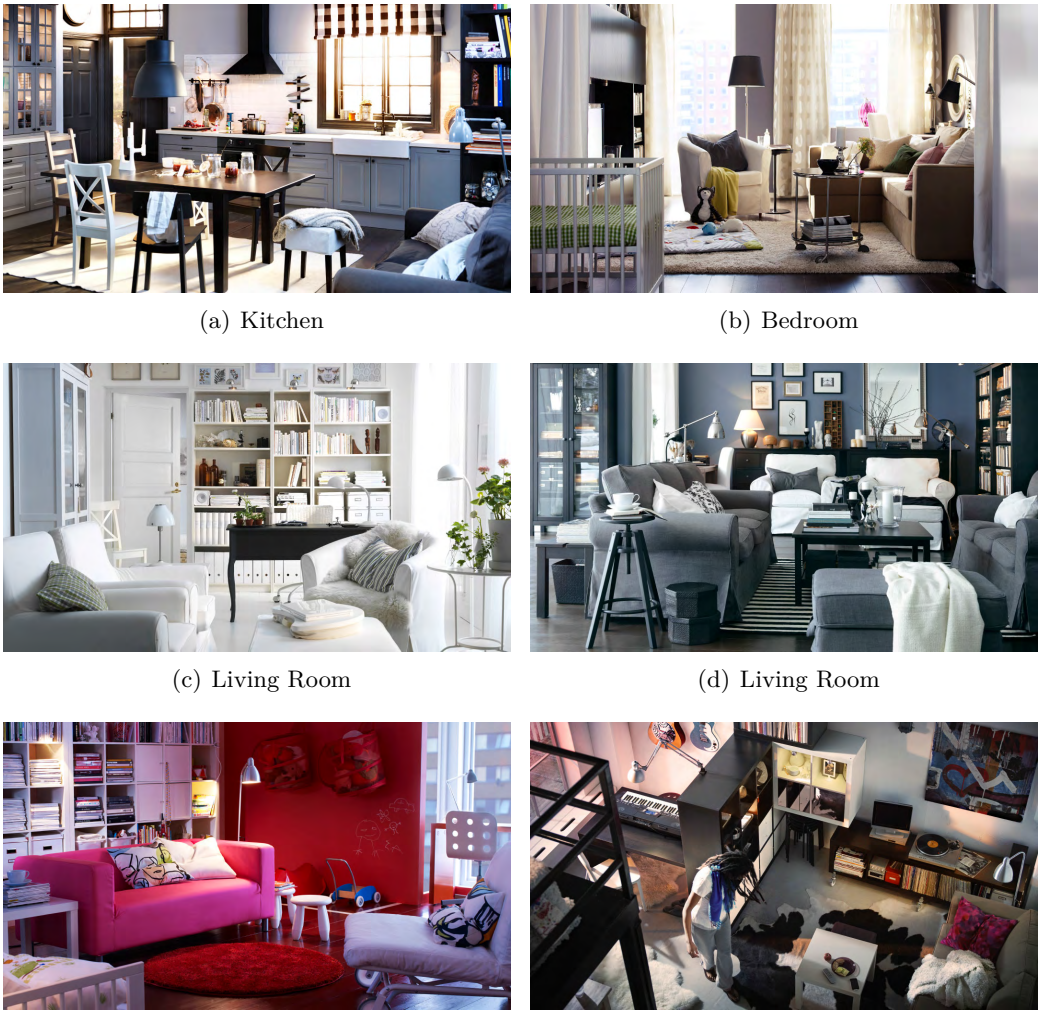
2 Manhattan, Hardware, and Environment

In this chapter, we show the technical aspects such as the sensors, robot and PC hardware applied within this thesis. We start with an overview of the *Manhattan system geometry* concept that is widely used in this thesis. The concept holds true for many domestic environments and is a key assumption in our work. We continue with the environment itself, which has been used for tests including a self-created database used for tests and benchmark. Due to the nature of our work, databases for both[1] 2D and 3D from the same image source are very rare. Next, we introduce the used sensors in this work and conclude with the introduction of robot "James". James was used for ground truth acquisition within this thesis. We provide ground truth for the Manhattan system geometry configuration per camera and frame. Please note that we do not provide ground truth for the sake of mapping and segmentation, because the quality of the output is more subjective and depends on the personal point of view and not a qualitative error that can be measured.

2.1 About Manhattan-like environments

When we consider "every day" home environments, (**man-made structure** one can notice a common pattern that holds true for many Western and Asian cultures: the basic layout of rooms is rectangular in most cases. We can find many examples throughout mankind from ancient Egypt to ancient Chinese culture. One of the possible reasons is that rectangular structure is easier to plan and build than non-rectangular structures. The probability oldest rule (more than 3500 years) [44] for building structures resulting in a rectangular shape is 風水 or *Feng Shui* (pronounced Feng Shu) from ancient china. Feng Shui states that everything in a home has to be "in harmony"

[1] From the same sensor and time stamp



(a) Kitchen (b) Bedroom

(c) Living Room (d) Living Room

(e) Kids room (f) Office

Figure 2.1: Sample home environments taken from the Austrian IKEA catalog (fall 2013). Despite that some environments are quite cluttered, most of the furniture is aligned with the walls in a rectangular layout

(also translated as "aligned with") with the four directions of north, south,west, east and heaven (=vertically aligned). The directions are not meant in the literal sense, but rather in the sense that each house must have the same five directions.

Another observation can be made about home environments. Furniture is also aligned with the major rectangular layout of the room. When we consider figure 2.1 we see that almost all objects are aligned with the walls, with some exceptions like an armchair (fig. 2.1(b) or loose toys (fig. 2.1(e)). These pictures are seen as a realistic example from the Austrian IKEA catalog edition Fall 2013. We did not use Google to search for home environment photos due to most of them showing an unrealistically clean

Figure 2.2: Example room with furniture not aligned to walls

environment, but with usually all furniture aligned to the walls. It turned out that it is quite hard[2] ($> 1\%$) to find a room where alignment does not hold true, with the exception of designer rooms or museums. Such an exception is shown in figure 2.2: Despite the fact that the furniture is not aligned to the walls, it is still aligned to each other. The "normal" alignment with walls probably has a purely practical aspect, for instance that it is easier to clean or to use. Some researchers believe that there is a link between the human perception and room layout within the Gestalt theory [45]: the human perception seems to favor structures that are aligned to each other, [46]for instance those with lines and enclosures.

A technical term from the computer vision community for this kind of rectangular aligned structures is the so-called **Manhattan-like environment** [18–21, 47]: the frequently observed dominance of three mutually orthogonal vanishing directions in man-made environments [21]. Sometimes it is also called "man-made structures" or "Manhattan world". The term "Manhattan" is named after the famous district in New York City in the U.S.A. Almost the entire district is based on a pure square grid or blocks of streets due to histrionic reasons. Almost every block itself is a collection of square-shaped houses [48] for instance the Rockefeller center or the former World Trade Center, with some exceptions like the Chrysler building. An example of a non-square layout in Manhattan is Broadway Street: it was a trail built by Native Americans that snaked through swamps and rocks. While the street literately swirls around the other square structure, it still maintains a square structure within: All houses and blocks

[2] We invite the reader to google "living room" (google image search or flickr) and see the results

align with it like with the rest of the square footprint [49].

Here we want to introduce the term that will be used in the entire thesis: the **Manhattan system**. The Manhattan system defines the layout of structure that is aligned within the three major axes of the Euclidean 3D space i.e. X, Y and Z within a Manhattan system. The system is not necessarily aligned to the observer's view, hence we denote it as the relative roll[3], pitch and yaw to the normal axis. Therefore, these angles will be referred to as **Manhattan system geometry configuration**. Here we want to emphasize that the Manhattan normal axes do not necessarily correspond to the world's normal axis since both are just a different point of view. The Manhattan system is technically nothing else than the rotation of the camera relative to the normal axis of the environment [47, 50].

In this thesis, we distinguish between two types of Manhattan systems: the **global** and **local** one. The global Manhattan system is the dominant system within a room, like the room layout itself assuming a rectangular footprint of the room. The local Manhattan system is another system with the global system that shares at least one axis of the global one. During the Manhattan system estimation process within this thesis, one global system many optional local ones are found. Other systems are discarded and ignored. In the case of Manhattan itself, the global system would be the city itself, while Broadway is a local system since all building are all vertically aligned to the ground. Another example of a local system is an armchair in figure 2.1(b): The chair has a rectangular shape and can also be considered as a Manhattan system. A more complicated example is shown in figure 2.2: Here it is debatable if the walls or the couches are the dominant structure from just one point of view. In this case, it can only be decided if both structures are tracked within the motion of the observer. Here we want to emphasize these extreme cases are relatively rare.

We can think of many cases when Manhattan systems are difficult to estimate, i.e. if not all structures are aligned orthogonally to each other. As an example we can use a scene from the historic city of Alkmaar near Amsterdam in Holland, as is shown in fig. reffig:intro:manhattan:samples:hardcase: Almost all houses have been built on sand and sink into the ground over time one by one. The left picture shows the case that a house slightly tilted to the front. Using the local/global terms we can detected the

[3]Technically normal vectors are used instead of the angles

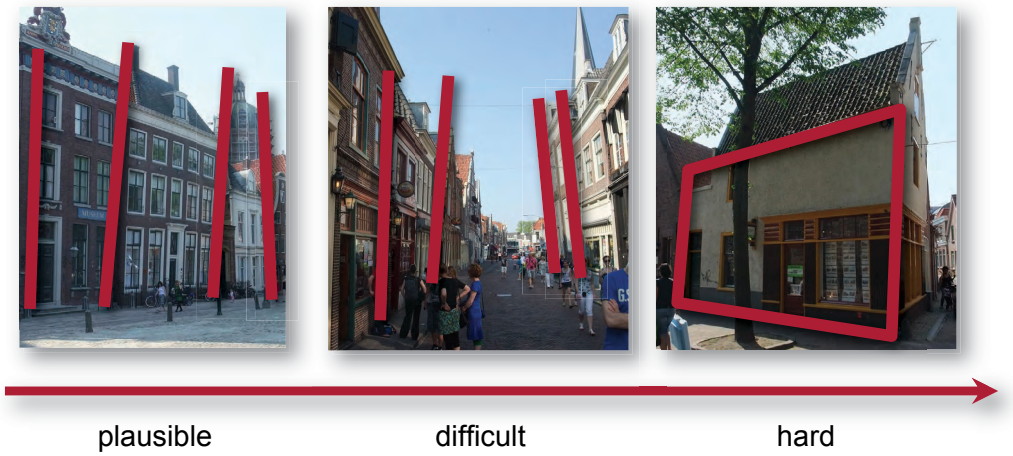
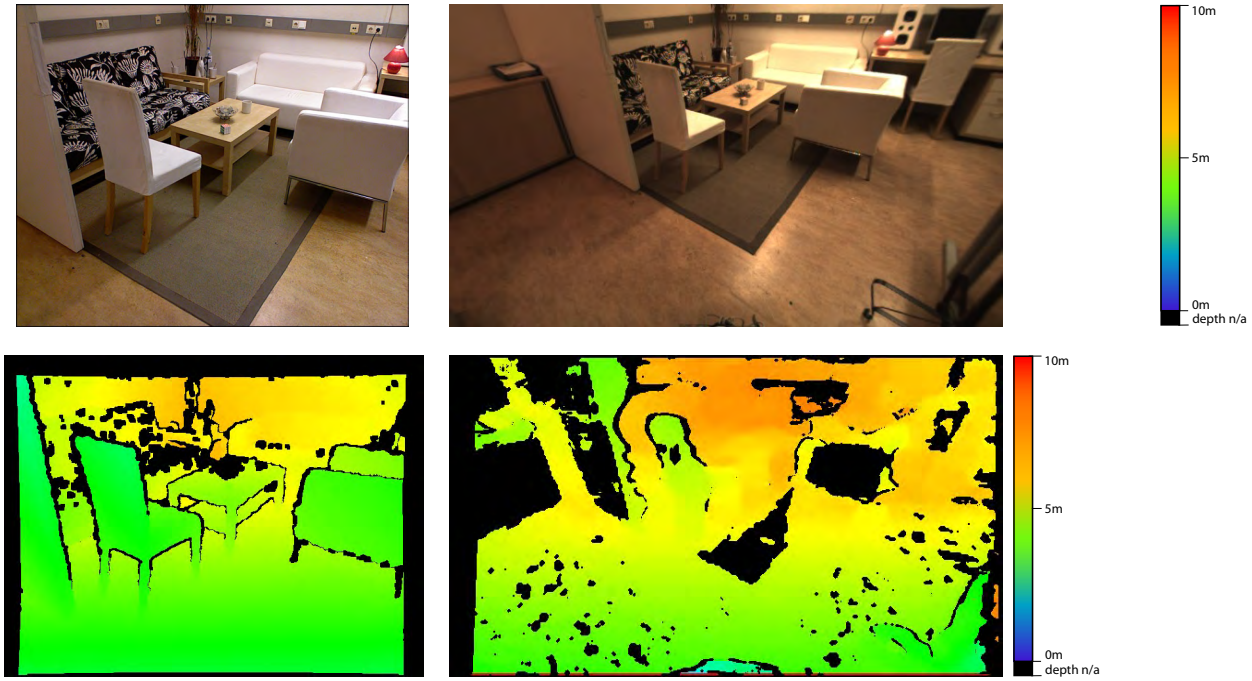


Figure 2.3: Different extreme cases of Manhattan systems

straight and tilited house. The center picture shows multiple houses (so local systems) tilted to the street. Here the proper global system can only be estimated if the ground is visible and if at least one house is not tilted. Otherwise, it is pure chance as to which one is estimated as the global one[4]. The right case is a classic example of sheared axis i.e. none of the axes are orthogonal to each other. This case can only be detected when the Manhattan system assumption is relaxed such as in the case where the angles are not a strict 90 degrees to each other.

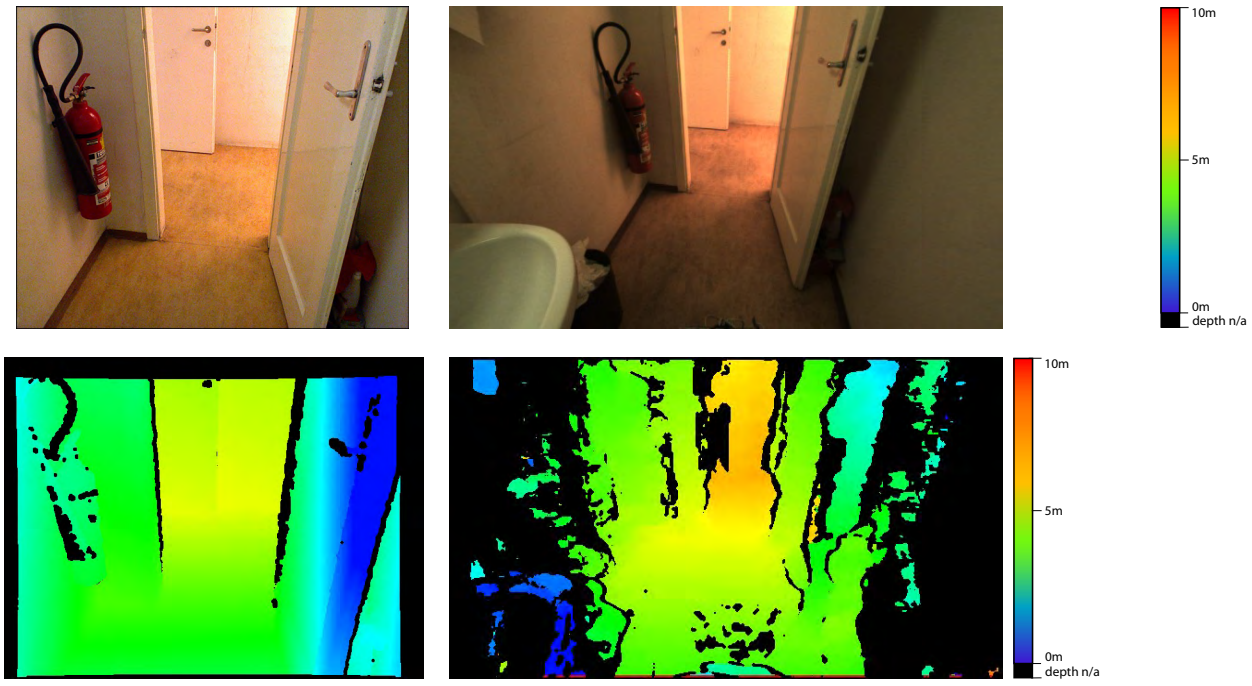
Many indoor environments can be considered as Manhattan-like since most walls of a room are aligned orthogonally to the ground, or they can be described as quasi Manhattan-like if the walls are not aligned orthogonally to each other. In many cases, furniture is also aligned Manhattan-like to its environment, e.g. a couch or cupboard can be aligned to a wall. Here we emphasize that it is not necessary for the furniture to be aligned to all three major axes; even if a table is not aligned to a wall, its table surface is usually parallel to the ground (quasi Manhattan-world).

[4]this can be solved by in-cooperating observers motion



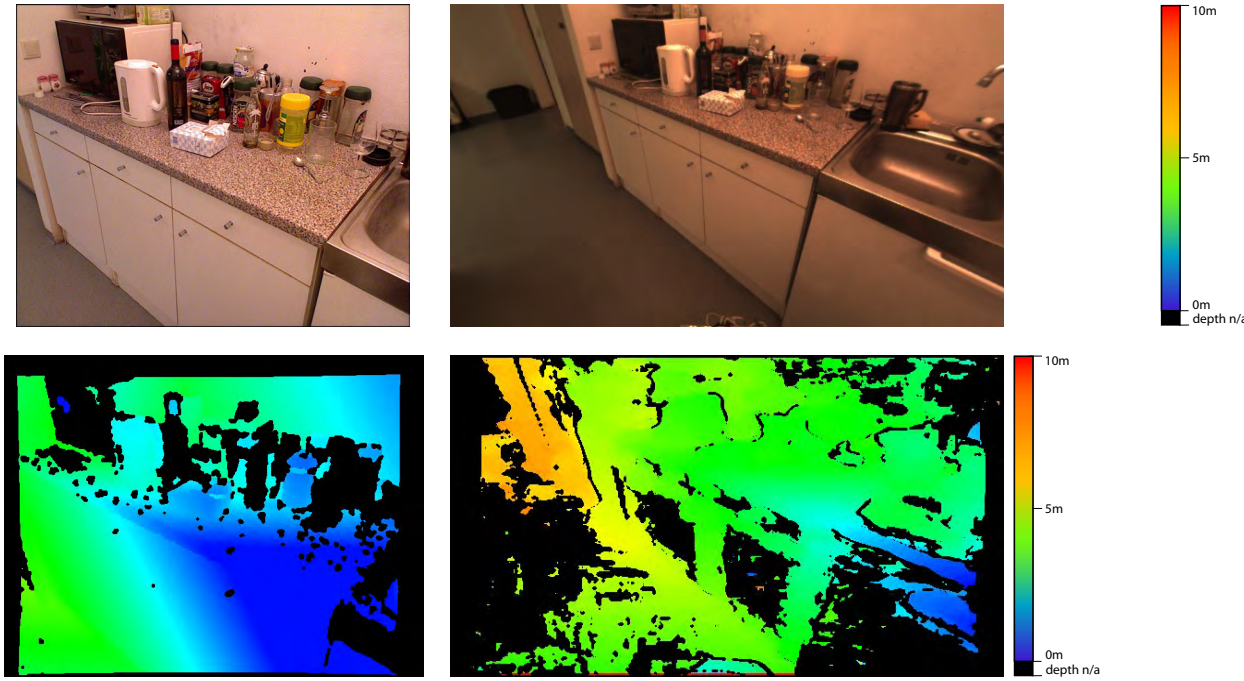
(a) Microsoft's Kinect (a) AIT Stereo Vision

Figure 2.4: Living Room, RGB on top and depth below in HSI colors



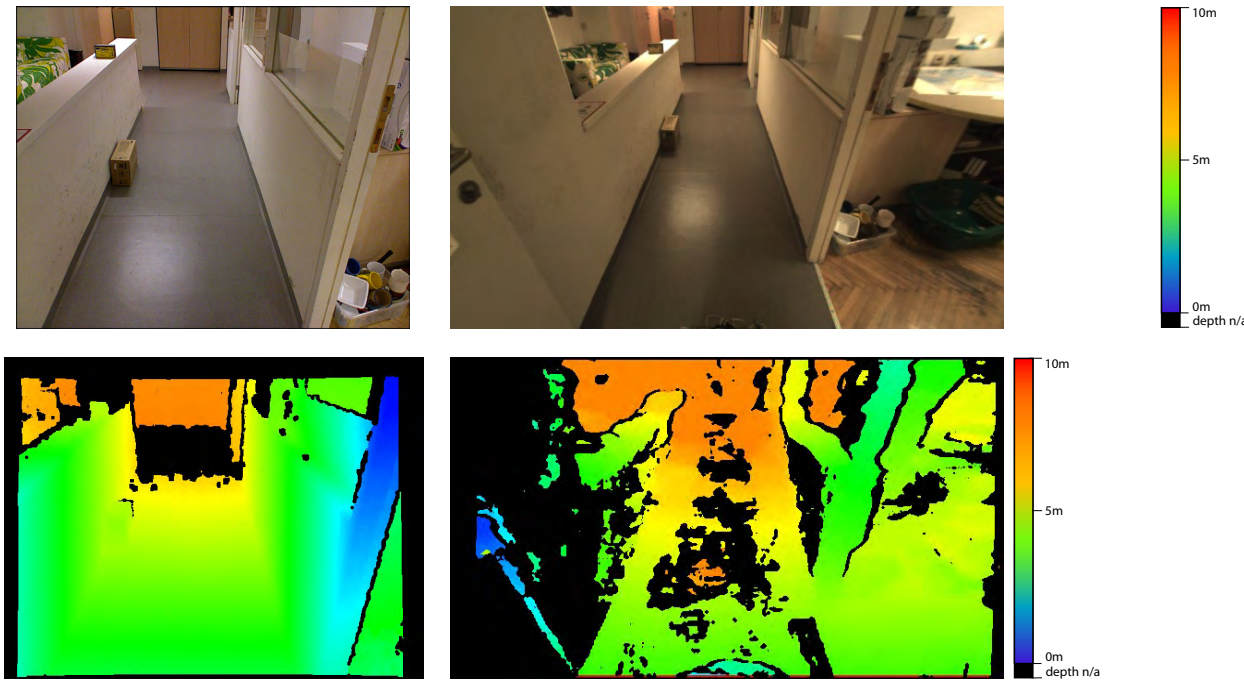
(a) Microsoft's Kinect (a) AIT Stereo Vision

Figure 2.5: Restroom, RGB on top and depth below in HSI colors



(a) Microsoft's Kinect (a) AIT Stereo Vision

Figure 2.6: Kitchen, RGB on top and depth below in HSI colors



(a) Microsoft's Kinect (a) AIT Stereo Vision

Figure 2.7: Corridor, RGB on top and depth below in HSI colors

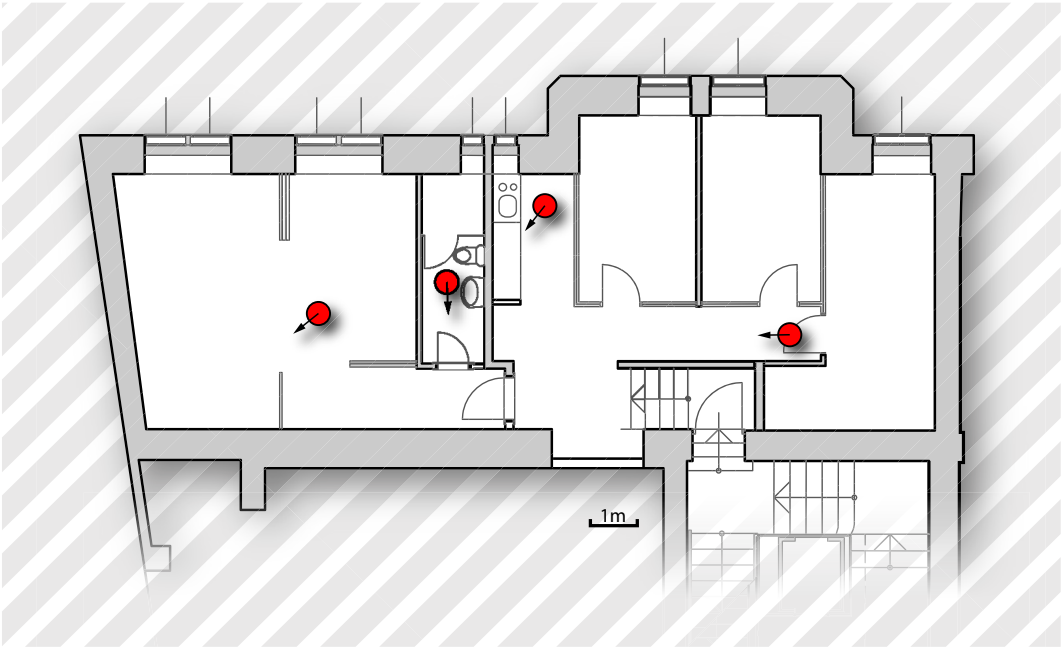


Figure 2.8: CAD Version of the Office. The Red markers depict the pose of robot for the sample pictures used within this thesis. From Left to Right: 1) Living Room (fig. 2.4), 2) Restroom (fig. 2.5), 3) Kitchen (fig. 2.6) and 4) Corridor/Lobby (fig. 2.7)

2.2 The Environment

We use a sophisticated lab environment that was decorated like a home environment, **as is** seen figure 2.8. The environment contains a living room on the left with a small and big sofa, a table and several sideboards in two heights. Next to it, on the right, we have a small restroom which is connected to the living room with a long corridor. In the center, we find a fully functional kitchen with a couch on the bottom of the map for convenience. On the right, we have three more rooms that are used as an office with an additional couch for the room on the right. The only furniture that is missing[5] from a typical home is a bed. The environment on the map has a total size of $220m^2$.

The evaluation data for our work was captured in the University of Technology in Vienna "TU Wien"(Gusshausstrasse 30, 1040 Wien, Austria). The building was made in 1880 and was originally used for the department of farming and agriculture. While the outer walls (thick walls) were brick, the inner walls were made from wood. The celling and floor were made with an early version of the metal carrier construction with multiple layers of wood on top as ground. Since the layout of the rooms have changed

[5]It was not allowed by the owner, because it can easily set on fire

within the last 130 years, almost none of the walls of our environment were straight. Another issue is fatigue of material over the years: Some parts[6] of the ground tend to sag (up to 15cm) since they are made from wood. This sag effect holds true especially for the large room on the left on the map and for the vertical corridor on the right above the staircase.

One can see from the map that not all rooms fit to the Manhattan system geometry assumption: for instance, one wall in the living room (left in the map). Another less conspicuous one is the right corridor: here the walls are not 100% parallel, because the office in the center is sightly bigger[7] than the other ones.
For this work we recorded 15 tours in the environment with a total length of $\approx 250m$ and an average traveling speed of $0.65\frac{m}{s}$. Please note that the environment was **not** cleaned or made tidy before to keep a more realistic setting (for lab standards). We noticed that boxes and batteries had been randomly placed in the environments while the kitchen was the most cluttered space in the environment. We choose four representative spots[8] in the environment that were used within this thesis, as shown as red arrows in figure 2.8: living room (fig. 2.4), restroom (fig. reffig:setup:env:Restroom), kitchen (fig. 2.6) and corridor (fig. 2.7). The living room spot shows the "hard" case from figure 2.3 on page 17 i.e. "shear". The living room is connected via a corridor to the lobby and restroom. Due to space limitations, a printer/shelf (see left in fig. 2.4) is located in this place. The restroom spot highlights the case of a narrow weak structured environment with a door while the kitchen spot is a good example of clutter. The last spot "corridor" shows a narrow but wide scene. The corridor leads to a small lobby that is both visible from the kitchen and the corridor. Please note that all evaluation was done on the complete dataset; we show always the same four places for better understanding and traceability.

We do not use an environment from a furniture store for evaluation like IKEA, because it is not suitable for our needs: In order to showcase the rooms in the exhibition, at least one wall is missing. In many cases, the missing wall is also the "biggest" one and is important for portraying a realistic setting. Another issue is the lack of doors. We also do not use real home environments for the sake of privacy.

[6] this is due to maintenance of the building and depend on the room
[7] See the non-straight wall on the bottom of the room
[8] All these spots are parts of a moving robot tour within the environment

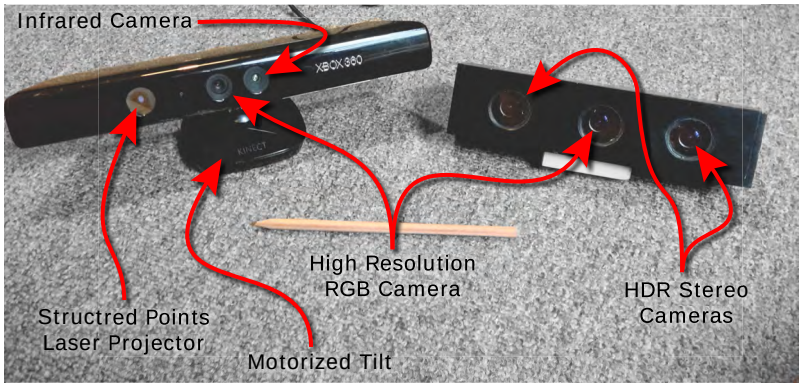
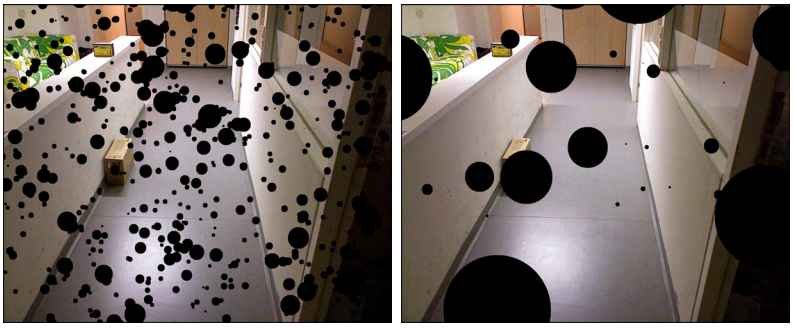


Figure 2.9: Microsoft's Kinect and AIT Stereo Vision

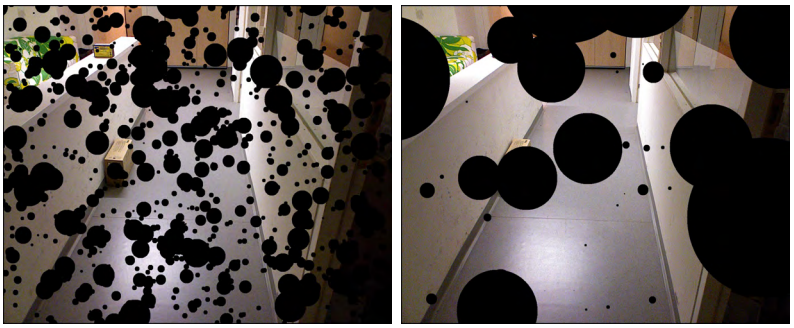
2.3 2.5d Sensors

Within this thesis we use particular sensors 2.5d i.e. Microsoft's Kinect and AIT Stereo Vision as representative sample sensors for the home robotics domain, see fig. 2.9. Both sensors deliver a 2D image plus depth per pixel, hence the name "2.5d" (.5d for depth). Both sensors can be used for the indoor robotics domain, while the Microsoft's Kinect was particularly made for this environment. First, we introduce Microsoft's Kinect followed by AIT Stereo Vision:

Microsoft's Kinect is a sensor that was originally created as an input device for Microsoft's XBox 360. The sensor detects the human using a skeleton tracker and is used "as the controller" instead of the game pad controller. The 2D image is captured with a single SXGA resolution (1280x1024 square pixel, approx. 8-15 fps) RGB camera. Depth is obtained using a structured light pattern projector in the infrared spectrum and by using infrared sensitive camera. The principle is similar to stereo vision just reverse; the pattern of the IR-projector is a-priori known ("Light Coding") and is used to estimate depth by matching them trough disparities. The estimated depth is projected back to RGB camera in VGA resolution (640x480 approx square pixel, 25-30 fps). This result is that there are "blind spots" in depth image due to the offset of the cameras. Teichman et al [51] remarks that the depth estimation accuracy differs from device to device and has up to 5% noise.
The Microsoft Kinect sensor is suitable for the indoor robotics domain for two reasons. First, the sensors are cheaper than laser scanners and offer a depth image at frame rate. The challenge with data from 2.5d data is coping with noise and uncertainty



(a) 20% Occlusion



(b) 40% Occlusion

(c) 60% Occlusion

(d) 80% Occlusion

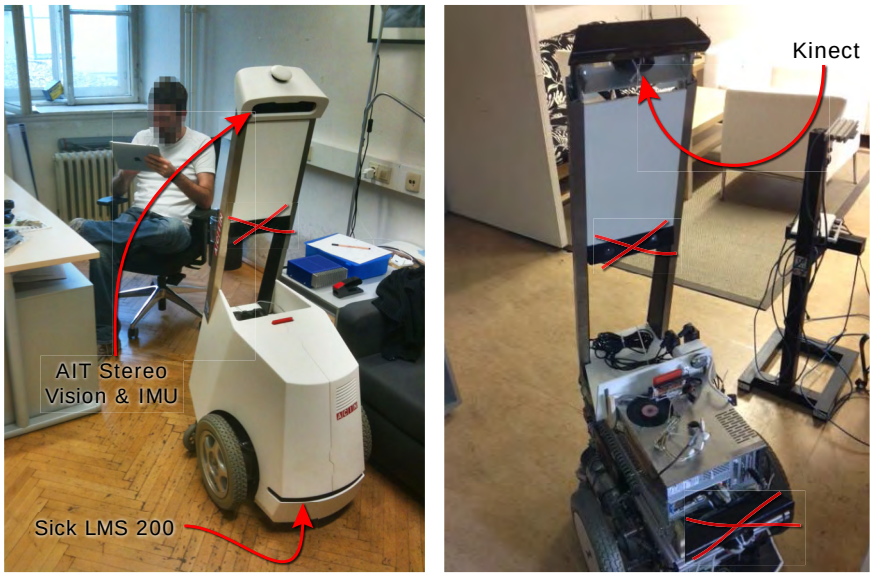
Figure 2.10: Two occlusion mask used for evaluation of 2D and 3D data. The left column shows the ”small” mask occluding non-connected parts of the image using a filled circles. The right one shows the ”big” mask that occludes connected areas

due to the nature of the sensors. For instance, the quality of depth data from the Microsoft Kinect depends on the reflection properties of the observed surface and the angle of incidence (assuming Lambert surfaces) due to the sensor use structured light dot pattern for the depth perception. Shiny surfaces are only detected within 4m with straight angle and 1.5m with 45deg angle in most cases. Another issue with the Microsoft Kinect is that the RGB camera cannot be adjusted and always sets its exposure rate and color balance for the brightest light source in the image. Since the sensor uses structured light in the infrared spectrum, the depth perception of the sensor is sensitive to sunlight. In this case, the Kinect looks at an area with bright sunlight like a sun reflection on the ground and is not be able to sense any depth data for this area.

The *AIT Stereo vision* is traditional passive stereo vision. We use in this thesis a custom version for Austrian Institute of Technology AIT with two HDR UEye PAL (720x576 square pixel, 25-30 fps) stereo cameras and a single RGB PAL Camera in the center. The stereo depth data is mapped on the center camera instead of the left or right camera as in traditional stereo systems. The issue with stereo vision is that it can only detect depth in the case of textured objects. We use a GPU implementation of the CENSUS AIT Stereo Engine [52] for dense stereo-data calculation at a resolution of 720x480, using 80 disparities and 16 subpixel interpolation. The GPU implementation of the engine enables us to work up to 120fps(see below for system specs). Experiments have shown that the IR projector of the Kinect is also slightly visible in the HDR stereo cameras in and also improve the stereo detection on untextured objects within 1m. Similar techniques are also used on Willowgarage's "PR-2" robot for the stereo system. We use this kind of sensor with a RGB camera to have comparable results with related monocular vision approaches.

Compared to Microsoft's Kinect, depth with stereo vision is far less dense, but is less affected by shiny surfaces. As a side note we like to mention that the competitor product of Microsoft's Kinect (XBox 360/ XBox One) the "Playstation Eye" from Sony uses stereo vision (Playstation 4).

Within this thesis we will apply benchmarks on both Microsoft's Kinect and AIT Stereo Vision using occlusion masks as shown in figure 2.10. The idea is to use two masks to occlude certain parts: Both 2D and 3D data is masked out and therefore not used and are identical for all images. We use two kind of masks: one that occludes small



(a) Original Design 2010

(b) Version (2012) used for experiments without IMU & AIT Stereo Vision

Figure 2.11: The Robot "James" in the original version (2.11(a)) and in the modified version for experiments without casing (2.11(b))

non-connected parts "small" and one that occlude only connected parts "big", see fig. 2.10.. Both masks are identical for each image and are generated for each amount of occlusion.

2.4 Robot James

For our experiments, we use a non-holonomic mobile robot manufactured by Bluebotics named "James", see figure 2.11 and table 2.1 for details. The intention of the robot was to be a butler. The basic platform was modified for our experiments by adding a Kinect on top of the Stereo cam with an additional xSens IMU inside the stereo camera. In the original version, the robot was only equipped with two AIT Stereo Vision and a Sick LMS 300. The Kinect and AIT Stereo Visionare were glued on a camera rig which is aimed approximately 35 degrees downward in respect to the robot's driving direction and is mounted at a height of a 130 cm over the ground. We use the servo motor inside Microsoft's Kinect to tilt the camera in the moving direction of the robot. The Kinect is tilted downward if the robot is not moving, or is moving backwards. When the robot moves, it aims at the point where the robot will be in 3 seconds using a linear velocity model.

The two pictures (see 2.11) show an early and late state, since the original picture

Table 2.1: Reference Robot used within this thesis

Manufacturer	Bluebotics
Type	Movement Prototype 2006 [11]
Drive	Differential (Non-holonomic)
Size	50x75x140cm
Weight	80kg (with batteries)
Wheels	32cm / Standard Wheelchair
Power	2x 12V/48Ah
Autonomy time	$\approx 2h$
On-board Sensors	Sick LMS 200
Additional Sensors	Microsoft's Kinect AIT Stereo Vision XSens MTi-10 IMU

of the robot during the experiments is lost. Please note that the crossed-out sensors were not used in the experiments. For the PC inside the robot, (see table 2.2) we used standard energy saving PC hardware that is commonly used in laptops. We show both robot and PC specs here in a table for convenience, since they will be referenced very often within this thesis.

The IMU and Sick laser scanner are only used for ground truth. We obtain ground truth for the Manhattan system geometry configuration per frame for both Microsoft's Kinect and AIT Stereo Vision using the following method. First we use the laser scanner to obtain the *yaw* angle from the Manhattan system geometry. This is done by using offline MCL localization with the Sick laser scanner and a SLAM map of the environment. The SLAM map was created using the *gmapping* package from ROS (`www.ros.org`) with an additional Laser scanner on the back of the robot, see figure 2.12. The SLAM map was also oriented to obtain proper[9] yaw angles per cell. The offline MCL also uses a ROS package *amcl* which was modified to be used iteratively, similar to *simulated annealing [53] technique*: with each iteration the error of the previous iteration is minimized in respect to the robot motion. Overall, we obtained an overall accuracy below 0.1 degree.

The two reaming angles *pitch* and *roll* were obtained from the 2D images of each sensor individually using *bundle adjustment* [50]. The idea is that a (in our case ordered) sequence of picture with known/guessed viewpoints is optimized by refining the viewpoints in respect to an overall consistency of all viewpoints. In many cases,

[9] We use the yaw angle from the dominant structure in the environment

Table 2.2: Reference PC Hardware on the robot used for all code within this thesis

Processor	Intel i7-3770
CPU Mhz	1.8Ghz Idle - 3.2 Ghz load
Number of CPU Cores	4 (8 hyper threading)
RAM	16 GB (PAE)
Harddrive	Samsung 120Gb SSD
GPU	NVidia Geforce GTX 280 1GB
OS	Ubuntu 12.10 (32bit)
GCC Compiler	gcc (Ubuntu/Linaro 4.7.2-2ubuntu1) 4.7.2
OpenMP	yes

features like SIFT [54] are used as the consistency criterion. Since bundle adjustment relies on proper initializations like ICP, we use the robots pose (X, Z and yaw) and use the IMU data for roll and pitch. For Y we use the a-priori height of the sensors. We use Christopher Zach's[10] code for *Simple Sparse Bundle Adjustment* (SSBA) for the ground truth. We use a simple velocity model and fix the intrinsic camera parameters since we are only interested in roll and pitch. The resulting accuracy is below 0.05 degree. We noticed that the initial X and Z values and yaw angle have been almost untouched (below 0.01%) by the SSBA per frame. Most changes were done with the Microsoft Kinect in respect to both Y and pitch, which is natural due to us controlling the motorized tilt according to the robots motion.

[10] http://www.inf.ethz.ch/personal/chzach/index.html

Figure 2.12: SLAM map of the environment using the *gmapping* package and two SICK laser scanner

3 | 2D Sensor Processing

The human visual perception uses multiple cues for estimation depth and recognizing structure. Besides using cues from binocular vision, like stereo vision in robotics, the human visual system uses other cues from monocular vision, for instance scale of known objects, motion parallax, lightning and shading. Another strong cue of monocular human vision is the linear perspective of objects: objects with the same size seem to ”shrink” when they are far apart and appear ”bigger” when they are closer together. Another effect is that from a standing point of view in a perspective, structures like lines that are parallel from an aerial view seem to meet at one point, see fig. 3.1. This kind of effect can be observed in cities like Manhattan, NY or also in many indoor environments.

The effect from figure 3.1(b) is referred to as the **vanishing point** in the computer vision literature [50]. According to Hartley and Zisserman [50] it is possible to recover the structure (e.g. a line) in the 2D image that belongs to a vanishing point, e.g. if the

(a) Front (b) Aerial

Figure 3.1: One Manhattan system from two points of view. The parallel red lines seem to meet at one vanishing point in the front view.

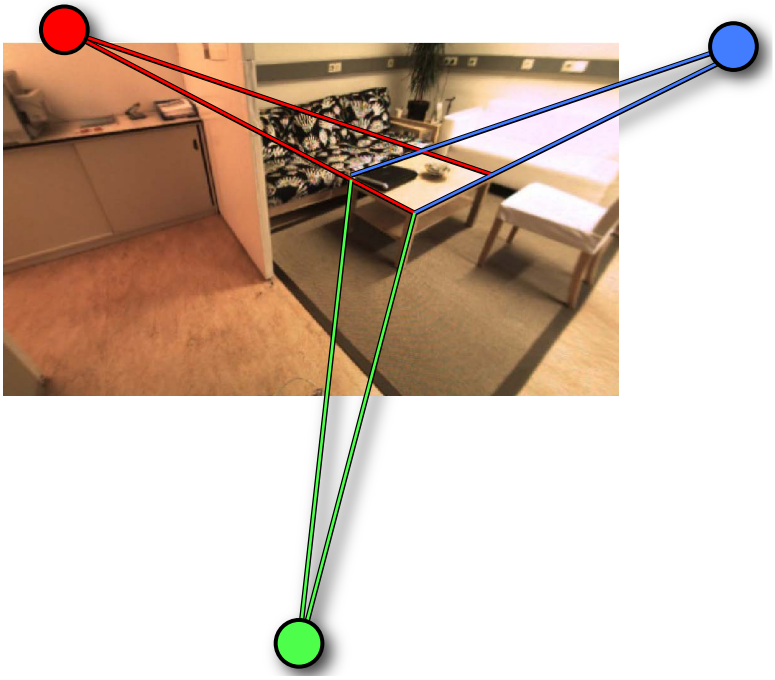
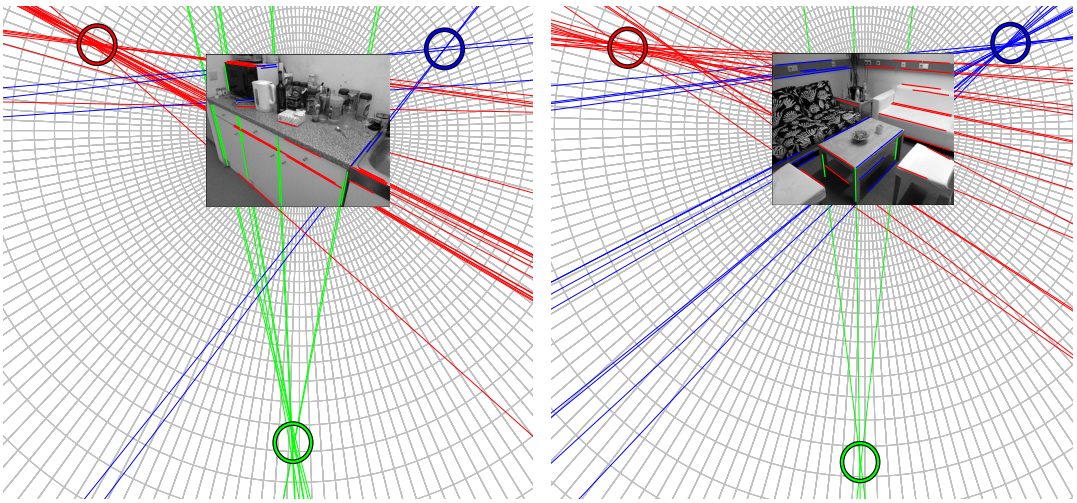


Figure 3.2: Basic Idea of vanishing points (shown as big circles) and 2D geometry: The table is aligned to the global Manhattan system and to the three vanishing points in 2D. Note that the vanishing points are outside of the image. The vanishing points are color-coded for better visibility: red=x, green=y and blue=z

structure is aligned with it. With no prior information, it is impossible to recover the size of the structure in 3D, except the 3D dimension of at least one structure is known. Here the same principle of scale of known objects is used like with human vision.
The main idea of our approach with 2D image processing is to exploit the concept of *vanishing points*. Technically a vanishing point is nothing more than the representation of a Manhattan system axis that is projected back to a 2D coordinate system (using infinite depth). In the case of one Manhattan system we have 3 vanishing points per image. If a structure in the real world is aligned to one axis, then the 2D representation (e.g. a line) will "vanish" in the corresponding point, see fig. 3.2.

Our method works in three steps: first, we roughly estimate the global and local Manhattan system geometry configuration using line detection and MSAC methods on the Gaussian sphere. In the second step we refine the estimates by projecting them back to the 2D image and applying local optimizers. In the third step we extract exact lines that are aligned with the Manhattan system geometry. These resulting lines are hard to extract using no prior information about room structure i.e. low contrast



(a) Kitchen (b) Living Room

Figure 3.3: Example scene where all three vanishing points have been detected and marked with colored ellipses (circles mapped on a sphere). The lines have been colored according to the assigned vanishing point. The detected lines have been set to infinite length outside the main picture for better visibility. The grid shows the mapping of the 2D image to the Gaussian sphere

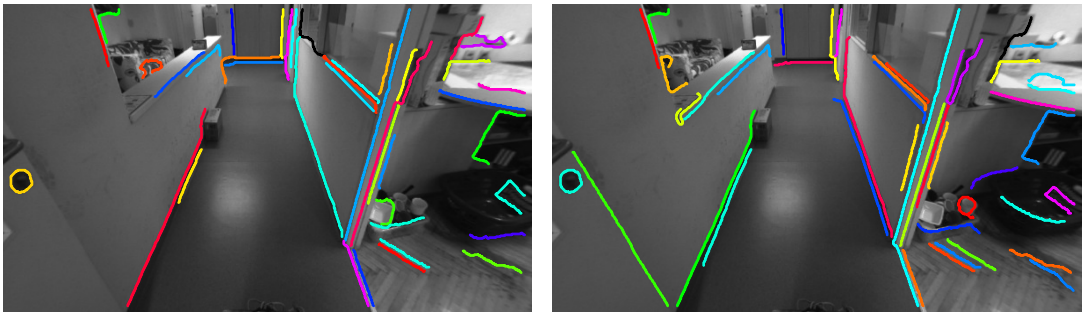
or blur. Flint et al [18] have shown that the extraction of lines can be significantly improved if the room geometry is known. We expand upon this idea by using local statistics on the image instead of grouping edges like Flint et al. [18].

3.1 Edge Detection

The first step of the 2D processing is the edge detection, which is made into lines in the next step and used to detect vanishing points for Manhattan system geometry. We detect and extract lines in this step with the typical computer vision method. First, we detect edges and group them into lines segments in order to fuse them into line segments with clustering. We assume that the 2D image of the 2.5d sensor is pinhole calibrated i.e. with little to no radial distortions.

In the first step we apply *contrast enhancement* to the image based on histograms to improve edge detection. Experiments have shown that 2D cameras of our sensors are quite different; while the AIT Stereo Vision uses a fixed exposure setting for the RGB camera, Microsoft's Kinect uses an auto exposure setting that always adjusts to the brightest light source in the image. This leads to different results with the 2D images.

Figure 3.4: Example scene where all three vanishing points have been detected and marked with colored ellipses (circles mapped on a sphere). The lines have been colored according to the assigned vanishing point. This image illustrates two cases of vanishing points, two outside the image and one far away. One can see that the lines of the far away vanishing point do not meet exactly in one point on the 2D image. The detected lines have been set to infinite length outside the main picture for better visibility. The Grid shows the mapping of the 2D image to the Gaussian sphere.

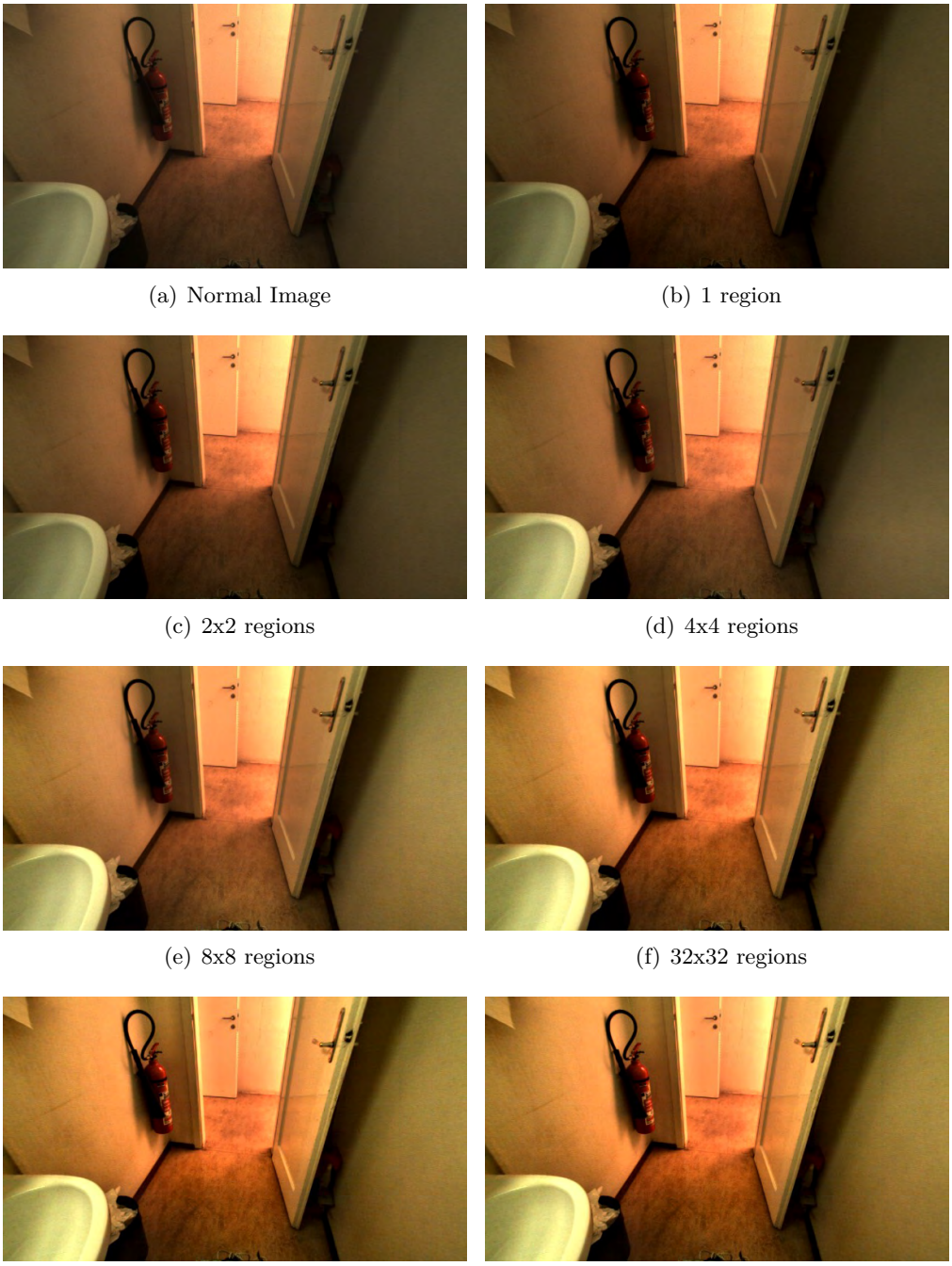


(a) Global Enhancement (b) Local Enhancement

Figure 3.5: Using local contrast enhancement to improve edge detection with canny edge detector: both images use identical settings for edge detection. The left image was normalized using one min/max for all values. The right one uses local min/max values with blending within a 8x8 regions/grid. One can see that far more (connected) edges are detected in the darker areas in front. The images are taken from AIT Stereo Vision.

While images with Microsoft's Kinect seem to be quite bright, images with AIT Stereo Vision appear darker with more contrast compared to Microsoft's Kinect. Due to our nature of our application, both fixed and auto exposure settings are not optimal since we observe two rooms with varying sources of light and brightness. The goal is to improve parts of the image that seem to be too dark (see fig. 3.5).

The contrast enhancement [55] per image works as such: we generate a histogram displaying values of overall brightness for a specific region of the image. The color values are converted to grayscale using the weighed averaging of the color components according to the Advanced Television Systems Committee (ATSC) BT.709. The BT.709 norm is a good trade-off of human color perception and computational speed, since it only requires three multiplications to calculate grayscale values. Note that both Microsoft's Kinect and AIT Stereo Vision deliver Bayer RGGB image for color and not a YUV component image. The histogram is used to estimate the approximate minimum and maximum values, such as minimum brightness and maximum brightness. Instead of using these values directly, we use a minimum value that represents either at least 1% (of the dark brightness values) of the data shown in the histogram or 99%. In many cases glossy spots are far brighter in the image than the actual brightest areas and represent less than 1% of the image. The same holds true for noise in the dark regions. The contrast normalization is processed with simple brightness equalization using the estimated minimum and maximum as new boundaries. In order to avoid artifacts in



(a) Normal Image (b) 1 region

(c) 2x2 regions (d) 4x4 regions

(e) 8x8 regions (f) 32x32 regions

(g) 256x256 regions (h) per Pixel

Figure 3.6: Comparison of local contrast enhancement using a different number of regions per image. The sample image (AIT Stereo Vision) shows an area with a dark room and bright corridor. One can see that the contrast in the dark room is enhanced with more regions. This is best visible on the wall on the right.

low contrast areas, we use an absolute threshold for the estimated boundaries, such as 20% for minimum values and 80% for maximum values. In order to maintain precision

(a) No blending (b) With blending

Figure 3.7: The importance of blending regions (here 32x32) to avoid artificial *banding* artifacts (AIT Stereo Vision). These artifacts can cause false edges with edge detection.

with the data we use a *floating point* representation of the output image rather than an integer representation.

In practice we use 8x8 overlapping regions within the 2D image to enhance the contrast of each individual block. These blocks overlap 50% with adjacent blocks to enable a smooth transition between them, see fig. 3.7. The estimated minimum and maximum values are also blended per pixel within the blocks as well as with estimates from adjacent blocks. This allows for an almost artifact-free contrast enhancement, which differs from per pixel contrast enhancement. In our approach, we do not use a true per pixel contrast enhancement with just neighboring pixels for histograms because it would take too much computational power. Please note that the extracted lines are only used to estimate the Manhattan system geometry, the lines are then extracted in another step. Since we use this kind of enhancement, we also ”enhance” the amount of noise that is created by artifacts in the detection of corners. Those artifacts are rejected within in the line detection in the next step. The runtime of our method on the SXGA Microsoft's Kinect is less than $\approx 12ms$ (see page 27, only one thread) since we compute 64 histograms (with 256 bins) each from $\frac{1}{32}$ of the image within linear runtime. A contrast enhancement per pixel needs $\approx 36s$ for one image.

There are many ways to detect edges such as calculating the first derivative of the image like Sobel [55] or filtering the second derivative like Laplacian filter [55]. We use a standard technique for edge detection, the canny edge [30] detector proposed by J. Canny in 1986. The image is smoothed with a Gaussian blur to reduce the noise. We use a 5x5 kernel, which produces high quality results quickly. In the next step the image gradients are extracted, using a first derivative Sobel filter for the x and y axes and the orientation for each pixel per gradient is calculated. The orientations are passed

trough a quantization and divided into four directions: vertical, horizontal and diagonal (such as angles from 0 to 180 degrees). North and south directions are reduced to only horizontal ones. The orientation per pixel is used in the non-maxima suppression to extract dominant gradients; a pixel is used as an edge if its gradient is the highest in the 5x5 neighborhood within gradients in counter directions. For instance if the pixel has a vertical orientation it is an edge if there is no pixel with horizontal orientation and a bigger gradient value than the vertical one. The output is a binary image with edges and a gradient value. In the last step the edges are connected to *linked edges* if the gradient of each pixel is a greater amount than a particular threshold. Assuming a floating point metric image with values from 0 to 1, we use 0.3 to categorize strong visual gradients. This can result in fragments within line-like structures. The canny edge detection uses a second threshold for hysteresis edge linking. The idea is that an edge with a lower gradient than the normal threshold can still be connected to another structure with linked edges, as long as the gradient of the edge is larger than the hysteresis threshold and the structure is in 3x3 pixel area. We use 0.15 as the hysteresis threshold. This results in 80% of the holes in line-like fragment structures being filled with hysteresis pixel. It also leads to artifacts within areas of high noise. The hysteresis is iteratively applied as long no new pixels are added to a linked-edge structure. The output is a binary image containing only the edges and is extracted in SXGA in less than $2ms > 1ms$ for AIT Stereo Vision).

3.2 Straight Line Detection

The next step in the 2D processing chain is to group the edges into straight line segments: raw *linked-edges* are transformed into *line segments* using the well known *split and merge* [56] method. The idea is to identify pixels that can be grouped into a line within a certain maximum error, referring to the distance of the pixel to the line model itself. Pixels with an error greater than this maximum error are excluded or "split" from the first pixel set and moved into a new set.

We use a similar method to the original Ramer–Douglas–Peucker algorithm [56]: Let $\mathcal{E} = (b_1, ..., b_n)$ be *linked-edges* segment containing n pixels with $b_i = B(u_i, v_i)$ using $u, v \in \mathbb{R}$ as coordinates in the 2D image. First we draw line $\overline{b_1 b_n}$ to build a line-segment from the first b_1 pixel to the last b_n pixel and calculate the pixel b_i $(b_1 < b_i < b_n)$ with the largest *perpendicular* distance Δ to the build line. If $\Delta > \varepsilon$ is greater than the

maximum error ε the line $\overline{b_i b_n}$ is split into two new lines: $\mathcal{E}_1 = \overline{b_1 b_i}$ and $\mathcal{E}_2 = \overline{b_{i+1} b_n}$. We recursively apply this method on all $\mathcal{E}_n$ line segments as long no more line is split into two new lines anymore i.e. if Δ does not exceed ε. In the following step we merge adjacent, e.g. $\mathcal{E}_i, \mathcal{E}_{i+1}$ line segments if the relative angle between them

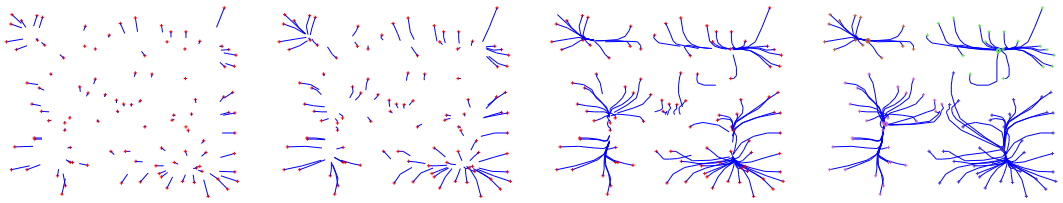
$$\varphi = \frac{\vec{\mathcal{E}_i} \cdot \vec{\mathcal{E}_{i+1}}}{|\vec{\mathcal{E}_i}| \cdot |\vec{\mathcal{E}_{i+1}}|}$$

is less than ϑ. We apply the split and merge sequentially as long a line is split, merged or a certain number (e.g. max iterations = number of pixel per line-segments) of iterations have been exceeded. For example, the maximum number of iterations equals the number of pixel per line-segments.

We use an $\varepsilon = 1.25$ to extract line segments with little "line to pixel" error and $\vartheta = 2$ degree. This results in a short, but precise line. In practice the amount of splits over merge is 10:1 since we allow only a small angular error for merging. The split and merge is executed on our machine (see page 27) within 4ms for 300 lines with ≈ 50 pixel length on one CPU.

In some cases a line can be detected in several parts due to factors such as noise during the sensor processing, occlusion, bad illumination or motion blurring. The resulting effect is visible in figure 3.5(b) on page 33: the lines in the pseudo color coding on the bottom left are interrupted by several pixels. Since lines are used to estimate vanishing points they need to be as exact as possible regarding accuracy. The accuracy depends on the number of pixels that make the line since we use a discrete grid. Experiments have shown that lines that are shorter than 12 pixels have lack of accuracy. These lines are usually rejected in the pre-processing since they negatively affect the accuracy when estimating the vanishing point.

We group lines based on the clustering of loosely-aligned line segments. A common technique for this is *Random sample consensus* (RANSAC) [57]. RANSAC is an iterative method that estimates parameters to a given model from a dataset with outlier rejection. The method uses an arbitrary number of model parameters in a configuration that is drawn from the dataset. Then this is evaluated by applying the model to the data itself. The data points that fit to the model within a certain threshold are counted per configuration. The models with the most inliers are chosen as an estimate.

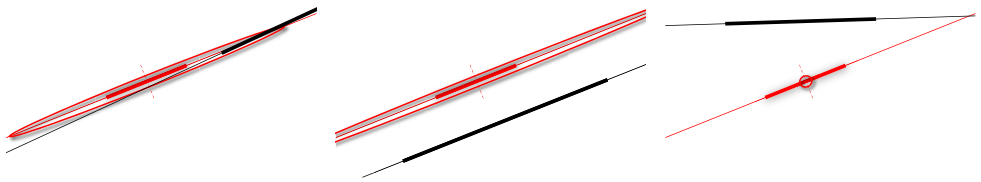


(a) 1. Iteration (b) 2. Iteration (c) 5. Iteration (d) 8. Iteration

Figure 3.8: Illustration of the mean-shift algorithm on 100 random data points and a Gaussian kernel (red points are data points, blue lines are the trajectories). One can see that the algorithm converges to clusters after less than 10 iterations

At this point we do not use RANSAC-based methods. This is because the short lines can show a greater error in the true line orientation than in longer lines. Since this causes issues with the orientation matching, and assuming that all segments are next to each other, this can lead to too many short lines that are not grouped correctly with longer ones. We can use a bigger threshold, but it can lead to artifacts. We also do not use RANSAC-based methods because of the non-deterministic nature of RANSAC itself. The actual number of lines is not previously known; therefore, the number of iterations to group all straight lines through RANSAC, is difficult calculate. The ratio of broken, interrupted lines with many segments to the non-broken lines is highly dependent upon the robot motion and the environment itself. For instance, we can observe less broken lines if they are far away, but these same lines are undetected if the robot rotates on the spot due to motion blur. Lines that are close to the robot are detected in many cases, but they are usually broken into parts depending on the speed of the robot.

We use the iterative generalized *mean shift* [27, 58] algorithm to group the line segments. The algorithm repeatedly shifts every data point to the mean of all data points within a defined area. In our case the area is defined as a kernel function $\mathrm{K}(x,l)$ and is assigned a weight depending on the distance from point x to point l, for all adjacent points. Note that the standard mean shift is using the Euclidean distance directly as a parameter for the kernel function. The advantage of this algorithm is that the number of clusters (here straight lines) do not have to be previously known. Let $l(x,y,\theta,\lambda)$ $x,y,\theta,\lambda \in \mathbb{R}$ be a straight line segment centered at x,y with orientation θ and length λ. Let $\mathcal{L} = \{l_1,...,l_n\}$ be a list of straight lines. First all line segments $\mathcal{E}$ are converted to $\mathcal{L}$ if they are at least 12 pixels long. Beginning with the line points $l \in \mathcal{L}$, the sample



(a) Aligned (b) Parallel (c) Not Aligned

Figure 3.9: Illustration of the kernel function $\mathrm{K}(x)$ of the (red) line to another (black) line. The ellipsoid is plotted at 3σ

mean shift $\mathrm{m}(x)$ is defined as

$$\mathrm{m}(x) \quad = \quad \frac{\sum_{l\in\mathcal{L}} \mathrm{K}(x,l)\mathrm{w}(l)l}{\sum_{l\in\mathcal{L}} \mathrm{K}(x,l)\mathrm{w}(l)} \tag{3.1}$$

with $\mathrm{w}(l)$ as a weighed function. The weight is the sum of all gradient pixels that the line segment is made of. This puts a bias on lines which are grouped from strong gradient edges. The difference $\mathrm{m}(x) - x$ is referred to as a *mean shift* in the literature [58]. The repeated movement of data points to the sample means is called *mean shift algorithm*. In each iteration of the algorithm, $l \leftarrow m(l)$ is performed for all $l \in \mathcal{L}$ simultaneously, see fig. 3.8. The algorithm is applied as long as either maximum number of iterations is reached or if all shifted points moves less than a certain threshold within one iteration.

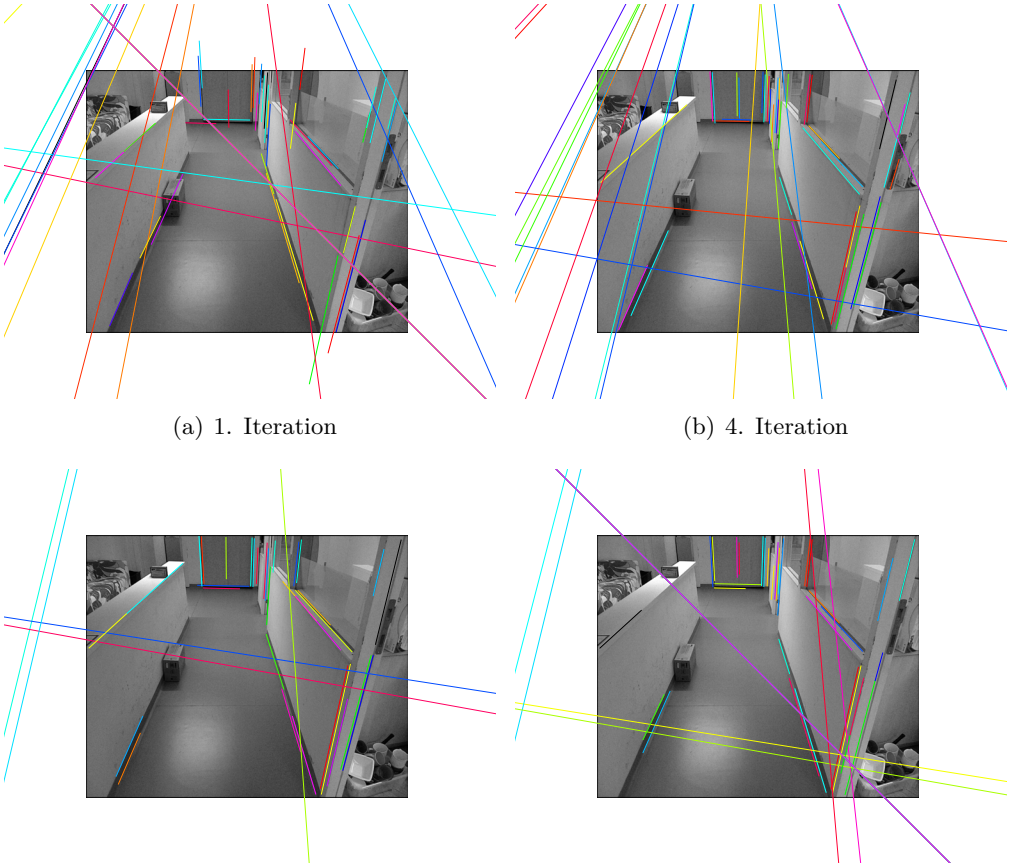
The kernel function $\mathrm{K}(x,l)$ is the heart of the mean shift algorithm and dependent upon the length x_λ of a line and the relative angle $\gamma_{x,l}$ ($\gamma_{x,l} = 0$ are parallel lines) from x,l and the Euclidean distance $\delta_{x,l}$ from the x and y coordinates (denoted with u,v). The idea is to model the kernel in the style of an ellipsoid as shown in fig. 3.9. The line direction is used as a major axis, while the orthogonal one as a minor axis. It be written as a vector

$$\vec{K_l} = \begin{bmatrix} \mathrm{gaussian}(\delta_{x,l}, x_\lambda)\mathrm{gaussian}(\gamma_{x,l}, \gamma_\sigma) \\ \mathrm{gaussian}(\delta_{x,l}, \sigma_y) \end{bmatrix}$$

using a normalized Gaussian function such as using η as normalizing constant

$$\mathrm{gaussian}(x,\sigma) = \eta \frac{1}{\sigma\sqrt{2\pi}} e^{-(B(u,v)'')^2/2\sigma^2}$$

and γ_σ as bias for the relative angle and σ_y for the minor axis. We use a constant



(a) 1. Iteration (b) 4. Iteration

(c) 16. Iteration (d) Final 49. Iteration

Figure 3.10: Using Mean shift to group line segments (36) to straight lines (25) from Microsoft's Kinect. The lines (thick line) and their direction are grouped in a iterative process to reduce the number of broken lines. The segments on the right are grouped into 6 segments for initially 11. Note that the pseudo coloring is not consistent in the subfigures

value for the minor axis to prevent parallel lines being grouped with each other, and so that only aligned lines should be grouped together. In practice we use $\sigma_y = 0.7$ pixel and $\gamma_\sigma = 2$ degrees just to allow some tolerance when grouping the lines. So far the vector $\vec{K_l}$ has not been aligned with the minor and major axis. This is done by applying the rotation matrix with the corresponding line orientation θ and obtaining $\mathrm{K}(x,l)$ by calculating the vector length

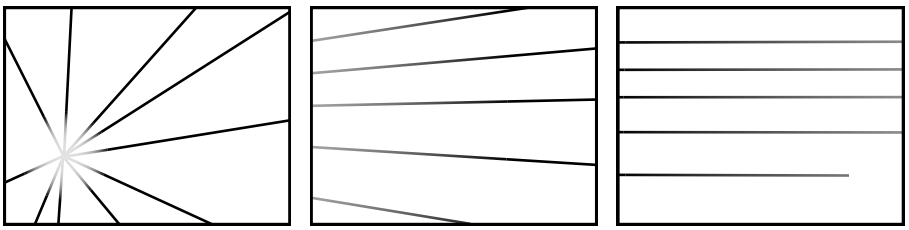
$$\mathrm{K}(x,l) = \left\| \vec{K_l} \begin{bmatrix} cos(\theta) & sin(\theta) \\ -sin(\theta) & cos(\theta) \end{bmatrix} - \begin{bmatrix} x_u - l_u \\ x_v - l_v \end{bmatrix} \right\|$$

We only shift, i.e. calculate the mean for each value, x, y, θ, we do not shift the length λ of the line because it is used to model the kernel itself. We determine 0.001

pixels as threshold for the x and y component and 0.01 degree for the orientation and use a maximum of 96 iterations. In many cases the convergence of data points is reached in less than a half of the iterations. Figure 3.10 demonstrates the results from the data from Microsoft's Kinect. One can see that the number of lines reduces over the iteration (grouped lines are drawn on top of each other).
The algorithm proceeds with an average runtime of $6ms$ for 50 line segments with **no** multi-threading. We apply a k-d tree to enhance the runtime with a computational complexity of $O(n \log n)$. AIT Stereo Vision obtains 48 line segments per image in many cases, with Microsoft's Kinect 64; $\approx 35\%$ of the line segments can be reduced using grouping.

3.3 Vanishing Point Estimation

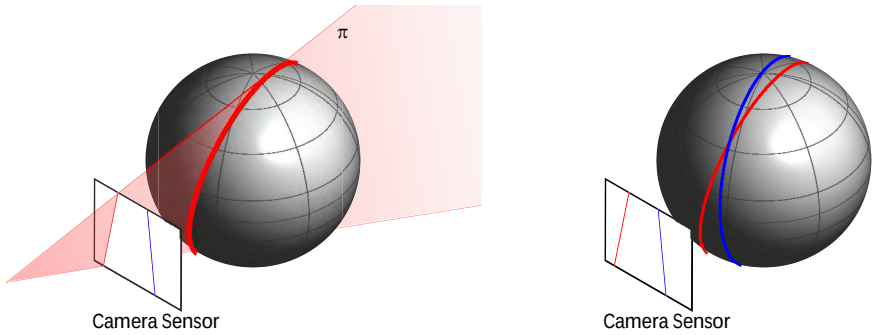
The detection of the *vanishing points* is the third step in the 2D processing chain. The main objective is to detect Manhattan system geometry in the 2D image that reflects the three main axes (see fig. 3.1 on page 29 for an example). First the global system is detected, then the local ones. Our approach is to directly detect a valid Manhattan system geometry with all three axes instead of determining the vanishing points independently and grouping them afterwards. We use robust estimators with outliers to estimate the systems using the Gaussian unit sphere rather than working on the planar 2D image. Then we refine the vanishing points by relaxing the orthogonal constraint.



(a) Inside (b) Outside (c) Far away

Figure 3.11: Samples for vanishing points on a planer surface. The vanishing point is located where all lines meet at one point. One can see the perspective of the lines strongly depends on the location of the vanishing point. For instance lines seem to be almost parallel when a vanishing point is far away from the image centre. Adapted from on Flint et al. [18]

Technically speaking, a vanishing point on a 2D image is where many lines meet at one point as shown in figure 3.11 assuming they belong to the same vanishing point.



(a) The (red) plane π intersects the sphere through its center

(b) The red and blue line are represented as ellipsoids

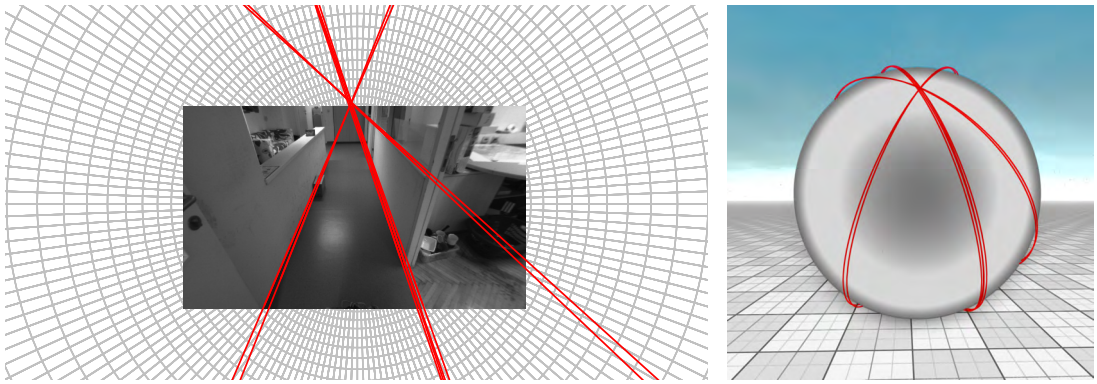
Figure 3.12: Two lines projected on the Gaussian unit sphere: The lines are projected as plane π through the camera sensor and intersects the Gaussian unit sphere

We can assume that all three shown cases (fig. 3.11): *inside*, *outside* and *far away* in one 2D image with valid Manhattan system geometry. This is also shown in fig. 3.4 on page 32: At least one vanishing point is *outside* and one *far away* due to the orthogonal structure of the Manhattan system geometry. One exception is when the observer is oriented at a 45 degree angle to all three normal axes (see fig. 3.3(a) on page 31)) only the *inside* and *outside* cases will show.

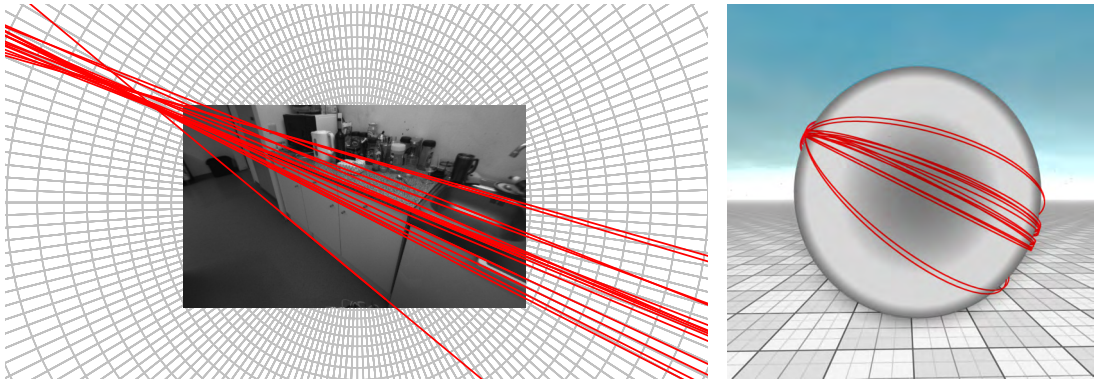
Many computer vision approaches utilize the 2D image space to detect vanishing points [18, 21, 50, 59] and use custom metrics to deal with the three vanishing point cases (Inside, Outside, Far away). We use the concept of the Gaussian unit sphere (see fig. 3.12 and fig. 3.13) to transform lines from a plane to ellipsoids on a sphere. The befit of the method is that two ellipsoids on the Gaussian sphere always intersect at two points as long they do not overlap each other. We can describe each point on the sphere with a vector and can calculate the distance of two points as an angle[1] rather than as pixels. This allows us to use one method to cover all cases as shown in fig. 3.11. Figures 3.4, 3.13, 3.14 and 3.3 show the Gaussian unit sphere mapped back to a 2D image as grid.

The vanishing point detection itself consists of two steps: First we estimate a rough global Manhattan system geometry that is represented by the three vanishing points using robust estimators with random sampling. The next step is to refine the points and relax the Manhattan system geometry assumption. The relaxation is done in two

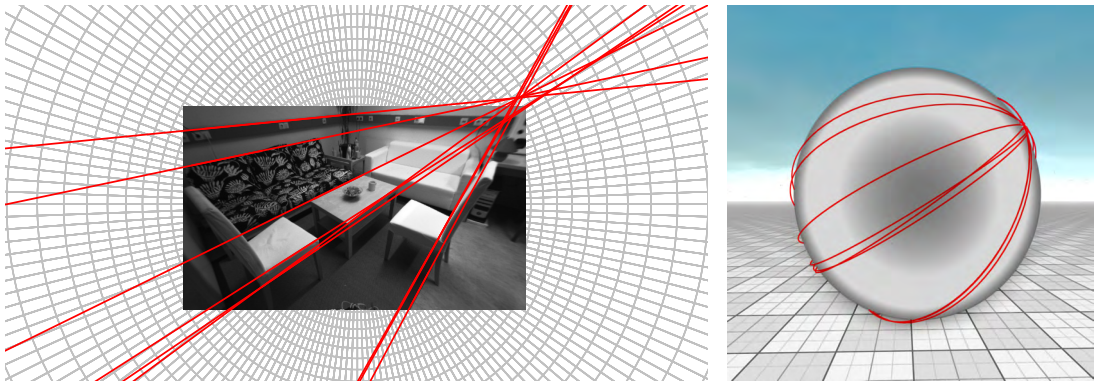
[1] Note that we use Euclidean distance on the sphere surface which is redundant to angles



(a) Corridor (Microsoft's Kinect)



(b) Kitchen (Microsoft's Kinect)



(c) Living Room (Microsoft's Kinect)

Figure 3.13: Examples of real data using the Gaussian unit sphere. Only line from one vanishing point is shown for better visibility. One can see the line intersections in 2D and on the sphere

steps: the first one uses another local robust estimator for each point individually, while the second one uses a local optimizer. If the global system can be estimated, the local ones are extracted the same way by sharing one axis with the global system.

In many cases a line does not intersect the vanishing point itself but rather its direc-

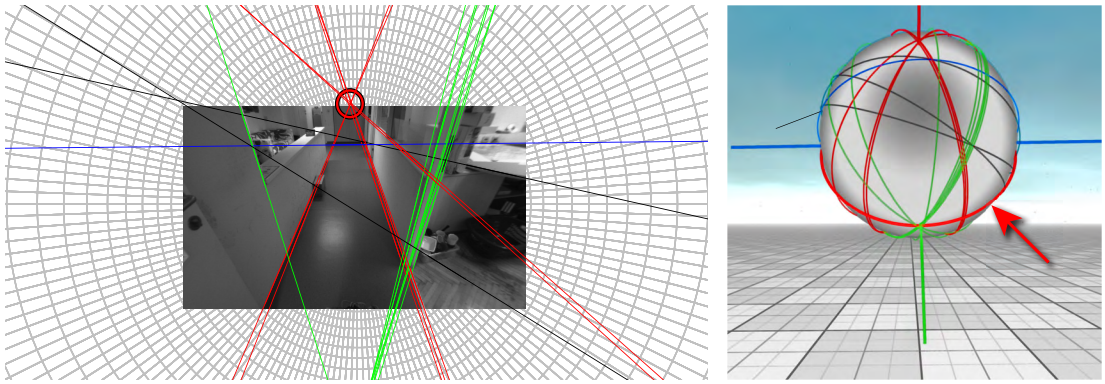
tion. Due to this we set the length of the lines, before we use them on the Gaussian sphere. First the lines with infinite length are projected as planes from the optical center of the camera sensor to the center of Gaussian unit sphere. The plane π is given in the standard Hessian normal form i.e. $ax+by+cz+d=0$, $\hat{h}(a,b,c,d)$. All planes are centered at the same point so they always intersect as long they are not identical or do not overlap. Let $\mathcal{P}=\{\pi_1,...,\pi_n\}$ be planes on the Gaussian spheres that have been converted from the grouped lines $\mathcal{L}$ from the previous step. Let intersect$(\hat{h_a},\hat{h_b})\in\mathbb{R}^3$ be a function to calculate intersection[2] of two planes in the Gaussian unit sphere. The function calculates the intersection as a line and converts it to a unit vector in the 3D space. Note that a vector in the unit sphere is considered to point in both directions, known as the normal vector and the inverted vector[3].

We use a variant of the well known Random Sample Consensus (RANSAC) [57] algorithm of the rough estimation: the M-Estimator SAmple Consensus (MSAC) [50]. RANSAC-based methods obtain their estimate by randomly selecting coefficients from a given dataset to a known model. The estimation is validated by applying the model (with estimated coefficients) to the data set and counting the number of points. $\mathcal{P}$ (=inliers) support the model within a certain threshold $t\in\mathbb{R}$ with error to model $e\in\mathbb{R}$ i.e. if $e<t$ the point is an inlierer, otherwise outlier. The estimation is iterative, in each iteration a new model is estimated and the number of inliers are counted. After a fixed number of iterations k, the model with the most inliers (=support) is used as output. MSAC accumulate the error of the model from the original data instead of counting inliers within a specific threshold for a model with the most support. The model with the lowest error is chosen. The error function C for a model with $i\in\mathbb{N}$ planes $\mathcal{P}$ is given as

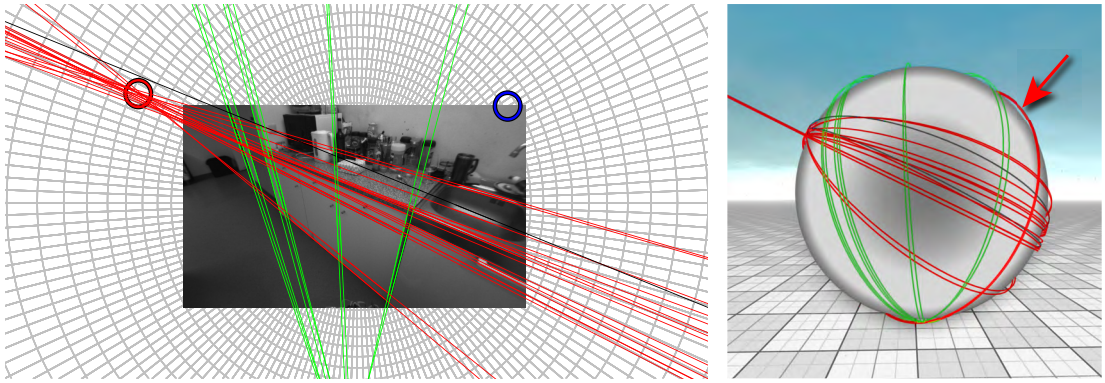
$$\mathrm{C} \;=\; \sum_i \delta(p_i) \text{ with } \delta(e_i)=\begin{cases} e^2 & e^2<t^2 \quad \text{inlier} \\ t_{max}^2 & e^2\geq t^2 \quad \text{outlier} \end{cases} \tag{3.2}$$

with $t_{max}>t$ as constant maximum error.
We use three randomly chosen planes $\pi_1,\pi_2,\pi_3\in\mathcal{P},\pi_1\neq\pi_2,\pi_2\neq\pi_3,\pi_1\neq\pi_3$ to construct a valid Manhattan system geometry within the Gaussian sphere. The planes π_1,π_2 are used to calculate the first axis of the Manhattan system geometry by intersecting them with $\vec{\mathsf{N}}_1=\text{intersect}(\pi_1,\pi_2)$. The second and third axes are obtained

[2] see Hartley and Zisserman [50] for detail
[3] This is the counterpart to the normal vectors in the 3D sensor processing



(a) Corridor (Microsoft's Kinect). The Gaussian sphere has been rotated for better visibility



(b) Kitchen (Microsoft's Kinect)

Figure 3.14: Examples of an estimated global Manhattan system geometry using MSAC. The red axis is the first axis that is estimated to use two random ellipsoids/-planes. An orthogonal ring is constructed and intersected with a random ellipsoid. This point is used as a second axis. The third one is constructed using the cross product of the first and second one. The planes have been colored according to their relationship to a vanishing point. Black planes do not have a relationship to a vanishing point

using a orthogonal plane that is constructed from the first axis (see the red arrow in fig. 3.14). $\vec{\mathsf{N}}_1$ is transformed into a Hessian plane with $\hat{\pi}(\vec{\mathsf{N}}_{1_x}, \vec{\mathsf{N}}_{1_y}, \vec{\mathsf{N}}_{1_z}, 0)$. Note that this is only valid because all planes intersect with the center of the Gaussian sphere. The second axis is aquired by intersecting the new plane with the third plane π_3 with $\vec{\mathsf{N}}_2 = \text{intersect}(\hat{\pi}, \pi_3)$. The third (redundant) axis is given with the cross product with $\vec{\mathsf{N}}_3 = \vec{\mathsf{N}}_1 \times \vec{\mathsf{N}}_2$. This approach is similar to Bazin and Pollefeys [60].

We define the error metric for all planes in the Manhattan system geometry model for the MSAC estimator. One way is to count the number of intersections, or the distance to the vanishing point, of all planes near the vanishing points [50, 59]. In many cases planes do not intersect directly at a vanishing point, but instead pass nearby, due to

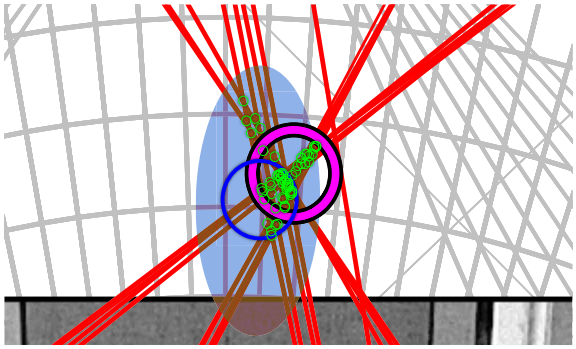


Figure 3.15: Refinement of the vanishing point using intersections of planes here shows the 2D projected plane from the Gaussian unit sphere. The blue circle shows the estimated vanishing point from the rigid Manhattan system geometry estimation. The green ones demonstrate the intersections while the thick magenta area displays the use of MSAC with least square fitting. The light blue area shows the 5 degrees distance to the non-optimized vanishing point with only intersections from this area used. The image is a detailed view, seen in fig. 3.14(a)

motion blur or poor camera calibration. We calculate the distance of the vanishing point as $\vec{\mathsf{N}} \in \mathbb{R}^3$ to the plane e.g. $\pi \in \mathcal{P}$ by projecting the vector on the plane [50]

$$\text{delta}(\vec{\mathsf{N}}, \pi) = |\pi_a \vec{\mathsf{N}}_x + \pi_b \vec{\mathsf{N}}_y + \pi_c \vec{\mathsf{N}}_z + \pi_d|$$

Note that this also is only possible because all planes intersect the Gaussian sphere center and the vector is a vector pointing in point in both directions simultaneously. Now we calculate the error per plane to its best matching vanishing point i.e. and use the distance (closest-point-on-plane to vanshing point) e for the MSAC error metric function. We use a t_{max} value for the MSAC that reflects an angle of 5 degrees on the Gaussian unit sphere and we use 512 iterations for MSAC. An estimated system is valid if at least two vanishing points are supported by at least 3 planes, such as with the inliers seen in RANSAC based methods. Figure 3.14(b) gives an example of a case where only two vanishing points are supported by the planes.

The estimation of the vanishing points is the result of a best fitting model for all three axes after averaging the error. In some cases the estimation can result in a trade-off with planes fitting "almost" to the rigid Manhattan system geometry assumption, see fig. 3.15. Due to this we apply the last step of our vanishing point detection: the refinement. The refinement relaxes the Manhattan system geometry by optimizing each vanishing point individually within certain limits. As long as the following two

conditions are met, each plane/line is assigned to a vanishing point. First, the error $e = \text{delta}(\vec{\mathsf{N}}, \pi) < t_{max}$ to the vanishing point is less than t_{max} and the plane can only be assigned to one vanishing point. If the plane cannot be assigned it is not used for the refinement, which is shown as black lines in all of the figures. The second condition is illustrated in figure 3.16: The three main vanishing points form a triangle where the relationship between the planes/lines/pixels cannot be determined. This is a common case with all Manhattan system geometry.

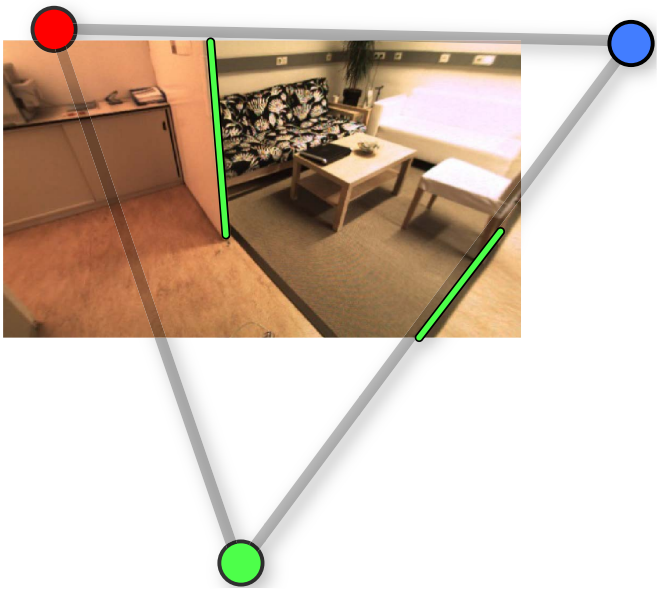


Figure 3.16: Example of an ambiguous vanishing point between lines. The one on the left can be assigned to the green vanishing point while the right one can belong to the blue and green one

We use the planes with the same assigned vanishing point to calculate intersections between all planes if they are less than 5 degrees away from the previously estimated vanishing point. The intersections are used for a second local MSAC estimator to obtain a more precise[4] the vanishing point. We use a t_{max} value for the second MSAC that reflects an angle of 1 degree on the Gaussian unit sphere and use 32 iterations. In the fashion of RANSAC estimator we use a weighted least square fit for the new vanishing point. A refined system is valid if at least 10 intersection support a new estimated vanishing point i.e. like inlier with RANSAC based methods. If no valid refined system is obtained the non-refined vanishing point is used. The variance N_σ for each

[4] the intersection of planes in many cases more precise than the shortest distance to a plane as before

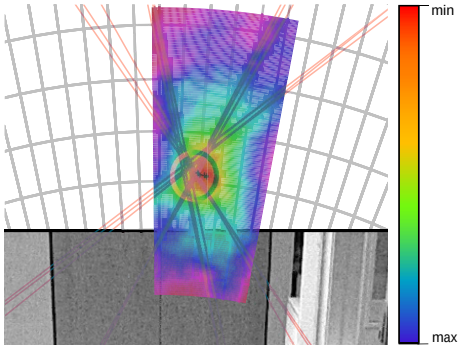


Figure 3.17: Plot of the entropy for refinement using histograms and hill climbing optimizers. The ring indicates the position of the non-optimized vanishing; the cross indicates the traveling of the hill climbing. The true position is found 0.02 degrees next to the original vanishing point. The image is a detailed view, see fig. 3.14(a)

axis is obtained from the variance of the second MSAC estimator. If the refinement is invalid, the variance of 5 degree is used. The variance is later used in the fusion of Manhattan system geometry.

An optional step is an additional optimization using a histogram and the well-known *hill climbing* [61] optimizer. Hill climbing is an iterative method that starts with an initial solution of a function $f(\vec{\mathsf{N}})$ and "optimizes" the solution by incrementally changing a single element of the solution. If the change produces a better solution, an incremental change is made toward the new solution. The method terminates if no further improvement below a certain threshold χ can be processed or if a certain number of iterations have been reached, see fig. 3.17. We use a variant that calculates a gradient of 5 (one centered and four in all major directions with a distance to the center point that corresponds to 0.001 degrees) sample points on the Gaussian unit sphere using the refined vanishing points are used as initial values. With each iteration the sample points are moved according to their gradient.

The heart of the hill climbing optimizer is the function $f(\vec{\mathsf{N}})$. The idea is to back project the vanishing point onto the 2D image and to then calculate this as a normalized polar histogram $p(\vec{\mathsf{N}})$. The histogram is centered at the projected vanishing point and uses the image gradients as input for the histogram. First all gradients are weighted ω according to their orientation (see chapter 3.1) to the back projected vanishing point. We apply a Gaussian weighing on the angular error and choose $\sigma = 15$ degrees. This results in a smooth gradient with the angular error which is needed due

to we use hill climbing optimizers. Note that the angular error only ranges from 0 to 90 degrees since we use the orientation[5] of a pixel and not the orientation as a normal vector.

We use multiple polar histograms in order to save the angular resolution of the pixels that are far away from the vanishing point, see fig. 3.23. The polar histogram is organized in rings with a fixed radius of 50 pixels. The number of bins depends on the distance to the vanishing point, or the number of rings. Since we use relatively small polar histograms we apply a linear function to determine the number of bins b per ring i that is $b = 360 \cdot (2i + 1)$. Using 360 bins and 50 pixel radius as a base is a good compromise between speed and accuracy. The average angular error using the sub-histograms is > 0.1 degrees with Microsoft's Kinect, > 8 degrees without. One alternative is to use a quadric function for both bins and radius per slice with a larger base radius such as 200 pixels. Due to computational and memory efficiency we use sub-histograms with a size of 360 bins that represent parts of the individual polar histogram. A sub-histogram is only allocated if at least one pixel lies within it.

[5] the orientation is consider to both directions

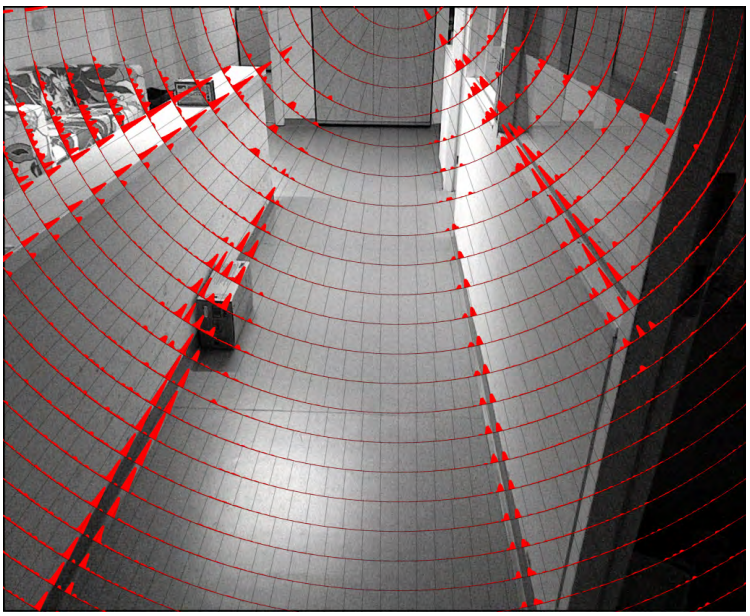


Figure 3.18: Sub-Polar histogram for a vanishing point. The gradients in the image are used as input. The histogram (red) bins are scaled with the square root for better visibility. The grid (black lines) represent 36 bins of the polar histogram

The error metric for the optimizer is calculated using Shannon entropy f (X = p(N)) with $f(X = p(\vec{\mathsf{N}}))$ with

$$f(X) = -\sum_{i=1}^{k} p(x_i) \log_{10} p(x_i)$$

where k where k is the number of bins of the histogram and $p(x_i)$ is the value in the histogram at bin i. In the case of $p_i = 0$ for some i, the value of the corresponding summand $0 \log_{10} 0$ is taken to be 0, which is consistent with the limit $\lim_{p \to 0+} p \log_{10}(p) = 0$. The overall idea of minimum entropy is depicted in figure 4.18 on page 98. Technically we perform a *hill descent* for optimization since we use minimum entropy. During an experiment, we use maximum iterations of 128 iterations and use χ to reflect 0.00000001 degrees.

The estimation of the local Manhattan system geometry is straightforward. After the global system is successfully estimated, we remove all lines that have been assigned to vanishing points including the ones with ambiguous relationships, see fig. 3.16. A random axis is chosen and a random plane is used to intersect the ring. The remaining processing is similar to the global one, except that here we only estimate parameters for the second and third axes since the first one have already been determined.

The algorithm runs on our test computer (see page 27) with an average runtime of $12ms$ (both Microsoft's Kinect and AIT Stereo Vision) for 50 line segments with **no** multithreading without the histogram based refinement. The first MSAC estimation uses 90% of the computational power due to the higher number of iterations. The histogram-based refinement runs with average of 300ms, with an average of 48 iterations and a linear runtime. The computational bottleneck is the generation of the histogram.

3.4 Line Extraction

The last step of the 2D sensor processing chain is the line extraction based on the estimated vanishing points. The aim is to extract lines from structure that is alligned to the vanishing point in the style of a guided line search [18]. The guided line search uses a gabor-based kernel filter [62] to extract structure that is aligned the vanishing point. Next a polar histogram, centered on the vanishing point, is used to detect candidate lines. The candidates are validated using a gradient threshold and grouped into a line if they are long enough. The major difference to the well-known canny edge detector

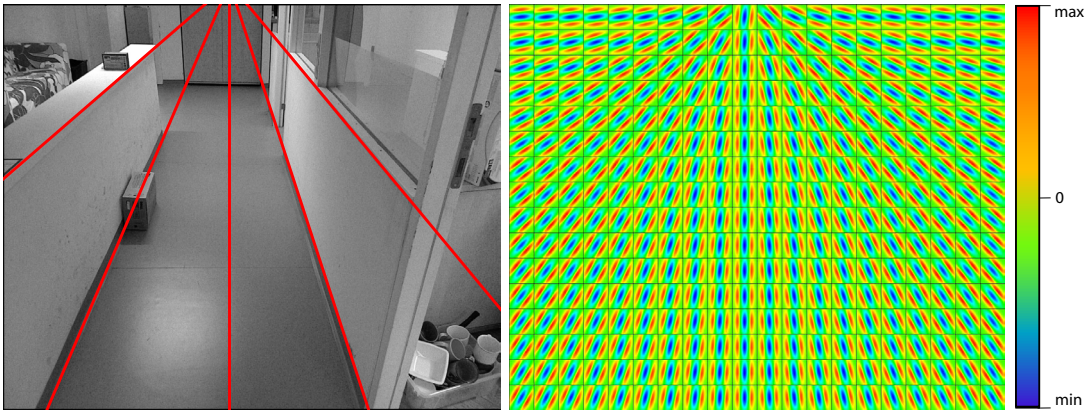
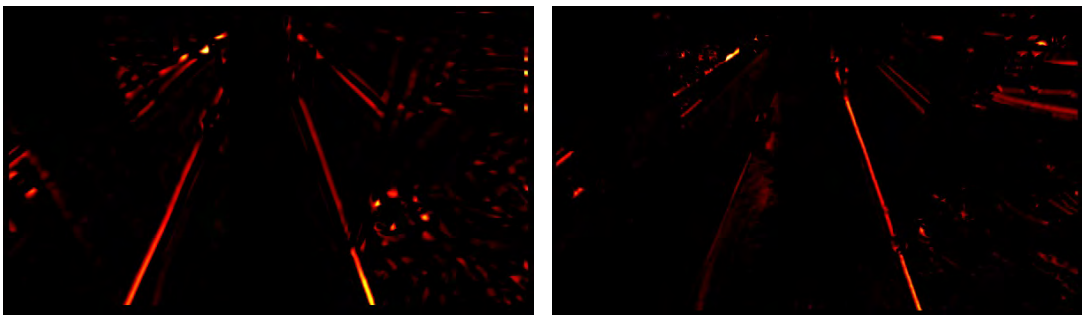


Figure 3.19: Using a parameterized Gabor filter (shown right) for edge detection. The Gabor filters are always oriented to the vanishing point of the left image (Microsoft's Kinect). We show the filter in a 20x16 grid using a 64x64 kernel for better visibility

is that we only look for lines in a specific direction rather than in all directions. This also allows us to use lower thresholds compared to the canny filter, since it would lead to noisy edges. The line extraction is completed individually for each vanishing point and is later used for the segmentation process in a later step.

The first step is to apply a Gabor filter for each pixel in the 2D image to obtain edges that are oriented to the vanishing point. Gabor filters are widely-used linear in computer vision for edge detection [63], face detection [64], object detection [65] and visual attention [66]. Named after *Dennis Gabor* (1900–1979) it is believed that Gabor filters are similar to the perception in the human visual system [62] for edge detection.



(a) Gabor (our method) (b) Sobel

Figure 3.20: Comparison of edge detection methods (AIT Stereo Vision, only gradient) with a priori known vanishing point. Both filters use a bias to suppress geometry for another vanishing point

The gabor filter is given as

$$\text{gabor}(u,v,\lambda,\theta,\sigma,\psi,\gamma) = \exp^{-\frac{u'^2+\gamma^2 v'^2}{2\sigma^2}} \cos(2\pi\frac{u'}{\lambda}+\psi) \quad (3.3)$$

$$u' = u\cos\theta + v\sin\theta \quad (3.4)$$

$$v' = -u\sin\theta + v\cos\theta \quad (3.5)$$

with (u,v) as image coordinates centered on $(0,0)$ in the fashion of kernel filters, (u',v') as rotated coordinates with the angle θ. λ represents the wave length of the gabor function (sinusoidal factor), σ is the sigma of the Gaussian envelope, ψ is the phase offset and γ is the spatial aspect ratio and specifies the ellipticity of the support of the function.

We parameterize the filter per pixel by setting θ to the relative angle to the vanishing point α to suppress structure that is not pointing to the point $\theta = \alpha + \frac{\pi}{2}$. Experiments have shown that it is best to use a second order derivative filter since it is more sensitive to avoiding artifacts $\lambda = \frac{\pi}{2}, \sigma = 0.56, \psi = 0.5$. If the kernel is small, a second order filter is more sensitive to noise than a first order one. The remaining parameter $\gamma = 3$ is set as the filter of width and height since we look for lines and not circular structures. We use a 15x15 kernel for Microsoft's Kinect and 7x7 for AIT Stereo Vision, which allows the filter to extract low contrast edges and suppress the noise.

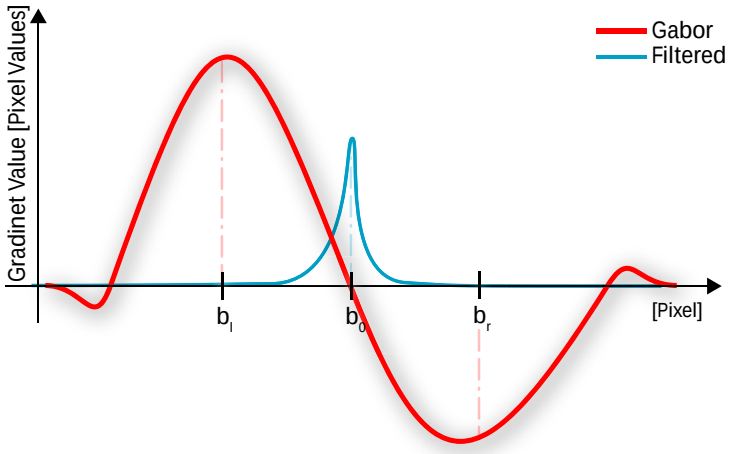
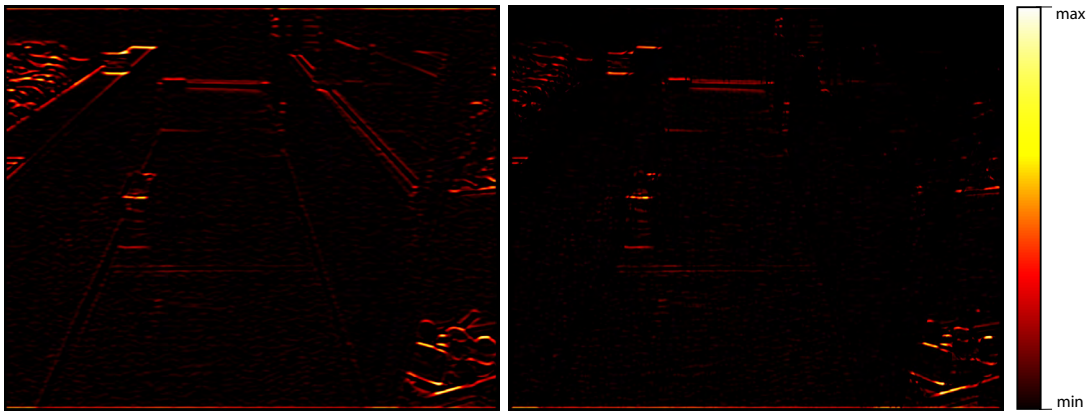


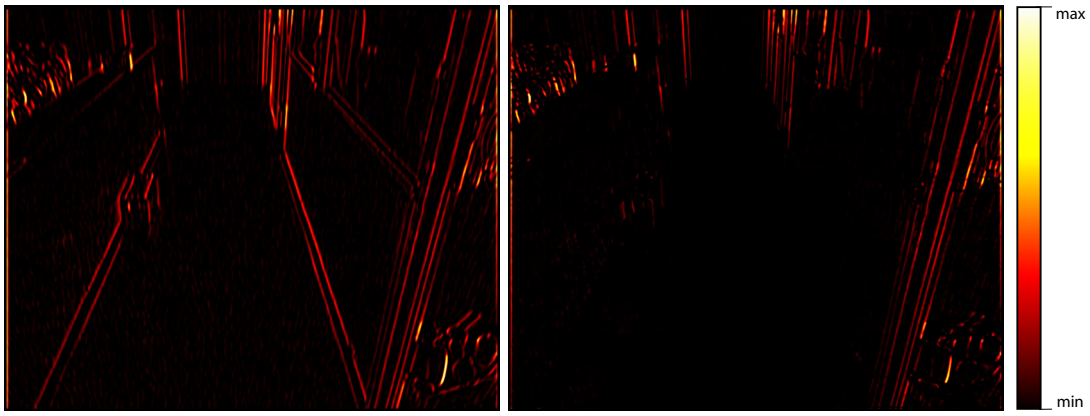
Figure 3.21: Processing of the gabor filter output (red) to gradient edges (blue)

The output values from the second-order derivative filter are transformed into edges by looking for where it reaches zero, namely the edge that is located between a negative

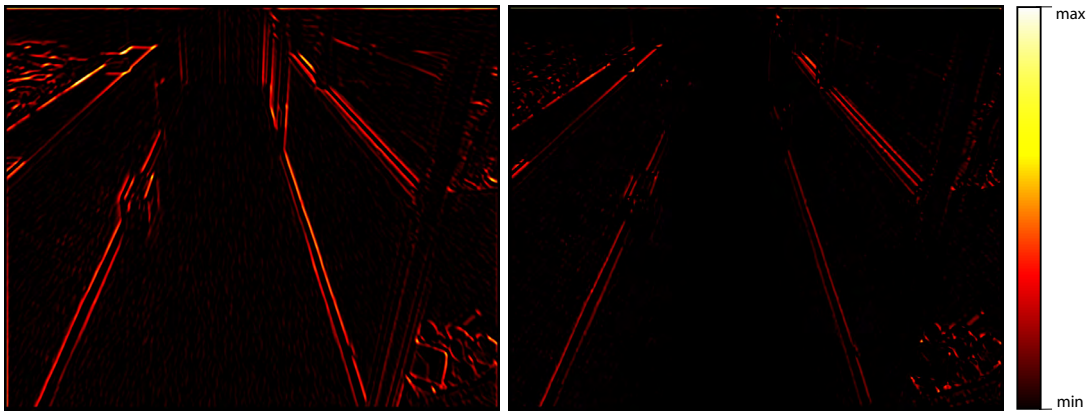
and positive peak. Ideally, both peaks have the same size. Let $b_0, b_l, b_r \in \mathcal{E}$ three pixels on the image $\mathcal{E}$: b_0 be the zero centered pixel, b_l the pixel for the left peak and b_r for the right one, see fig. 3.21. b_l and b_r placed $\frac{15}{4}$ away from b_0 and rotated orthogonally



(a) X Axis Vanishing Point Geometry



(b) Y Axis Vanishing Point Geometry



(c) Z Axis Vanishing Point Geometry

Figure 3.22: Detected edges using the Gabor filter after gradient conversion. The images on the left show the raw output while the one on the right shows ambiguous vanishing point membership suppression. The original image is shown in figure 3.19.

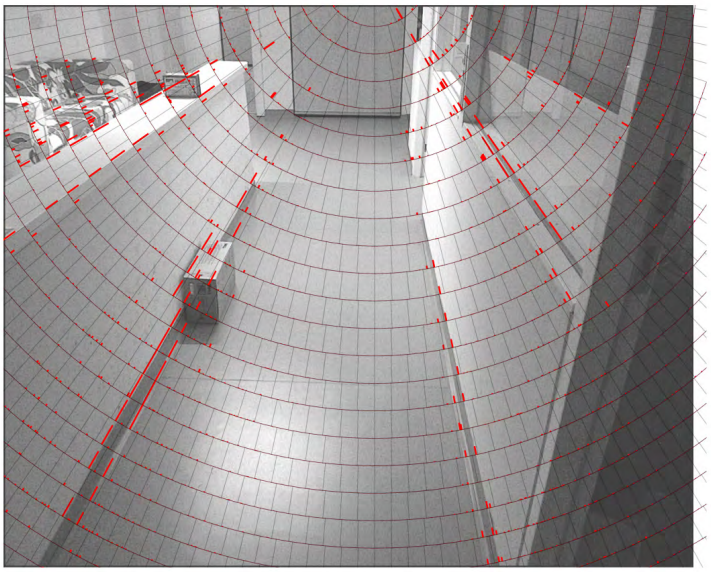


Figure 3.23: Sub-Polar histogram applied to the edge image and after non-maxima suppression. The histogram (red) bins are scaled with square root for better visibility & the contrast of the image was reduced

relative to the vanishing point so it overlaps with the gabor filter. We use the following metric to transform the filter output to edges with

$$b'_0 = \begin{cases} \lfloor min(|b_l|, |b_r|) - b_0 \beta h_i \rfloor & \text{if} \quad (h_l < 0 \wedge b_r > 0) \vee \\ & \qquad (h_l > 0 \wedge b_r < 0) \\ 0 & \text{else} \end{cases}$$

β as a factor to suppress noise from the edge detection. In practice we use $\beta = 10$ to let only strong edges through and this is applicable to all pixels.

The edges b'_0 from all vanishing points, both global and local except for those sharing the axis with the global one, can be extracted and stored as separate images $\mathcal{E}' = (q_1, ..., q_n)$ for n vanishing points. In order to avoid the ambiguous vanishing point membership effect (see fig. 3.16 on page 47 and fig. 3.22) we subtract the images from each other. This is done by generating a counterpart image p_i for each q_i containing the maximum value *per pixel* from all other images except qi. The new image $p'_i = \lfloor p_i - q_i \rfloor$ is used for further processing.

The actual lines are extracted in two steps. To begin, we use the same polar sub-

histogram ,has been introduced in the previous chapter. However, we do not use the angle bias, see fig. 3.23 and 3.20. The histogram is centered on the vanishing point, built using p'_i. Next we apply non-maxima suppression for each sub-histogram to identify possible lines: Let h_i be an element in the sub histogram at position i. Then the non-maxima suppression is seen with

$$h'_i = \begin{cases} h_i & \text{if} \quad h_{i-2} \leq h_{i-1} < h_i \wedge \\ & \quad\ \ h_{i+2} \leq h_{i+1} < h_i \\ 0 & \text{else} \end{cases}$$

We use suppression with the width of -2 to 2 to be more robust to noise.
The second step is to extract lines from the remaining non-zero histogram bins. First we do a search of maxima, seen as non-zero bins in the histogram, along with angles through all sub histogram to seek one constitutive line. We allow a tolerance of 0.2 degree degrees while identifying the bias seen in long lines compared to short ones. Experiments have shown that it is best to start with long constitutive lines since they are less sensitive to noise. If a constitutive line is found trough the sub-histograms we apply an edge hysteresis grouping in the style of the canny edge detector using two thresholds (see chapter 3.1). The first threshold is set to cover 20% (0.2) of the edges resulting in gaps, which are balanced by the hysteresis threshold (0.05). The linked edges are grouped into a line if they are at least 8 pixels long or if they are half of the kernel size of the Gabor filter. The pixels that have been used for edge detection are set to zero and the algorithm is repeated until no non-zero pixels are left. Although we subtract the vanishing point edge images p'_i from each other, small artifacts that are usually half size of the kernel can still form flat edges in highly textured areas. We keep the sum of the gradients from the edge hysteresis grouping per line.

The algorithm proceeds with an average runtime of $3s$ on the CPU for Microsoft's Kinect with three vanishing points and 1s for AIT Stereo Vision using **no** multi-threading. The limitation of our approach is the Gabor filtering with a 15x15 kernel on a SXGA image. In practice we use an OpenGL GLSL GPU accelerated filtering that uses pre-calculated Gabor filters and calculates the edge images p'_i within 5ms for Microsoft's Kinect and 2ms for Stereo Vision. Our method uses the texture units within the GPU to gain computational acceleration and is able to calculate four p'_i at the same time. The generation of histogram and non-maxima suppression runs with linear runtime, namely 12ms for Microsoft's Kinect and 8ms for AIT Stereo Vision.

The actual line extraction is the fastest part with less than 2ms for the grouping and 9ms for line extraction using Microsoft's Kinect (3ms and 2ms for AIT Stereo Vision).

3.5 Results

We evaluate two aspects of our approach: the accuracy of the vanishing point detection and line extraction. The evaluation for accuracy is done with our database using Microsoft's Kinect and AIT Stereo Vision (see chapter 2.2 on page 20) and the York Urban DB[6] [67]. Both databases provide ground truth for vanishing points while the York database also gives ground truth for line detection, since the labeling of the lines is entered manually. For reference to our own database, see chapter 2.2 on page 20. We choose 1024 random images with robot localization accuracy of at least 5cm translational error and of at least 0.5 degrees angular error.
The York database contains both indoor (45) and outdoor (57) images of a Manhattan-like structure of buildings, corridors and rooms. The images were taken with a calibrated Panasonic Lumix DMC-LC80 instead of a robot. The database contains the major three vanishing points in a strict orthogonal layout similar to our method and is represented as three vanishing points. The overall precision of the vanishing points is not stated in the database, but seems to be less than 2 degrees (estimated using the ground truth lines as input for the vanishing point estimation) for the unconstrained vanishing points. The orthogonal vanishing points have to be converted from the unconstrained ones, so we use these as ground truth here.

3.5.1 Accuracy

First we must consider the accuracy of detecting the vanishing points by using the angular error to that matches the best with that of the ground truth data. The vanishing points were detected on 95% of all images. Note: we do not measure the error of unfound vanishing points and only use the global Manhattan system geometry. We evaluate four possible cases of our approach using the permutation of the components such as the raw output after refinement, the optimizer, and the raw output of the MSAC vanishing point estimator. The raw output with no further filtering will be referred to as "No Refinement" and used as a base for the other three outputs. These include "with Refinement", using plane intersections (see fig. 3.15 on page 46), "with

[6] http://www.elderlab.yorku.ca/YorkUrbanDB/

Optimizers" using only Hill climbing optimizers (see fig. 3.17 on page 48) and the combination of both as "Refinement and Optimizers".

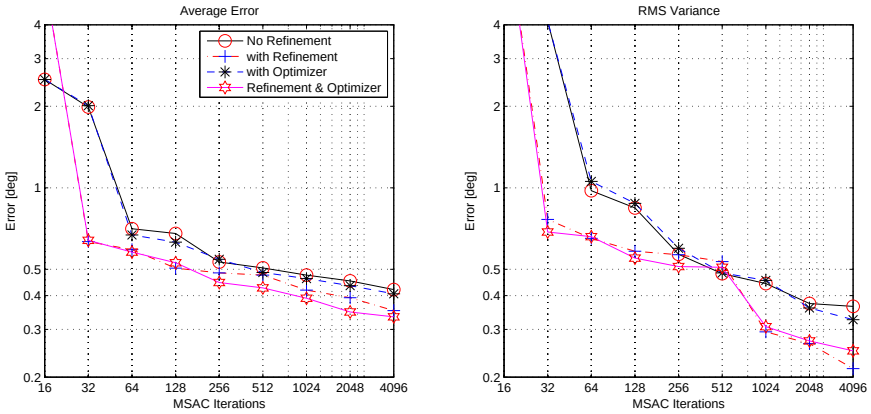
Figure 3.24 shows the average error and mean square root variance (RMS) plot using a various amount of iterations in MSAC of the raw vanishing point detection. We use a logarithmic scale for better visibility. Microsoft's Kinect, AIT Stereo Vision and the York Urban database show a similar trend for the average error and variance. For instance, the error converges asymptotically with an increasing number of iterations. The raw vanishing point detection shows with all three sensors the highest error, the combined one with both refinement, and the optimizer with the lowest error that shows the best results. The AIT Stereo Vision displays the lowest error because it has the largest field of view and best in-image dynamic range of all sensors. The lowest error here is 0.334 degrees for 4096 iterations (0.427 degrees for 512 iterations). Microsoft's Kinect shows the second best error with 0.468 degrees / 4096 (0.64degrees / 512). One interesting observation is that the variance improves when using more MSAC iterations compared with the average error seen in Microsoft's Kinect and AIT Stereo Vision. This is because of the nature of our database; about 20% images do not contain a clearly visible vanishing point due to a quite cluttered environment, with only 10% of the lines belonging to the true vanishing point. The last one is the York Urban database with 0.6915 degrees / 4096 (0.824 degrees / 512). One reason is the mixture images: some with no clutter at all and some (about 30%) with a high amount of clutter. The clutter appears in both indoor[7] and outdoor[8] image sets, so we do not consider these[9] separately. Another reason is that we notice when the images are poorly distorted from the calibration.

Next we can discuss the influence of the refinement and optimizer on the accuracy. As stated previously, the combination of both demonstrates the best results with our used sensors. The improvement of accuracy depends primarily upon the environment in case only the refinement or optimizer is used alone. For instance, the performance of the optimizer strongly depends upon the initialization. Considering the error of Microsoft's Kinect and AIT Stereo Vision we can see that the optimizers improve the

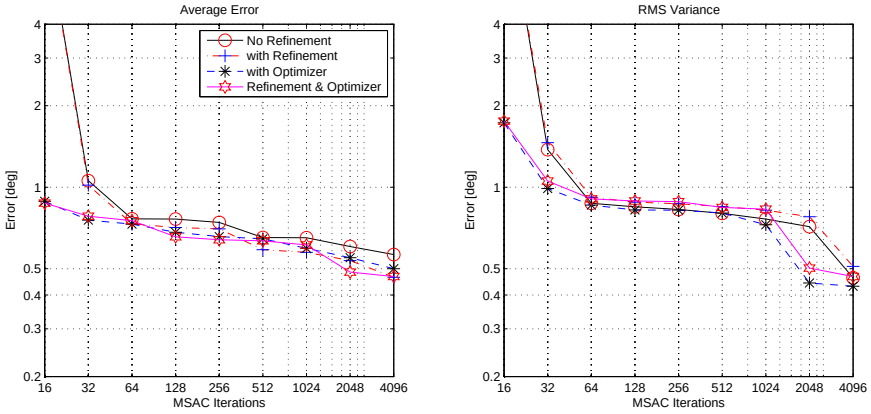
[7] clutter (ordered in the number of appearance) people sitting, people walking, plants, bicycles, fire extinguisher, trashcans, etc

[8] clutter (ordered in the number of appearance) trees, bushes, people walking, cars, motorbikes etc.

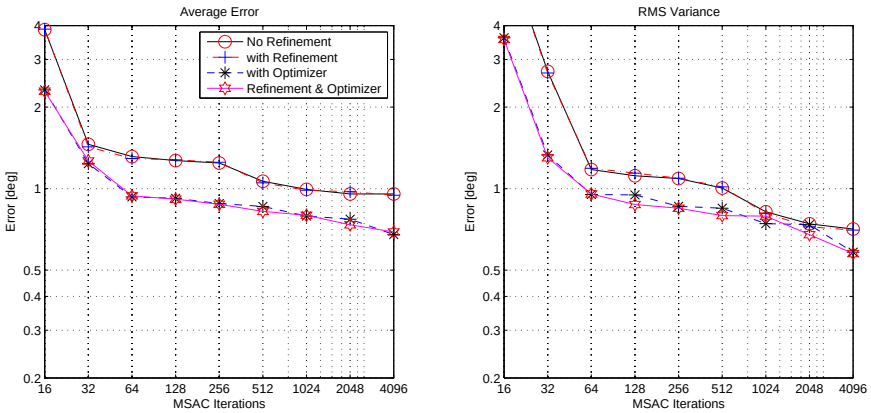
[9] experiments have shown that our approach works better on outdoor scenes because of the higher contrast seen in images.



(a) AIT Stereo Vision



(b) Microsoft's Kinect



(c) York Urban DB

Figure 3.24: Evaluation of vanishing point detection accuracy using our and the York Urban database. All plots use logarithmic scale for both axes.

error far less than the refinement. This is again due to the nature of the used database; the environment is made from large structures resulting in several, long lines (46 in total, with 32 long ones). It is also important to notice that the walls in the environment are white and do not contain any structure. Since the optimizer uses gradients the output is only slightly better because the input is almost identical to the lines. As result, the optimizer can become stuck in local minima instead of at the real vanishing point. The MSAC estimation is often a compromise for all vanishing points since it estimates an orthogonal system and is only relaxed by refinement and optimizers. The refinement shows a significant improvement in performance compared to using an optimizer alone. This is easily visible in the average error of fig. 3.24(a).

Figure 3.24(c) shows a contrasting case. Here the optimizer outruns the refinement if they are used separately. The images of the York Urban database contain a relatively high number of lines per image, such as 200, since many of these images were taken from far away. The maximum distance to an object with our own database is less than 5m due to the narrow environment. The lines show also a higher variation in relative angle to the vanishing point. This results in a smooth gradient map, which is optimal for an optimizer similar to fig. 3.17 on page 48. Since the refinement is based on intersections of planes, seen with lines on the 2D image, it does not show a major improvement due to the high number of intersections. With our database there are around 32 intersections per vanishing point, and about 300 seen in the York Urban database. The large amount of short lines from various angles leads to a Gaussian distribution around the MSAC estimated vanishing[10] point and not the true one.

Now we also consider the influence of the number of lines, along with the planes on the Gaussian sphere, on the accuracy of our vanishing point detection using our MSAC estimator, refinement, and optimizer, see fig. 3.25. The lines have been randomly chosen 100 times per image and the error was averaged. One can see that the average error and variance converges asymptotically with the increasing number of lines for Microsoft's Kinect, AIT Stereo Vision and York Urban database. AIT Stereo Vision converges here with 16 lines faster than Microsoft's Kinect with 32 lines because of the large field of view. Both show a similar trend in error and variance since they use the same environment. The York Urban database converges after 128 iterations and improves significantly using all of the lines. This is due to the relatively higher

[10] due to that the MSAC estimator already produces compromised vanishing points

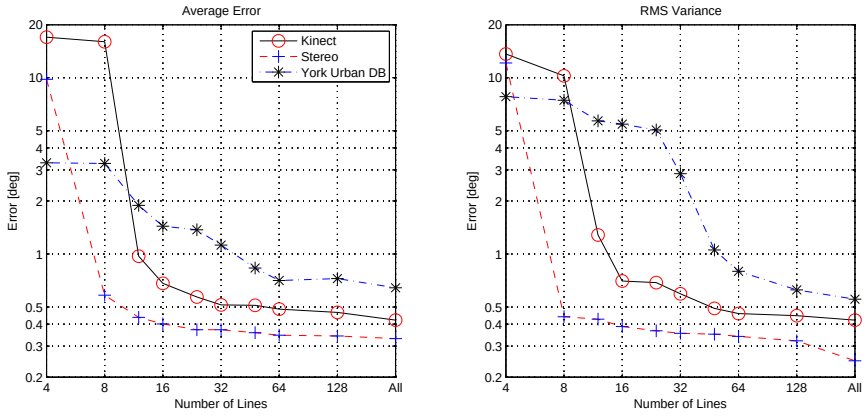
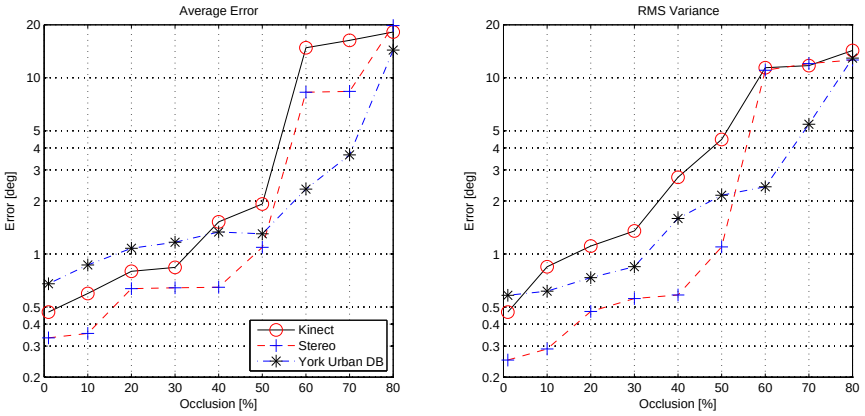


Figure 3.25: Evaluation of vanishing point detection accuracy using different numbers of random lines. The plots use logarithmic scale for both axes

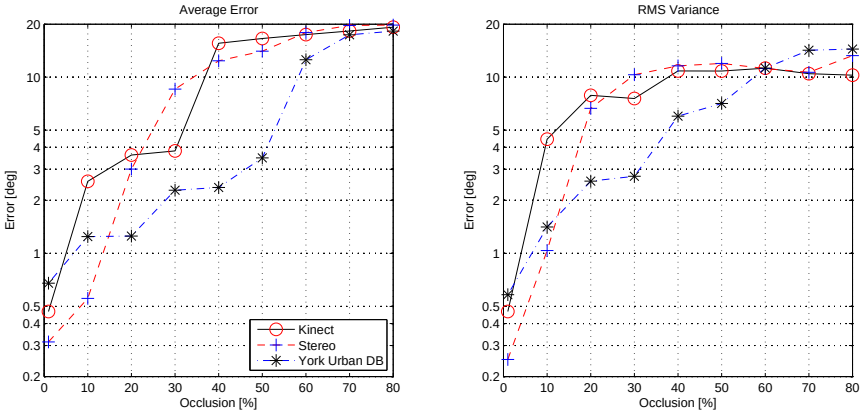
number of lines in contrast to Microsoft's Kinect and AIT Stereo Vision. For instance, there are approximately 80 lines per vanishing point with the York database and 12 for Microsoft's Kinect and AIT Stereo Vision. Although these lines are chosen randomly, many lines in the image do not belong to a vanishing point and lead to a loss of accuracy since the real vanishing point has not been estimated. The ratio of the relationship of the line to a vanishing point to the non-related lines is on average 1:3 for the York Urban database and 1:1 for Microsoft's Kinect and AIT Stereo vision. In some cases using fewer lines with the York Urban database allows only one true vanishing point to be found; however, this is rejected[11] because several lines are found for the other two vanishing points. In order to demonstrate less error with the lines, we do not use the minimum number of lines to planes as constrained per vanishing point. Note we do not use the "minimum number of lines/planes" constrained per vanishing point in this particular evaluation in order to show the error with few lines

After that, we also consider the stability of our approach regarding occlusion, see fig. 3.26. The goal is to use two masks to occlude certain parts, for instance 20%, in the image that are not used for processing with our method. The two types of masks that we use include. The first one we labeled as "small," since it occludes smaller, non-connected parts, and the other mask occludes only connected parts, so we have labeled as "big", see fig. 2.10 on page 23 for details. Both masks are identical for each image and are generated for the amount of occlusion. The York Urban database performs very similar on both types of masks since the images contain a high number

[11] False vanishing points are found instead in many cases



(a) "Small" Occlusion Mask



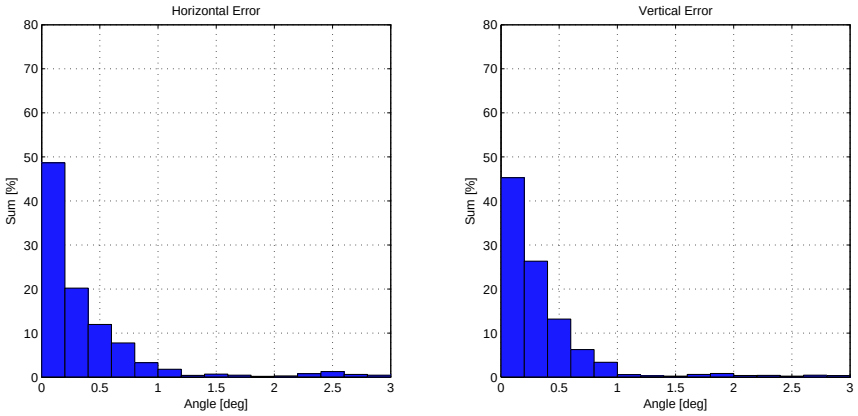
(b) "Big" Occlusion Mask

Figure 3.26: Evaluation of vanishing point detection accuracy using two different kinds of artificial occlusion. The "small" mask blocks non-connected parts of the image using a filled circles. The "big" mask blocks connected areas. These plots use logarithmic scale for the error axis

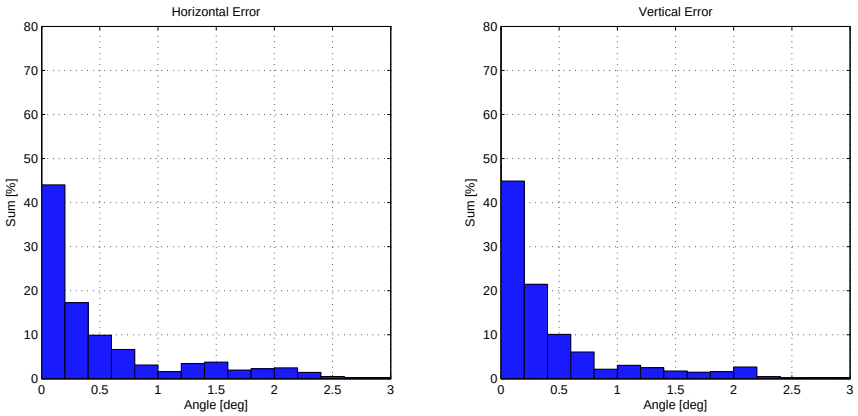
of short lines. An occlusion of up to 50% still delivers adequate results. Microsoft's Kinect and AIT Stereo Vision display a similar trend for both masks while AIT Stereo Vision performs slightly better. The "small" occlusion mask only has influence of up to 40% since we use a mean shift based on the grouping of straight lines, small gaps can be compensated. With 50% and above, we face the problem that the line segments are too short and will be rejected by the mean shift grouping. Another issue is that lines are not as likely to be grouped together[12], if the gaps are too big[13]. This is also the case for the "big" occlusion mask except it already blocks after 20% occlusion. Since the mask blocks connected parts in the image, the lines appear as broken far earlier.

[12] Resulting in many short lines

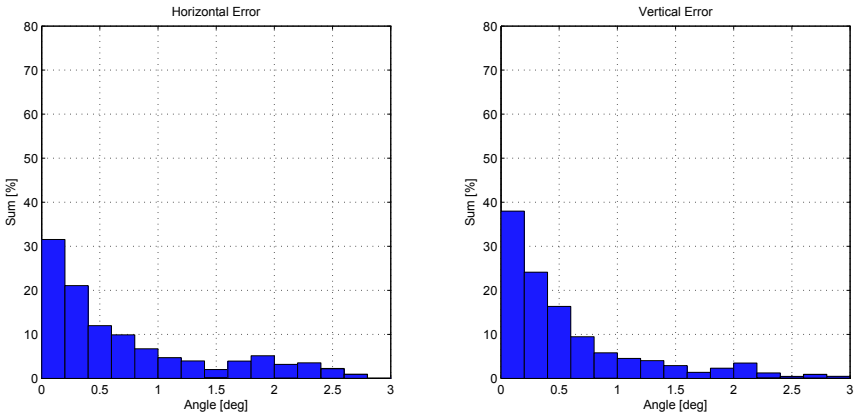
[13] We only allow a 2 degree tolerance for line detection and Mean shift grouping



(a) AIT Stereo Vision (Cumulative error seen on the plot on the left $\approx 95.5\%$, right $\approx 95.8\%$)



(b) Microsoft's Kinect (Cumulative error seen on the plot on the left $\approx 95.4\%$, right $\approx 95.5\%$)



(c) York Urban database (Cumulative error seen on the plot on the left $\approx 91.2\%$, right $\approx 93.0\%$)

Figure 3.27: Histogram of the angular error in reference to AIT Stereo Vision, Microsoft's Kinect and to York database

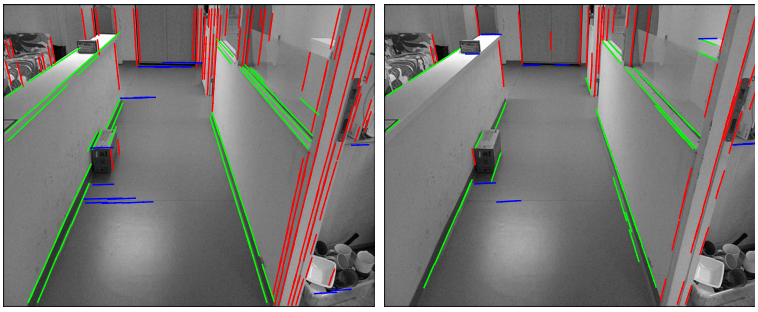
Before we discuss line extraction, we must also consider the overall performance of our approach. We can view this as a histogram demonstrating the error for AIT Stereo Vision, Microsoft's Kinect and the York Urban database. This is seen with the vertical (y axis) and horizontal axis (X/Z axis) combined. AIT Stereo Vision performs best overall because of its wider field of view, followed by Microsoft's Kinect, and lastly York Urban database (see above). The York database shows some artifacts on the plot due to the relatively small number of sample images.

3.5.2 Extracted Lines

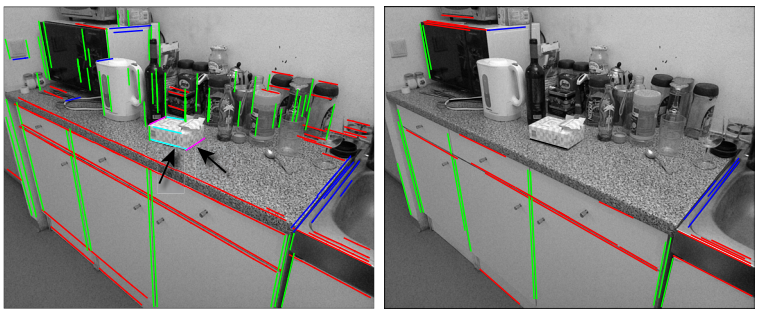
First we need to consider the line extraction with our database using Microsoft's Kinect. The line extraction from AIT Stereo Vision is similar, but results in fewer lines because of the lower image resolution. Figure 3.28 shows the results of our method compared to the unconstrained canny edge filter using the same contrast enhancement. One can see that our method extracts more straight lines in low contrast areas than the canny edge filter. For instance, this can be seen with the wall on the left in fig. 3.28(a) or with the door in fig. 3.28(d). This is mainly due to the Gabor filter that is oriented toward the vanishing point, and is visible in fig. 3.20 as well as through the use of histograms. Figure 3.28(b) and 3.28(d) show a global and a local Manhattan system geometry. The local system is shown in cyan and magenta such as the sugar box and the almost-open door. Some plausible lines are detected on the couch in figure 3.28(c) because of the pattern. This scene is known as a bent Manhattan system geometry, where the two walls are both aligned within a 90 degree angle, see fig. 2.8 on page 20.

Our approach shows a slight impact on textured areas like the kitchen plate since it cannot be filtered out with our ambiguous vanishing point membership suppression method, see fig. 3.16 on page 47. In fact, these incorrectly detected lines are half the length of the Gabor filter, making them quite short, and they are rejected if shorter than the 10 pixels seen with Microsoft's Kinect (3/4 Gabor filter size) or 5 pixels seen with AIT Stereo Vision. Since the results from AIT Stereo Vision and Microsoft's Kinect are similar, we only show Microsoft's Kinect here.

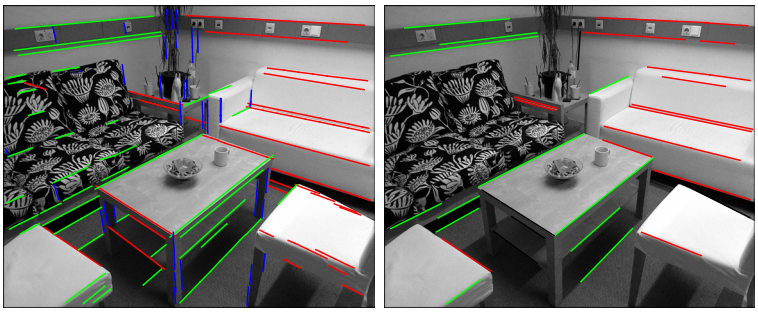
In order to consider the results from the York Urban database, see fig. 3.29 for a sample gallery. One can see that most of the lines that are oriented toward the Manhattan



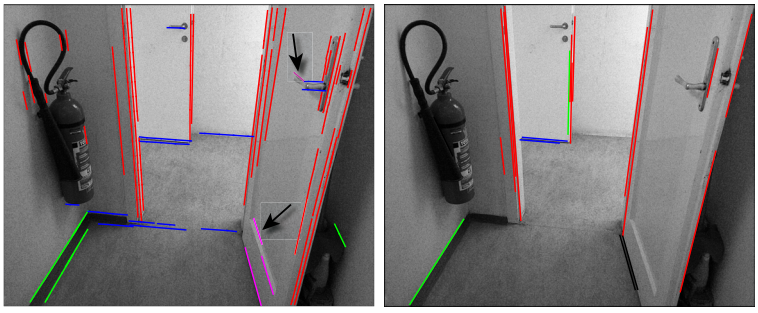
(a) Corridor



(b) Kitchen (5 vanishing points)

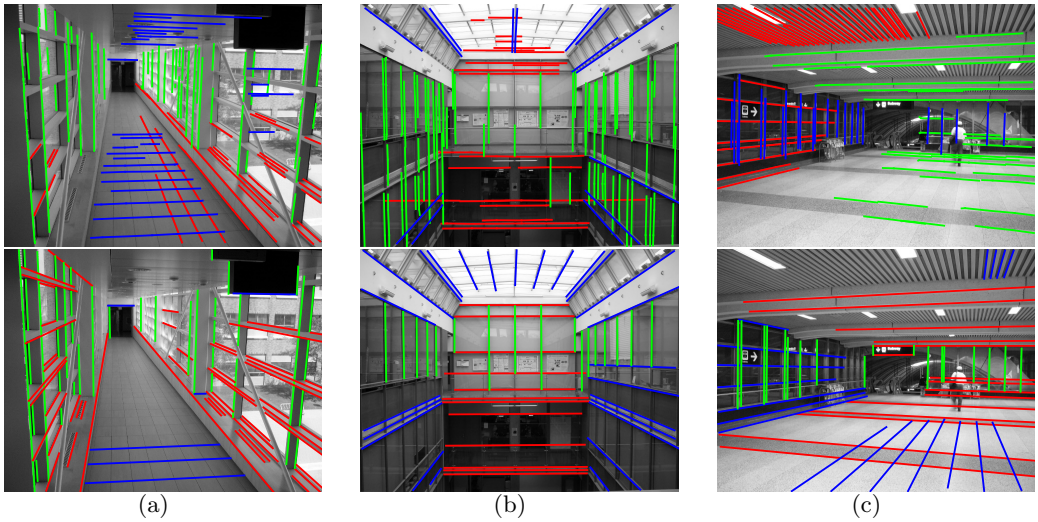


(c) Living Room

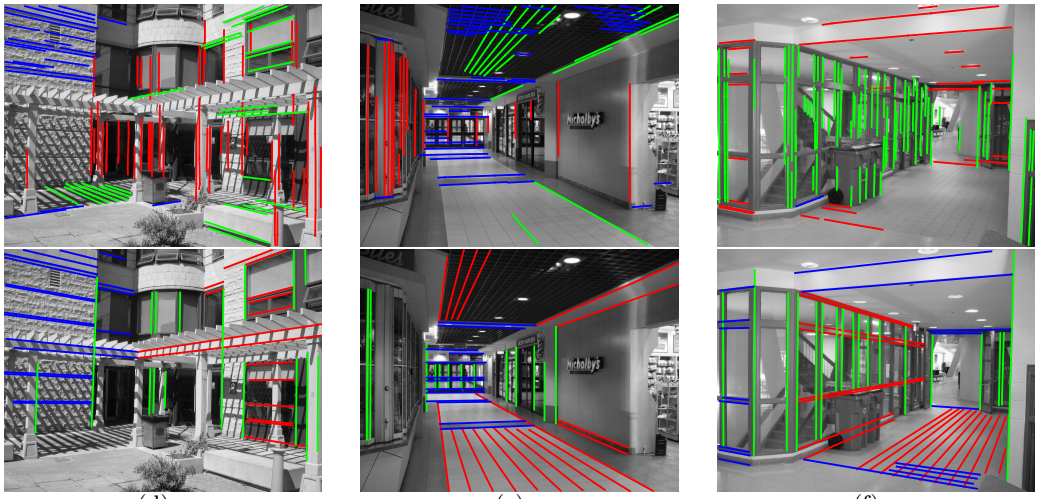


(d) Restroom (4 vanishing points)

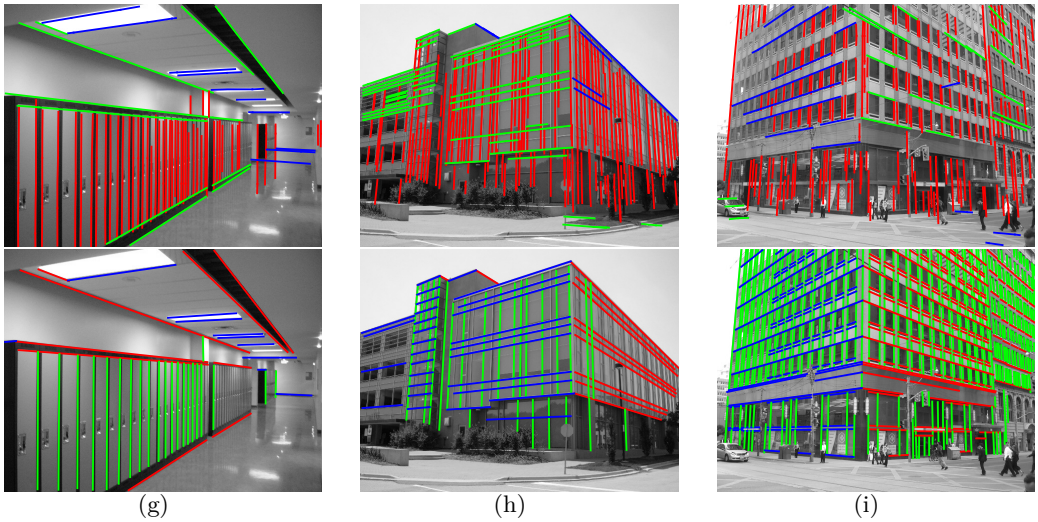
Figure 3.28: Extracted Lines from Microsoft's Kinect using our method (left) and standard Canny edges (right) with the straight lines filter, see chapter 3.2. All lines are at least 20 pixels long. Note 3.28(d) & 3.28(b) detected multiple vanishing points (see black arrows)



(a) (b) (c)



(d) (e) (f)



(g) (h) (i)

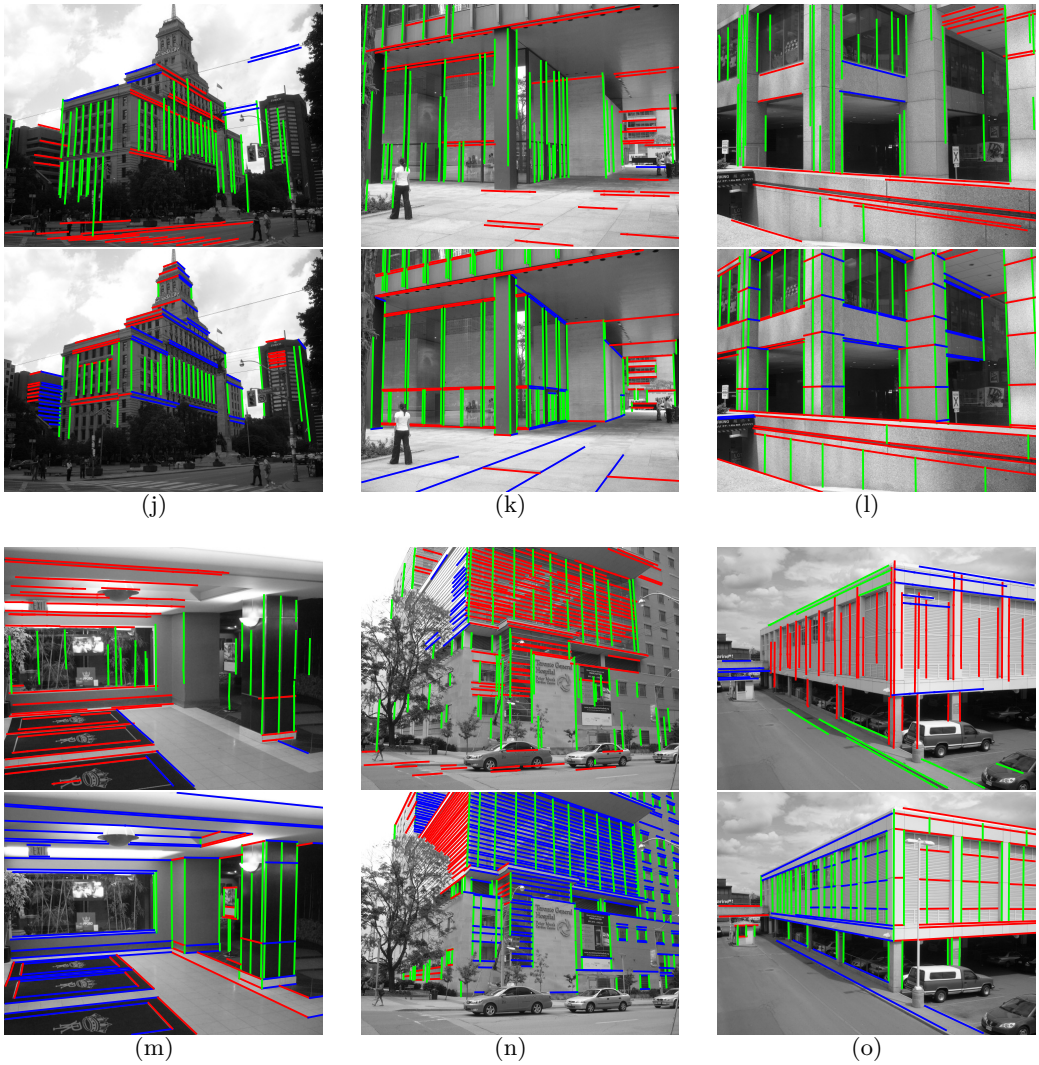


Figure 3.29: Extracted Lines from the York Urban database using our method (top) and provided ground truth. Note that the ground truth was labeled by hand and do not cover all lines in the images and that the colors in both images are not consistent.

system geometry and have been extracted; however, some structures were missed, like with fig. 3.29(k) and 3.29(l) where one vanishing point is seen at an incorrect position (fig. 3.29(k)) so that there are no extracted lines seen. In figure 3.29(l) we see that some lines are missing due to the ambiguous vanishing point membership suppression. Some lines are missing (left front wall) in fig. 3.29(n) because the pattern on the wall is almost the same size as the Gabor filter kernel. One can see also some plausible false positives on the street and trees. Some other false positive are visible Minor issues such as the reflection of structures in windows (see 3.29(b)) or on the ground (see 3.29(g) are seen.This is also sometimes seen with shadows if the light source is 90 degreesin

relation to the surface, e.g. 3.29(d). One drawback to our approach is that lines tend to overshoot the real edge[14], because we use Gabor kernel instead of a Sobel kernel e.g. fig. 3.29(g) or 3.29(f). On the other hand, our approach is able to extract edges with an extremely low amount of contrast, such as those in floor tiles in fig. 3.29(a) or those seen in the structure of the ceiling 3.29(e).

3.6 Related Work

The problem of vanishing point estimation has been studied in computer vision and robotics literature since 1970. Many approaches use edges with known orientation, linked-edges, or straight line segments for the estimation. In many cases there are two main aspects of vanishing point estimation. These include the estimator itself as well as the feature space representation, and are also known as planar images such as the surface or Gaussian sphere. Another aspect is the extraction of the Manhattan system geometry configuration system and other features such as structures that are oriented to a vanishing point.

One early concept that was used for vanishing point detection is Expectation Maximization EM [68]: The goal of EM is to give edges a known orientation [32, 69] or line segments [18, 70] using EM for a previously known number of vanishing points. One key issue with EM-based methods is the initialization since it can become stuck in local minima. As Nieto and Salgado [70] states the performance depends mainly upon the amount of visible Manhattan system geometry within the image, such as the amount of occlusion. The original approach from Kosecka and Zhang [32] estimated a loose set of vanishing points with no orthogonal constraints using only edges. The approach was extended by Schindler and Dellaert [69]: They introduced an "unknown" term so that an edge can belong to an unknown non-Manhattan system geometry structure. Flint et al. [18], Denis et al. [67], Nieto and Salgado [70] use line-segments as features instead of edges. Both approaches by Denis et al. [67] and Nieto and Salgado [70] are quite similar while [70] uses Gaussian sphere and [67] uses the planar surface. Flint et al. [18] expands upon these approaches by using a rigid Manhattan system geometry system, like our method, instead of grouping vanishing points in a post-processing as seen with previous approaches. First a rough Manhattan system geometry configuration is estimated using EM with a bias toward vertical structures. Next the configuration is

[14] half size of the Gabor filter kernel

relaxed within very narrow limits and vertical lines are extended using a guided line search, also similar to our approach.

Random Consensus sampling methods RANSAC [57] have also been used for vanishing point detection. One approach seen with Schaffalitzky and Zisserman [71] works in two steps and detects rigid Manhattan system geometry systems. Vanishing points are estimated by intersecting line segments on a planar surface. Then, in the second step, the vanishing points are randomly grouped into Manhattan system geometry configuration candidates. The systems are validated by testing them to co-planar constraints such as if they show a rectangular structure within the Manhattan system geometry. This approach was later revisited in the book by Hartley and Zisserman [50] using MSAC estimators, while Mirzaei and Roumeliotis [72] improved the results using a least-square-fit style solver. Another technique is proposed by Bazin and Pollefeys [60] by directly constructing valid Manhattan system geometry configurations on a Gaussian sphere using three lines similar to our approach.

A recent method used in the computer vision literature for vanishing point estimation is the *J-Linkage* approach originally proposed by Toldo and Fusiello [73]. The approach uses the same idea of random sampling like RANSAC [57] by maintaining a set of minimal samples. Each sample denotes a tentative model and represents a possible solution of the estimator. The fitness of a model to the dataset is calculated per item[15] and stored in a fitness map. The goal is to cluster two sample models with the biggest overlap in the fitness map and to group them if the overlap is within a certain threshold. The approach is repeated until models can no longer be grouped. This method allows us to estimate an unknown number of multiple models. Tardif [59] uses J-Linkage with linked-edges on a planar surface as quasi-line segments. The location of the vanishing point is the estimated model itself. The most plausible Manhattan system geometry configuration, within limits, is then created from the vanishing point estimates. Zhong et al. [74] builds upon this approach by biasing the vertical aligned structures within the Manhattan system geometry. The approach uses a similar strategy to ours: first a rough configuration is estimated using quasi-line segments while they are refined in a second step. Lines that belong to the same vanishing point are grouped together if they have the same orientation. In the final step, vertical line segments underneath grouped lines are extracted. Other clustering techniques are also used for vanishing

[15] One item can belong to multiple models

point detection, such as Sinha et al. [75] who use mean shift clustering on the Gaussian sphere.

Another method for estimating vanishing points is Hough Transformation (see Bormann et al. [76] for details): Li et al. [77] use 1D Hough Transform for two kinds of vanishing points: 1) within the image and 2) ones that are outside the image frame. Each edge of the image is converted into an infinite line in the Hough Space. Vanishing points within the image are obtained using first a 1D histogram and finding the horizontal location with the most intersecting lines at one point. The vertical position is obtained with a second vertical histogram. The method for the second case is similar but utilizes a polar representation rather than the Cartesian space like the first one. A histogram is built only on the angular part, then on the distance. While this method is not as precise as other methods like [59] it is still significantly faster than other approaches.

Some authors use hybrid methods to estimate the vanishing points, grouping line segments and the Manhattan system geometry configuration into one framework. The study from Barinova et al. [78] uses a graph-based representation that connects the vanishing points to lines and image pixels. A max-flow graph cut is used to obtain the vanishing points, which are then grouped into Manhattan system geometry configurations in post processing. Antunes and Barreto [79] also use a graph-based representation with only vanishing points and Manhattan system geometry systems as nodes directly instead of modeling explicit lines in the graph. The approach is able to detect multiple valid Manhattan system geometry. Hornacek and Maierhofer [80] use voting schema to group lines from multiple views directly for rigid Manhattan system geometry systems instead of grouping them first.

3.7 Discussion

In this chapter we presented two algorithms: one for estimating global and local Manhattan system geometry and one for line extraction with a previously known Manhattan system geometry. Both use a calibrated and undistorted 2D image as input only and can be also used for non-2.5d sensors like webcams. The Manhattan system geometry estimation is based on lines with robust MSAC estimators for a rough but appropriate estimate. In a later step we apply methods to improve the detected var-

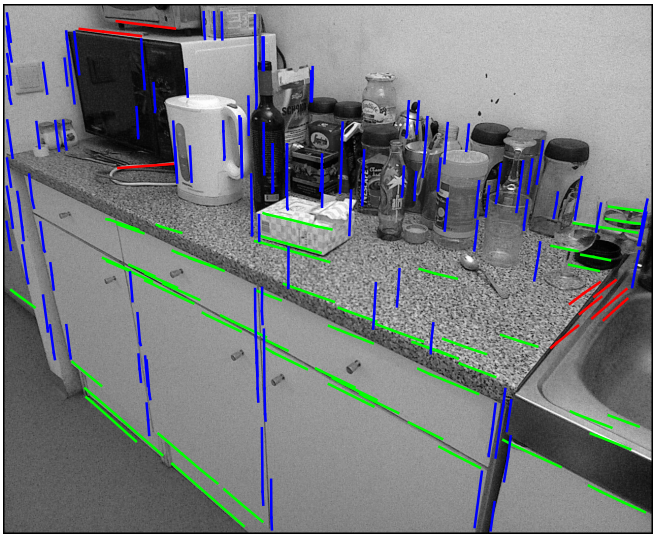
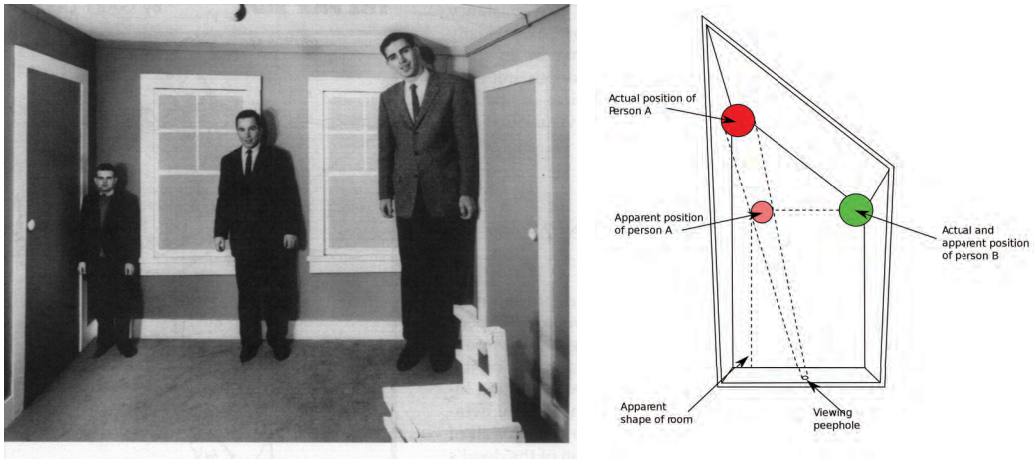


Figure 3.30: Line extraction with an incorrectly detected vanishing point (the bottom one is shifted too far to the left) using Microsoft's Kinect. Artifacts are visible on the vertical lines and are due to no proper ambiguous vanishing point membership suppression

nishing points such as the Manhattan system projected back in the 2D image. We use one method for improvement used for few but long lines (see fig. 3.15) and one for many but short lines (see fig. 3.23). The second method is the extraction of lines that are oriented to the Manhattan system geometry. We filter the image using a parameterized Gabor filter that rejects all structures that are not properly aligned. Histograms are then used to detect line candidates and are extracted using edge grouping with hysteresis. This allows us to extract strong lines that have low contrast.

The major limitation to our method is that it relies on the proper detection of straight lines for Manhattan system geometry estimation using a canny edge detector: It is only partially effective when combatting occlusion since long lines are needed for estimation, see fig. 3.26. Another issue is that our methods need at least 8 lines from two vanishing points that belong to the Manhattan system geometry despite occlusion. In certain cases, the robot is too close to walls and so only a few lines might be visible or the environment might not contain any visual edges. Another rare occasion occurs when lines might form an incorrect vanishing point that is only visible from a certain location. An example of this is the Ames optical illusion room [81], see fig. 3.31. At this point of our processing chain, we cannot validate a system since we detect it only from frame to frame and have no prior information of the motion of the robot. This is



(a) Front view, image taken from [81]

(b) Areal view from top

Figure 3.31: Ames optical illusion room: Objects in the room seem to have the wrong size depending upon their position in the room. This is due to the distortion present in the room itself, but it is not visible from the viewpoint of the observer

completed in a later step, namely sensor fusion in chapter 5.1 on page 122.
The limitation for the line extraction is that it relies upon a properly detected vanishing point in order to give accurate results. Figure 3.30 shows an example with three incorrectly detected vanishing points. Despite this, some lines have been extracted with the typical zig-zag pattern. Other artifacts are visible in the table because they have no ambiguous vanishing point membership suppression (see fig. 3.16 on page 47) for the real vanishing points. We want to emphasize that the line extraction uses only the estimated Manhattan system geometry from this chapter. In a later step we will apply a sensor fusion from both 2D and 3D sources to improve the overall performance, thus applying line extraction afterwards.

The advantage of our method is that we directly estimate a ridged Manhattan system geometry instead of estimating them separately as many other methods do [18, 21, 50, 59]. This results in high robustness to false positive systems. The use of a two-step strategy is also efficient such as first determining an overall estimate with strong metrics and later refining this with weaker ones. The use of hill climbing optimizers is efficient[16], but not optimal and can be replaced by using a quasi-Newton [61] or Levenberg−Marquardt algorithm [50]. Another advantage is the use of Gabor filters for edge detection, which provide a high toughness against noise. This also allows for

[16] since it is an optional step

the extraction of lines within low contrast structures as mentioned above. The use of histograms is a very stable method to visualize suppressed noise that could not be sorted out by the Gabor filter.

Table 3.1 gives an overview of the proposed algorithms and their properties. We want to emphasize that the algorithms are suitable to be used on a robot since they run with $O(nlogn)$ runtime. Algorithms like the Gabor filter and ambiguous vanishing point membership suppression have been implemented on the GPU for a faster processing time. The main limitation is the optional optimizer; if the most accurate results are needed, it can also be implemented on the GPU.

Table 3.1: Summarized properties of the introduced algorithms in this chapter with non-multithreading and non-optimized code on our test computer (see page 26). The typical runtime is given for Microsoft's Kinect and is on average 1.5× slower than AIT Stereo Vision

	Straight Line Detection & Mean shift line grouping	Vanishing Point Estimation	Refinement with planes intersection & MSAC	Histogram Optimizer with Hill climbing	Gabor Filter & Line Extraction
Accuracy	+	o	+	++	+
Robustness	+	++	++	+	++
Estimating multiple model		yes			
estimation method	mean shift	MSAC	MSAC	hill climbing	direct
parametric Model	yes	yes	yes	yes	yes
num model dimensions	2	3	3	3	2
Polynomial Complexity	O(m log n)	O(nm)	O(nm)	O(nm)	O(m log n)
typical n	640x480 or 1280x1024	50	50	640x480 or 1280x1024	640x480 or 1280x1024
typical m	48	512	32	64	64
typical runtime (ms)	6	11	1	300	17(GPU)

4 | 3D Sensor Processing

The use of 3D data for robotic applications has a long tradition within the community in various applications, e.g. navigation [6, 82], obstacle avoidance [6, 83], mapping [84, 85] and self-localization [86]. Traditional stereo vision is probably one of the types of sensors that have been used for the longest time for this purpose. It was originally made from matching spare features [50] due to limitation in computational power and camera quality. The result is the ever popular multi-view geometry community in computer vision e.g. [87–90]. A not so recent development is the dense stereo matching that is common today among embedded systems e.g. *Point Grey's Bumblebee* (1996) with 5fps in 320x240 pixel. Another type of popular sensor is tilts or rotates from laser scanner like the *Velodyne Lidar HDL64E*. These kind of sensors originates the use of

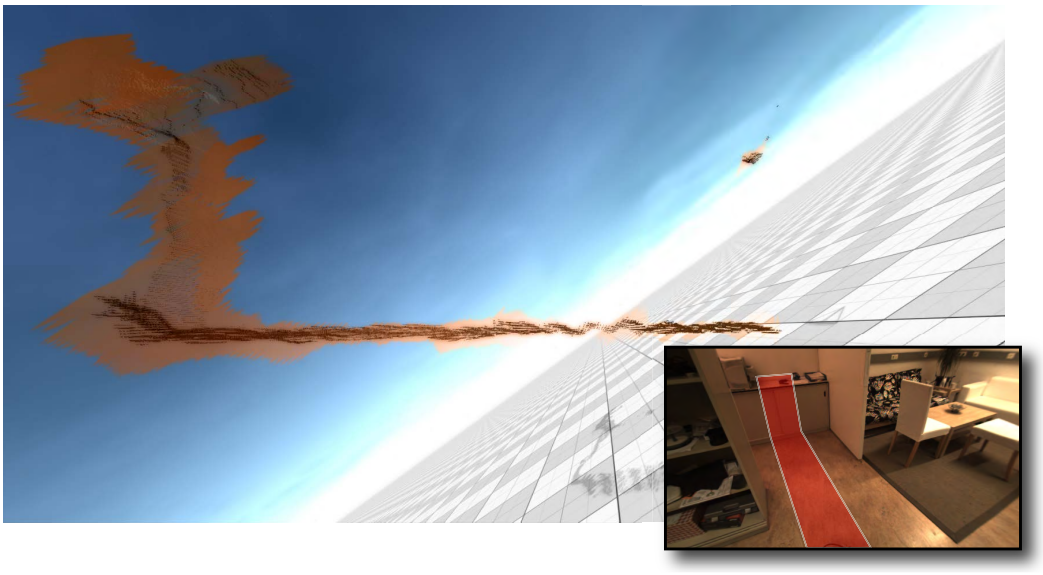
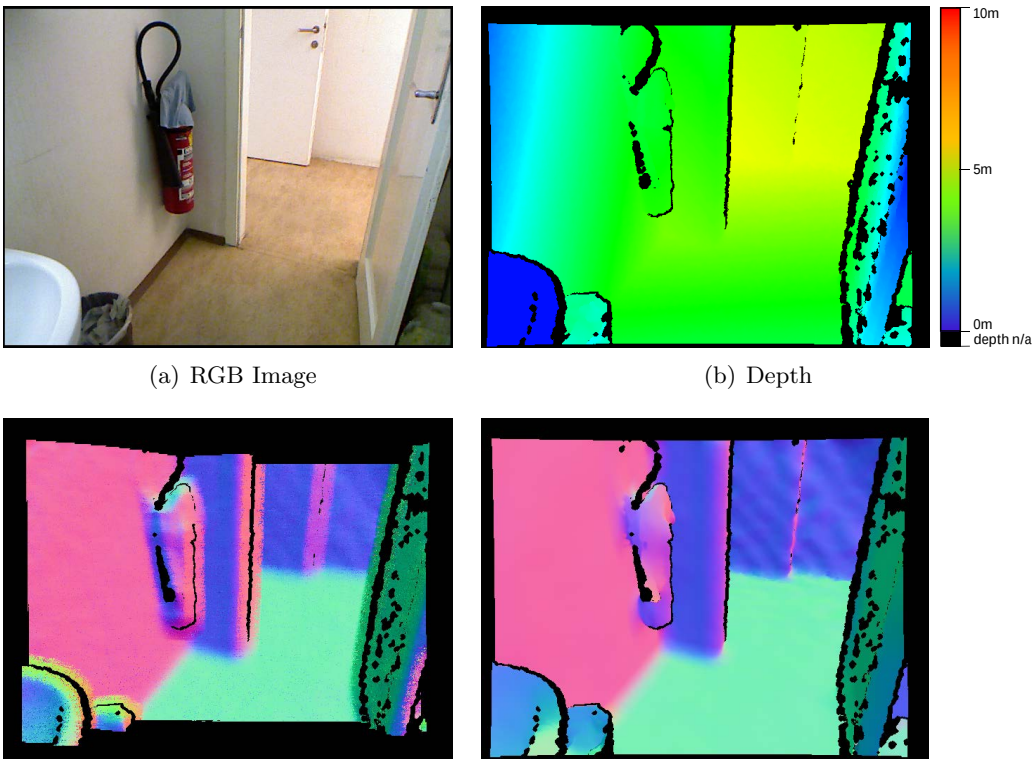


Figure 4.1: Visualization of Point clouds using AIT Stereo Vision from an indoor environment from the marked area. The cloud is shown from the side, the tilt from the camera (red) was removed. The transparent ellipsoids represent the uncertainty of the non-transparent point clouds

2D laser scanners that were originally intended for safety applications like the *Sick LMS-200*. Depending on the sensor, they deliver either a high resolution with a low frame rate (15s per scan, 360 x 360000 scan rays) like *Rigel VQ-180* or lower resolution (300-900fps, 64 x 4000 scans rays) with a high frame rate like the *HDL64E*. Before the public release of Microsoft's Kinect some years ago, two other sensors were quite popular for 3D sensing: The *SwissRanger SR-2* and later the *SR-3000*. The sensor uses the time-of-flight principle to estimate depth with 176x144 (SR-3000) pixels with a relative limited field of view compared to Microsoft's Kinect and laser scanners.

3D sensor processing with the so-called *point clouds* (i.e. voxels, see fig. 4.1) uses a common strategy for many applications within robotics and computer vision [91]: First, outliers are removed from the data set such as false positives caused by for example reflection or aliasing on corners. Then, local features per voxel are calculated like normal vectors and are used e.g. for parametric fitting of primitives like planes. As the name *point cloud* suggests, many algorithms, like RANSAC plane fitting for example, do not exploit the underlying ordered data structure of the sensor as is in the case of the grid voxel structure of the 2.5d sensor. Another aspect is that voxels are modeled as floating points with no explicit uncertainty handling for the sensor processing in many cases. The result is that only "very certain" data is used from the sensor. For instance, many approaches use Microsoft's Kinect data only within a short range (e.g. 1m) or use stereo data with a texture post-processing filter to obtain only certain results. Our experiments have shown that on average 20-35% of the data can be considered as certain ($\approx 5mm$ accuracy) from both Microsoft's Kinect and AIT Stereo Vision, resulting in that more than half of the depth data is not used (=very uncertain).

The idea behind the 3D sensor processing is similar to that of the 2D one: First, estimate the global and local Manhattan system geometry and extract features for the final image segmentation. The features are namely normal vectors, planes that are aligned with the Manhattan system geometry and wall hypothesis. Our approach with 3D sensor processing is that we explicitly model uncertainty with a novel confidence in depth perception metric and prior knowledge about the camera sensor characteristics. We also explicitly exploit the underlying grid structure of the native sensor space for extracting features. Figure 4.1 shows the motivation for this kind of processing in the example of AIT Stereo Vision: One can see that the ground (marked with a red rectangle) shows a wave like structure while the voxels are shown as ellipsoids, that



(a) RGB Image (b) Depth

(c) Integral Image (PCL), 100ms (d) kd-Trees 20cm (PCL), 4500ms

Figure 4.2: Comparison of different normal vector estimations methods. The color of the normal image reflects the direction of the normal vectors

is to say voxels with associated uncertainty. This is due to the sub pixel processing known as interpolation. Note that the error within the native sensor space is only within a few bits while it accumulates within the Euclidean space. While the ground is relatively certain, other areas like the cupboard (upper rectangle) are far less certain. Note that the error on the cupboard is about 4x times higher than on the ground, but accumulates far more in the Euclidean space which results in a larger uncertainty. One can see a ground-like structure on the ground and a wall-like structure for the walls within the uncertainties rather than only on the dots.

4.1 Normal Vector Estimation

The use of normal vectors has become a de facto standard feature in 3D sensing over the last decades [91–93]. The usual approach is to calculate a normal vector for each corresponding voxel in the data set. Depending on the type of sensor, this task can be computationally demanding, for instance if the data is not aligned in a grid-like

structure or if the data is noisy. In the case of 2.5D data, the data is a aligned in an image-like grid structure which enables us to calculate the normal vector in $O(n \log n)$ runtime. We use a variant of the Stefan Holzer method from the Point Cloud Library [91] to calculate normal vectors based on integral images; for example, the 2.5D depth is used as image. The use of integral images [94] allows calculation of an average sum of any rectangular size area with $O(1)$. The sum of the area is used for smoothing the depth data within a rectangle to avoid either noise or uncertain depth, see fig. 4.7(a) on page 84. Holzer's method uses a static rectangular size for all pixels in the 2.5d data. We extend this by using a dynamic rectangular window size depending on the distance to the centered pixel. We use 0.05% of all pixels for the rectangle in a range of $0.3m$ and a maximum of 2% for a distance $< 5m$ with linear interpolation.

Figure 4.2 shows a comparison of our normal vector estimation and the standard kd-Tree approach using the Point Cloud Library. One can see that the integral image method gives adequate results within planes with respect to normal vectors. The "noise" within these areas is produced by the limited precision of 32-bit float values in the integral image. The corners show artifacts due to that the estimation is done on a dynamic 2D window regardless of the distance of the voxels within the rectangle. Kd-trees calculate the real Euclidean distance of all voxels, but with significantly higher

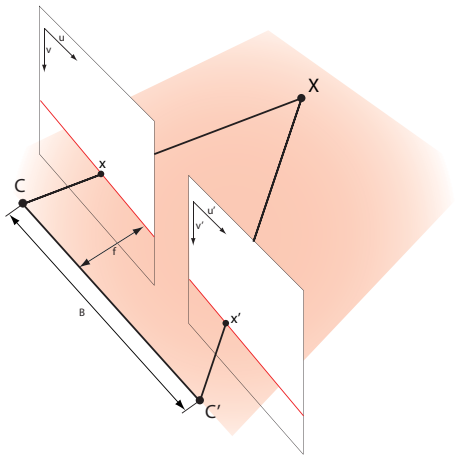


Figure 4.3: 3D Reconstruction of two parallel cameras using triangulation and epipolar geometry: Both left C and right C' camera observe the same object X on the red plane, represented as red line on the image space. The image points x and x' are back projected to rays on the plane from the camera center. The distance to X from each camera center can be calculated when the focal length f and baseline B are know

computation[1] times than integral images. Another effect is that the integral image method shows a margin around the outer boundaries. This is caused when the rectangle is partially out of the image at its borders. We do not repeat voxels at the borders to avoid artifacts. Kd-Trees are not used in this work because they are computational and far more expensive. The artifacts are filtered out by robust filter methods. Rusu et al. [91] suggest not using all voxels within the depth data and down sample them to a 5cm voxel grid within a specific range. The method is up to 30x faster than the normal unconstrained kd-Tree method depending on the depth distribution. We do not apply this, because we lose depth data that we need for later processing.

4.2 Modeling Uncertainty

One major challenge in 3D computer vision is how to cope with uncertainty. Depending on the type of sensor, the certainty of depth data depends for example on the distance to the object, texture or material properties. For instance, the uncertainty of a laser range scanner depends more on the material that reflects the laser beam than on the distance. Many laser sensors do not only report the distance but also the amount of back-refection. These two components can be used to model the certainty as a linear function, see Siegwart et al. [95] for more details.

Many 2.5d/3D sensors use the concept of *triangulation* [50] using two or more viewpoints for depth estimation. The AIT Stereo Vision and Microsoft's Kinect both use a straightforward case for triangulation; they use parallel cameras with *epipolar geometry* constraint, see fig. 4.3. Both cameras C, C' (left and right in fig. 4.3) are parallel, have the same focal length f and same height of the camera center. The principle depth estimation is as follows: One object X is observed in both viewpoints on the image plane x and x'. Using epipolar geometry (in the two camera cases a plane) both x and x' are projected as rays on the plane and meet at the object location X. The three lines of C, C' and X create a triangle called the plane epipolar plane. The depth Z to the object X is obtained using the *disparity* d principle [50] that is:

[1] 45x times slower on our test computer, see page 27

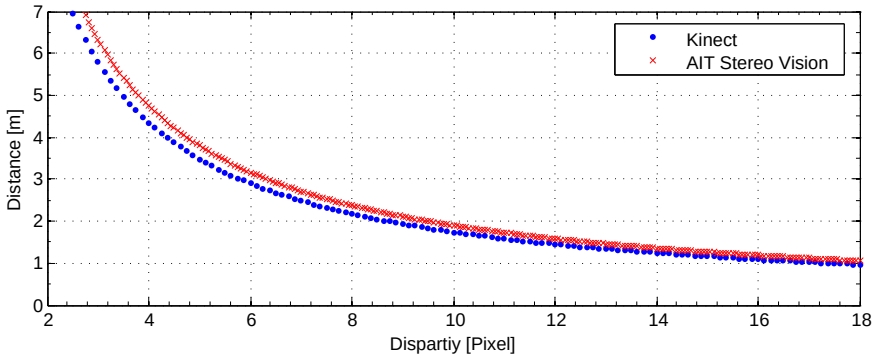


Figure 4.4: Mapping from (sub pixel) disparity to depth using AIT Stereo Vision and Microsoft's Kinect. Only every 4th disparity value is shown for better visibility

$$d = x - x' \tag{4.1}$$

$$= \frac{B \cdot f}{Z} \tag{4.2}$$

Disparity d is inversely proportional to the depth Z to the object. AIT Stereo Vision uses block matching between the left and right camera (red line), and chooses x and x' according to the most similar match. Microsoft's Kinect however uses one camera and IR-projector, but uses the vice versa principle as stereo vision; the pattern of the IR-projector is already known so the camera matches it in all possible disparity values. Since C and C' are cameras, x and x' are matched within the resolution of the cameras. Some sensors use interpolated sub-pixel accuracy, like 16 sub pixel for the AIT Stereo Vision and 8 pixels for the Microsoft's Kinect . Therefore, the estimated depth of both Microsoft's Kinect and AIT Stereo Vision is **not linear**, as shown on figure 4.4. Both sensors show a similar disparity-to-depth curve despite the different baselines and focal lengths of both sensors. The AIT Stereo Vision camera has in theory double the disparity resolution as the Microsoft's Kinect.

The uncertainty of the depth *reconstruction* results from three major factors [50]: 1) the quality of calibration[2], 2) the layout of the cameras and 3) the certainty of the match of the two points x, x'. Since we use black box sensors, we have little influence on the calibration and layout of the sensors. The effect is shown in figure 4.5. The setup on the left figure is ideal for a reconstruction with little uncertainty (assuming identical calibration quality on all three setups) due to that the cameras are not parallel

[2] i.e. low re-projection error (e.g. > 0.2 pixel), also depending in the quality of lens and CCD chips

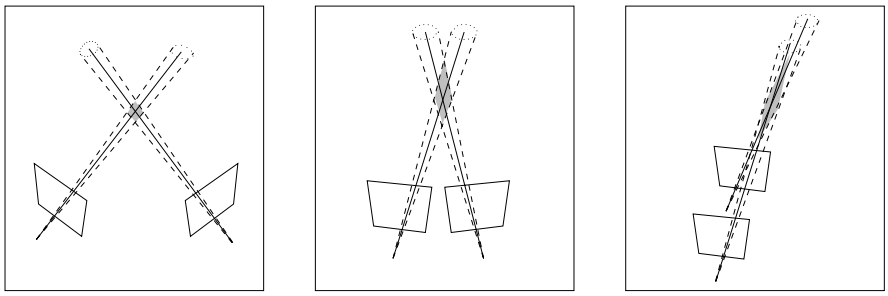


Figure 4.5: Examples for uncertainty of depth measurement depending on the camera configuration. The gray area depicts the uncertainty within the black lines and dotted lines. The black line corresponds to sub-pixel match x, x' on both cameras. The dotted line represents the uncertainty due to calibration of the setup. Picture taken from Picture taken from [50, pp 312]

and use different viewpoints. This kind of setup is quite popular within the multi-view reconstruction geometry [87–90] computer vision community. In many cases, only one camera is used and moved around within an environment. One part of the reconstruction is pose estimation of the camera relative to the environment. The center image represents the situation of the sensors using parallel cameras. It is a good trade-off of uncertainty and frame-rate depth data, because this means that depth data is immediately available. The right figure shows the case that is popular with the robotics and vision community for visual odometry [96–98] and structure from motion [18, 99]. In many cases one camera is used and moved within a planar (sometime known) motion. With robotics, a combination of the center or right case setup with robot motion is used to enable a quasi left case setup. One popular case for this is traditional map building using SLAM [100–102].

We model the depth uncertainty as an ellipsoid function of the disparity and the certainty of the match of the two points x, x' itself, see fig. 4.6. Using disparities allows us to work directly on the **native sensor space** of the 2.5d depth sensor. For the sake of simplification, we assume that disparity values are $d > 0 \in \mathbb{N}$. The idea is to model the uncertainty as an ellipsoid aligned to the focal center of the (left) camera using the 3D point as a center or mean value. The "thickness" of the ellipsoid reflects the size of on the image plane through the focal point. The length of ellipsoid is set to actual disparity values that have been mapped back to the Euclidean space. Please note that the disparities do not scale linearly in the Euclidean space (see fig. 4.4). This simplified approximated model is valid as long the 2.5d cameras have a field of view of 180 de-

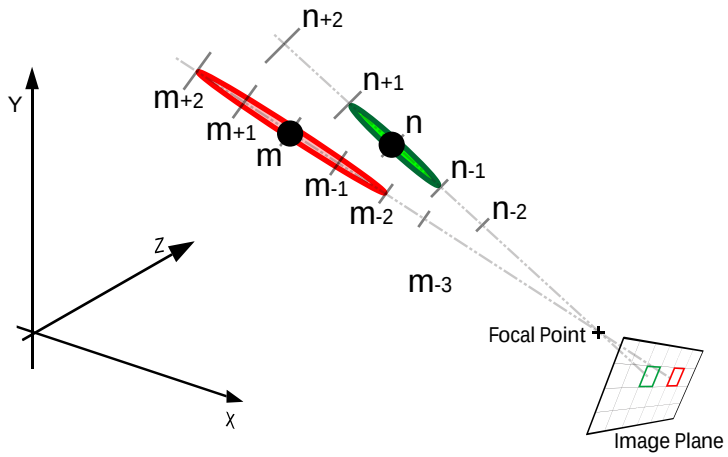


Figure 4.6: Modeling depth uncertainty using ellipsoids: The ellipsoids are aligned with the focal point and centered at the two 3D points (shown as black dots) at disparity values m and n. The length is within the disparity range (i.e. native sensor space) using a confidence in depth perception metric c_{onf}, see below. Here n is two disparities long, and m is four disparities long

grees and both cameras are aligned parallel to one another, see [50, pp. 322] for details.

The length within the disparity space is obtained using a c_{onf} metric that reflects the certainty of a depth match of x, x' (see fig. 4.3). Let conf max be the maximum disparity value of a sensor. The length plength of the ellipsoid of a 3D point p with conf as corresponding confidence in depth perception $0 < c_{onf} \leqq 1 \in \mathbb{R}$ metric is given using robust M-estimator [50]: $e \to 1 - \frac{c^2}{c^2+e^2}$ with

$$p_{length} = (1 - \frac{{c_{onf}}_m^2}{{c_{onf}}_m^2 + {c_{onf}}^2}) \cdot {c_{onf}}_{max} \tag{4.3}$$

assuming p_{length} is in disparity space and ${c_{onf}}_m$ as constant for mapping. We use ${c_{onf}}_m = 16$ for stereo vision and conf ${c_{onf}}_m = 8$ for Microsoft's Kinect . Since the Microsoft's Kinect and AIT Stereo vision use different techniques to obtain depth, we show the calculation confidence in depth perception metric individually. The usage of the M-estimator has the advantage of a bounded error function.

4.2.1 Microsoft's Kinect confidence in depth perception metric

Experiments with Microsoft's Kinect have shown that close objects, e.g. 2m, seem to be more certain than far objects, e.g. 4-8m. One reason for this is probably that the used technology behind Microsoft's Kinect where a projected dot pattern is used. Dots that

are far away are probably much darker and smaller than close ones. Microsoft's Kinect also tends to interpolate missing data points in the native sensor SXGA resolution and apply a median-filter like down sampling on the output VGA depth data. The idea is simple: those points that are not interpolated are more confident. We estimate those interpolated and filtered points by applying the heuristic directly on the native sensor space i.e. disparity.

The method is straightforward assuming that there are local homogeneous structures within the image: We calculate within a 3x3 window for each pixel $B(u,v)$ the min and max value and store the absolute difference of min-max in a new image. Let $\mathcal{K}_{uv} = \{B(u-1,v-1), B(u_{-1},v), ...\}$ be the surrounding pixel values within a 3x3 kernel cen- tered at u,v.

$$B(u,v)' \quad = \quad max(\mathcal{K}_{uv}) - min(\mathcal{K}_{uv}) \tag{4.4}$$

Note that the Microsoft's Kinect reports missing matches with the value 0 as disparity. We apply a 3x3 quasi Gaussian smoothing on all pixels $B(u,v)'$ to reduce noise. We use a kernel to multiply each element $\mathcal{K}'_{uv}$ of the kernel (list). The kernel is written 3x3 for better readability:

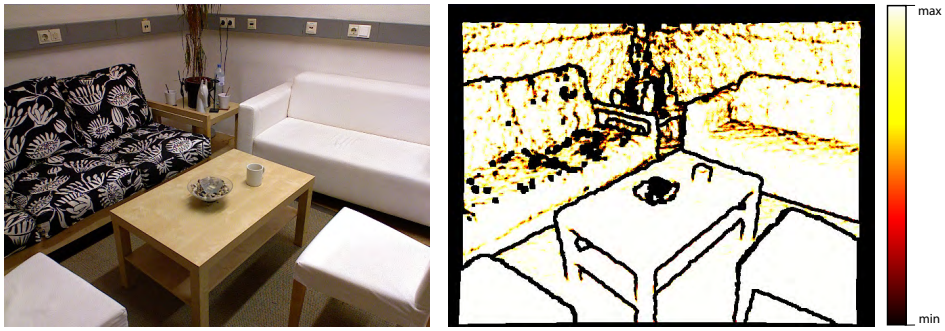
$$B(u,v)'' \quad = \quad \frac{1}{14} \cdot \left\{ \begin{array}{ccc} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{array} \right\} * k'_{uv} \tag{4.5}$$

In a last step we scale the values to $c_{conf_{uv}}$ using a normalized Gaussian function using η as normalizing constant

$$c_{conf_{uv}} \quad \propto \quad \eta \frac{1}{\sigma\sqrt{2\pi}} e^{-(B(u,v)'')^2/2\sigma^2} \tag{4.6}$$

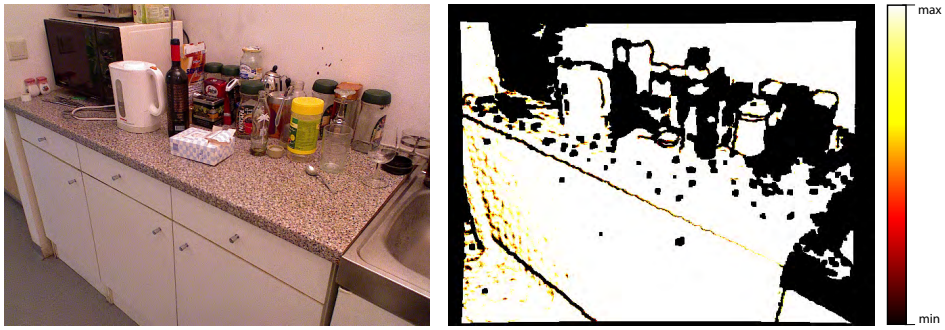
using $\eta \in \mathbb{R}$ as normalizing constant (so $B(u,v)'' = 0 \rightarrow 1$) with $\sigma = 12$. The result is that a disparity value of 12 disparities represents 32% certainty. A value of 24 would represent only 5% certainty.

Figure 4.7 shows the metric with our own indoor database. One can see that the confidence of depth does not scale linearly. While the near wall on the left is quite "confident" we see less confident areas on the distant wall due to the structured light

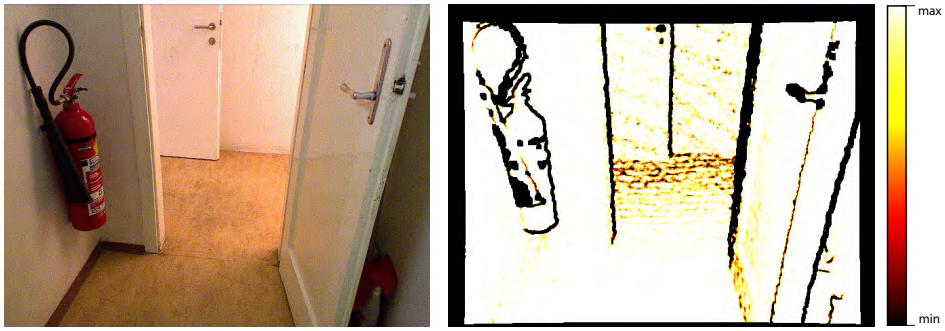


(a) Living Room

(b) Corridor

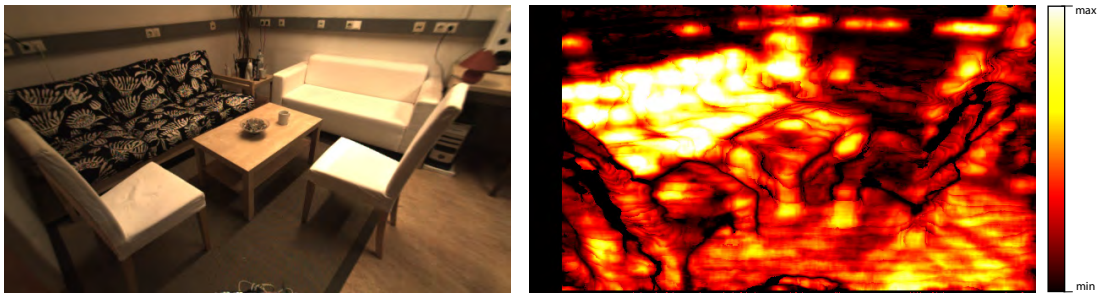


(c) Kitchen

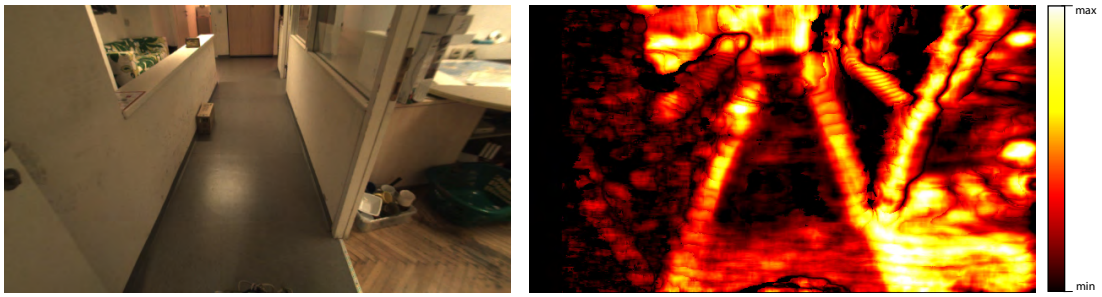


(d) Restroom

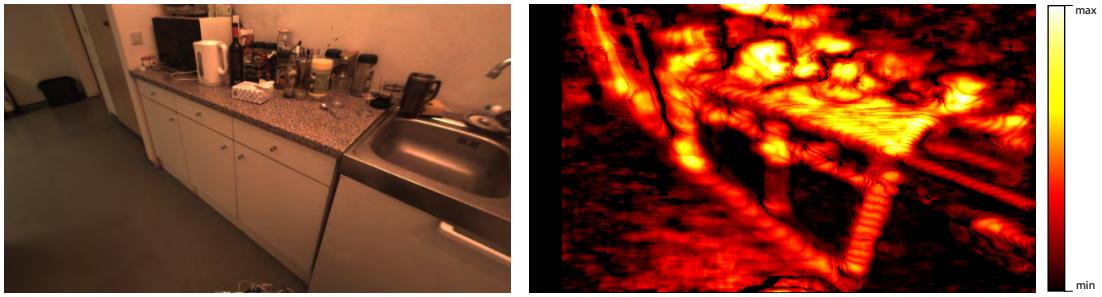
Figure 4.7: Sample confidence in depth perception metric images of our own the indoor database with Microsoft's Kinect



(a) Living Room



(b) Corridor



(c) Kitchen

(d) Restroom

Figure 4.8: Sample confidence in depth perception metric images of our own the indoor database with AIT Stereo Vision

sensor principle of Microsoft's Kinect. Another effect is that all depth edges are also not confident which is on purpose due to Microsoft's Kinect interpolating in these structures. We also see in figure 4.7(d) & 4.7(b) that the confidence depends on the

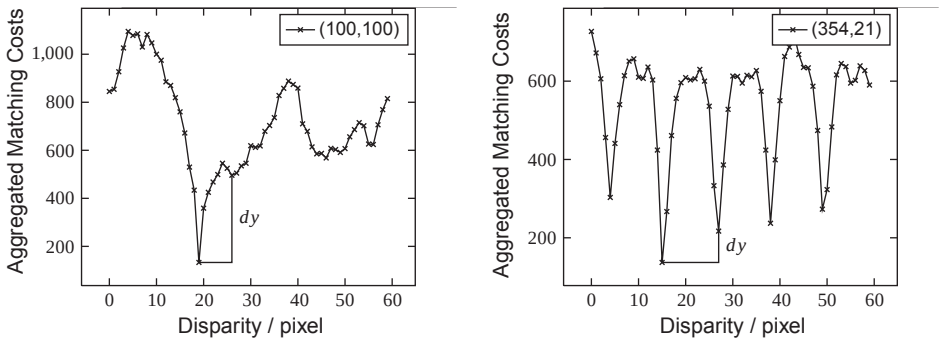
orientation of the planar structure to the camera.

While having local homogeneous structures is quite a strong assumption, it still holds true for many indoor cases with planar surfaces. It is sensitive to depth difference due to missing depth data or real depth jumps. This is on purpose since the Microsoft's Kinect seems to interpolate the most in these cases.

4.2.2 AIT Stereo Vision confidence in depth perception metric

The confidence in depth perception metric for stereo vision is directly obtained from the stereo matching process within the software [38, 103]: The idea is that a stereo match such as a x, x' (see fig. 4.3 on page 78) is more confident if it is clearly distinguishable from the 2nd best match, e.g. x, x''. AIT Stereo Vision uses an aggregated matching cost function using a Census transformation, see [103] for details. The lower the cost for matching the more likely it is to be the same observed structure within x, x'. Figure 4.9 gives an example as to how the confidence in depth perception metric is obtained resulting in low and high confidence values. Both curves show the likelihood of a match using a cost function. The lowest cost in the winner-takes-it-all manner is used as match for x (within the disparity range), if the value is lower than a certain threshold. The confidence is obtained by calculating the difference between the best match cost and the 2nd one within all disparities, here d_y. The confidence in depth perception metric in the software is obtained with

$$c_{conf_{uv}} \quad \propto \quad 1 - max(1, 4\frac{d_y}{d_{max}}) \tag{4.7}$$



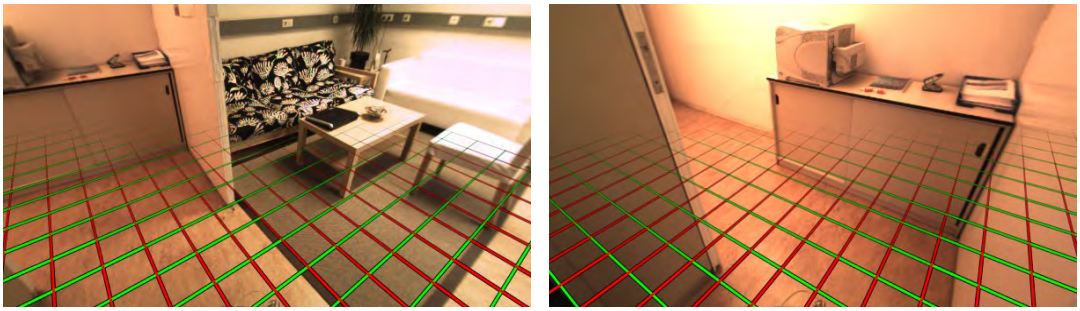
(a) high confidence (b) low confidence

Figure 4.9: Calculation of c_{conf} AIT Stereometric in the AIT Stereo Vision disparity matching, plots taken from [103, pp 1186]

with d_{max} as the maximum possible cost value. The confidence is also used as an additional threshold for validating a possible x, x' tuple. Figure 4.8 shows example images with AIT Stereo Vision. One can see that the confidence in depth perception metric is quite different from Microsoft's Kinect one (see fig. 4.7). This is because stereo vision is passive and depends on texture rather than emitting structured light. It is visible that not only textured areas are quite certain (e.g. the left couch in fig. 4.8(a)), but also shadows and cluttered areas (see fig. 4.8(c)).

4.3 Manhattan System Estimation

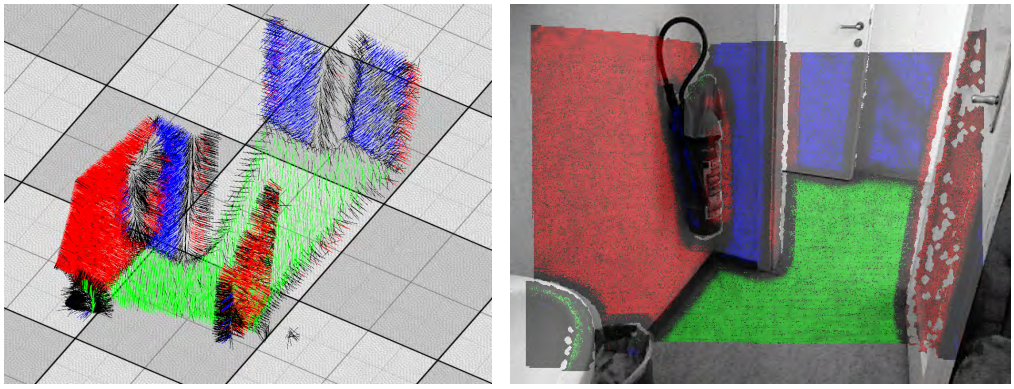
The estimation of the Manhattan system geometry is an essential step in our approach. The idea is to first estimate the *global* dominant system and then find local ones. The estimation process is different from the 2D-based one while the robust estimation techniques are similar as is the case for RANSAC. Here we only use depth data and ignore properties of the 2D image like color and texture. These components are later combined in the sensor fusion process, see chap. 5.1 on page 122. We present two methods to estimate the Manhattan system geometry: 1) a feature-based estimation and 2) an estimation based on histograms. The feature-based estimation uses normal vectors in combination with robust RANSAC/MSAC filters to estimate global and local systems. The histogram-based method uses particle filters with histogram to estimate the global system using minimum entropy as a metric.



(a) Living Room (b) Corridor

Figure 4.10: Sample of an estimated global Manhattan system geometry from AIT Stereo Vision using Minimum Entropy Estimation. The system has been back-projected on the ground as grid

The feature-based method emphasizes the estimation of multiple systems but depends on the quality of normal vectors. The histogram-based method emphasizes robustness



(a) Normal Vectors with Manhattan system assignment

(b) Grayscale Image with Manhattan system assignment

Figure 4.11: Estimated Manhattan System of a sample scene. The colors indicate the membership to one of the three major axis

with the explicit modeling of uncertainty rather than on accuracy, see fig 4.10 and page 20 for an overview of the environment. We do not show here Microsoft's Kinect and the feature-based method, because they look the same as this one.

4.3.1 MSAC Manhattan System Estimation

The estimation approach works as follows and is visualized in fig. 4.11: First, we estimate the global Manhattan system geometry: the roll, pitch and yaw of the observer relative to the Manhattan system geometry system. The Manhattan system geometry system is technically nothing else than the rotation of the camera relative to the normal axis. We use normal vectors (see chapter 4.1 on page 77) and robust estimators with the handling of outliers. The normal vectors (from the voxels) that are plausible to an estimated Manhattan geometry are marked and assigned to one of the three normal axes of X, Y, and Z. The remaining non-marked voxels are used to estimate local[3] systems in conjunction with already marked ones. In a last step, the system is refined using statistical methods and the variance is calculated.

We propose a novel approach [26] using a variant of the well-known Random Sample Consensus (RANSAC) [57] algorithm for robust estimation with the handling of outliers. RANSAC based methods obtain their estimates by randomly selecting coefficients from a given dataset to a known model. The estimation is validated by applying the model (with estimated coefficients) to the data set and counting how many points

[3] the assumption is that a local system shares one axis of the global system

$\mathcal{P}$ (=inliers) support the model within a certain threshold $t \in \mathbb{R}$ with error to model $e \in \mathbb{R}$. If $t > e$, the point is an inlier, otherwise it is an outlier. The estimation is iterative; in each iteration a new model is estimated and the number of inliers is counted. After a fixed number of iterations k is met, the model with the most inliers (=support) is used as estimate. The number of iterations $k \in \mathbb{N}$ to find a valid estimate depends on the ratio of outliers to inliers. That is

$$k = \frac{\log(1-p)}{\log(1-(1-\epsilon)^s)} \tag{4.8}$$

with s as the number of sample points that are needed to estimate the model coefficients and $p \in \mathbb{R}$ as probability that the algorithm selects at least once only inliers from the input data set. $\epsilon \in \mathbb{R}$ is the probability of choosing an inlier each time a single point is selected. In many cases $p = 0.99$ is chosen as golden standard, see [50] for more details. A variant of the standard RANSAC algorithm that we use for the Manhattan system geometry estimation is the M-Estimator Sample Consensus (MSAC) [50]. Instead of counting inliers within a specific threshold for a model with the most support, we accumulate the error of the model from the original data. Here, the model with the least error is chosen. The error function C for a model with $i \in \mathbb{N}$ points $\mathcal{P} = \{p_1, ..., p_n\}$ is given as

$$\mathrm{C} \quad = \quad \sum_i \delta(p_i) \text{ with } \delta(e_i) = \begin{cases} e^2 & e^2 < t^2 \quad \text{inlier} \\ t_{max}^2 & e^2 \geq t^2 \quad \text{outlier} \end{cases} \tag{4.9}$$

with $t_{max} > t$ as constant maximum error.

The idea is to describe the Manhattan system as a model of three orthogonal vectors $\vec{\mathsf{N}}_1, \vec{\mathsf{N}}_2, \vec{\mathsf{N}}_3$ one for each axis. Since we do not know which axis will be estimated first, we name them with numbers rather than with x,y,z. The vectors to express the *orientation* to an axis i.e. the normal vector is virtually aimed in both directions of the axis. To estimate the global system, we use a model that uses 3 sample points from the dataset to construct Manhattan system with always orthogonal vectors. The idea is simple: first we select a random vector to obtain the first axis and find the "roll" of the vector using two more points to construct a plane, as is shown in fig. 4.12. Let $\vec{A}$ random vector of the dataset and set it to the first axis

$$\vec{\mathsf{N}}_1 \quad = \quad \vec{A} \tag{4.10}$$

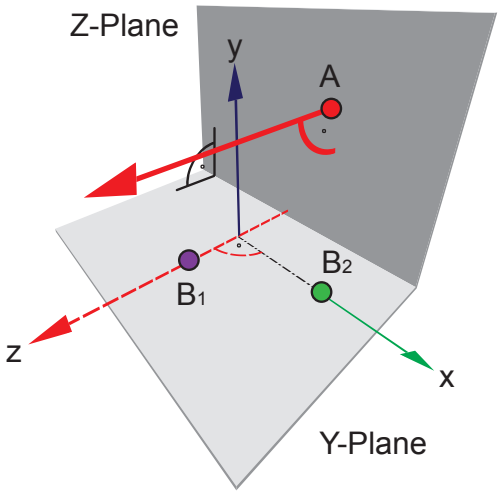


Figure 4.12: Estimating the Manhattan configuration using three normal vectors and MSAC: A is used as a seed for the Manhattan system for the first axis. B_1, B_2 are used to calculate the "roll" of the second axis and the third axis is redundant. Here we assume that A and B_1, B_2 do not share the same orientation plane of the Manhattan system, but that B_1, B_2 does

The random vector is drawn from a previously sorted list according to the vector's corresponding voxel's confidence in depth perception c_{onf}, see chapter 4. The list has been sorted with an integer[4] *bucket sort* [104] in linear runtime from high to low confidence values. The drawing itself is biased with a Gaussian weighting ($\sigma = 0.5$) to ensure high confidence vectors are more often chosen than low ones. The overall assumption is that $\vec{A}$ is a point on a Manhattan-like structure e.g. on the "Z-Plane". Next we calculate the relative rotation from $\vec{N}_1$ to a orthogonal plane that corresponds to a different Manhattan system in the same system, for instance the "Y-Plane" or "X-Plane". This is constructed with two helper points B_1 and B_2 assuming they are from the same plane

$$\vec{\mathsf{N}}_2 \quad = \quad \vec{B}_2 + \vec{a}((\vec{B}_1 - \vec{B}_2) \cdot \vec{a}) - \vec{B}_1 \tag{4.11}$$

The second vector is obtained by shifting the first vector to B_1 and using B_2 as the "roll" component. Note that we do not check in advance if B_1 and B_2 are on the same plane, since we have no prior information about planes in the 2.5D data at this step. Such "implausible configurations" usually generate a Manhattan system with a significantly less inliers/support than a plausible configuration like in figure 4.12. Since

[4] the c_{onf} values have been mapped to 1024 integer values

the Manhattan system is redundant to one axis, we only need to calculate two axes. This is elucidated in figure 4.12 where the x axis is redundant. The third vector is the cross product of both previous vectors:

$$\vec{N}_3 \quad = \quad \vec{N}_1 \times \vec{N}_2 \tag{4.12}$$

This approach always generates a valid Manhattan system using three vectors. The estimation of the global system is done using MSAC at a threshold of $t = 5$ degrees, $t_{max} = 50$ degrees and $\epsilon = 0.3$ assuming that only 30%[5] of the dataset corresponds to the global Manhattan system. The use of the significantly larger $t_{max} >> t$ raises the probability that the MASC will favor the dominant system. The error metric is carried out by comparing the relative angle (dot product) of each vector of the set with all three vectors. The smallest angle is chosen, resulting in an angle always between $0-90$ degrees, since an axis does not have an orientation like a normal vector. Figure have an orientation like a normal vector. Figure 4.11(a) shows an example using color coding for the Manhattan axis membership and using color saturation as the relative error to the axis.

After the global Manhattan system geometry with the most support has been estimated, we apply some heuristics to test basic plausibility. First, we count per axis vector the number of inliers (within the threshold) in the style of RANSAC regardless of the point's confidence in depth perception metric c_{onf} value. If the vector with the maximum number of inliers represents at least 30% of the data as well as 10% of the median[6] inliers, we assume the estimate as plausible. Note that tracking is done in another part. In practice, the maximum and median number of inliers are similar (e.g. both 30%) in cases like corridors or hallways, like in figure 4.11(b). We do not use the number of inliers from the remaining axis for cases where only one wall and the ground is visible. This is the case when the robot from figure 4.11(b) leaves the restrooms and rotates on the spot in front of the wall. Experiments have shown that in these cases only structures from two Manhattan axes are visible. Note that this is only the case in $< 1\%$ of our test dataset.

The estimation of the local Manhattan systems is straightforward: First, all inliers from the global system are marked and removed from the set. Then, the same process

[5] Resulting in $k = 169$ iterations
[6] the axis with the second most inlieres

is repeated from the global estimation except that $\vec{\mathsf{N}}_1$ is initialized with a random axis from the global one. We use the same thresholds from the global one except for the basic plausibility test: Since the global system was removed from the set, we only count the maximum number of inliers, at least 3% from the total dataset. Experiments have shown that usually no local systems are found since they are too small in the image, like small boxes on the table. In many cases, open doors or huge boxes are found as a local system, if they are not aligned with the global Manhattan system. In some cases, it can happen that a door or other structure is not 100% aligned to the global Manhattan system. This leads to local estimated systems that are almost identical to global ones. This is due to we remove points of a estimated structure. In some cases "ghost points" remain due to we use strict threshold of of 5 degrees for the estimation and point removal. Therefore some remaining points lead to the estimation of the same global system. This can be solved by using a RANSAC variant that estimates multiple systems in one step e.g. MultiRANSAC [105] or by rejecting systems that are similar to the global one within 1 degrees. We do not use a MultiRANSAC in our approach since we explicitly estimate local systems that share at least one axis with the global one and do not look for independent ones.

In the last step, we apply a refinement on all estimated (global and local) systems in order to relax the estimated systems. In many cases, walls are not completely straight due to issues during building or material fatigue over time. The relaxing is done individually per axis and is similar to the well-*known expectation maximization* [68] algorithm. The idea is to optimize iteratively the axis membership of points to the given dataset. This is necessary since MSAC estimates a rigid model based on three sample points rather than on an averaged model. Note that with many RANSAC/MSAC based methods, a least square fit of the data is used as an average model. Our method is based on iterative *mean shift* tracking [58, 106] using the Gaussian unit sphere. We calculate the mean value $\mathsf{N}' \in \mathbb{R},\mathbb{R}$ of all angles $n \in \mathbb{R},\mathbb{R}$ on the unit sphere per axis N (from $\vec{N}$) using a Gaussian kernel δ so

$$\mathsf{N}' = \eta \sum_i \delta(n_i, \mathsf{N}_\mu, \sigma) \tag{4.13}$$

$$\delta(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \tag{4.14}$$

with η as normalizing constant. The kernel is centered on the axis at μ on the Gaussian unit sphere withs $\sigma = \frac{t}{3}$ (3σ represents 99.99% uncertainty). We calculate the variance

N'_μ with

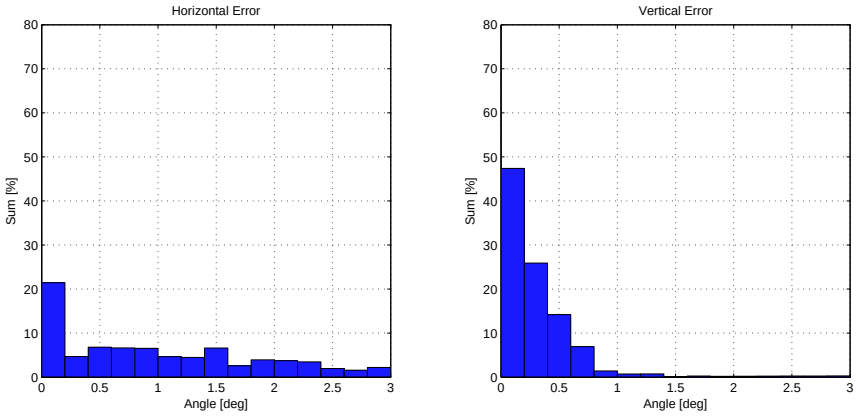
$$\mathsf{N}'^2_\sigma = \frac{\sum_i \delta(n_i, \mathsf{N}_\mu, \sigma)(N' - n_i)}{\sum_i \delta(n_i, \mathsf{N}_\mu, \sigma)} \tag{4.15}$$

The shift from N to N' is called *mean shift*. We apply the mean shift iteratively and and use the calculated variance as Gaussian weighting in each iteration. We terminate the iterative process if one of the following conditions is met: 1) if the difference of $|\mathsf{N} - \mathsf{N}'|$ after a iteration is less than a threshold (i.e. 0.1 degrees), 2) if more than 15 iterations are done, 3) if the N' is more than t (5 degrees) away from the initialized center or 4) if the number of points within the kernel is less than 5% of the entire dataset. In the last step, we set the new axis value N' to N and set the last variance to N_σs for each axis. The variance is later used in the fusion of Manhattan system geometry.

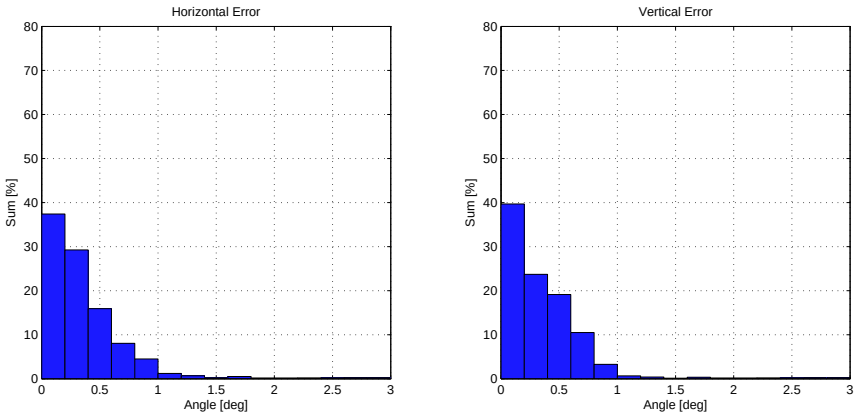
Our code runs on our test computer (see page 27) at 55ms for AIT Stereo Vision with **no** multi-threading. For Microsoft's Kinect, we need 90ms using all data. The computation on Microsoft's Kinect needs more time since it is far more dense than stereo vision in many cases. The evaluation of global MSAC Manhattan probes consume 50% of the overall time, the local system uses 10%, and the rest is used during the refinement.

Figure 4.13 shows the average error of both AIT Stereo Vision and Microsoft's Kinect using our own benchmark data (see page 20) as histogram. The error for the vertical and horizontal axis is shown separately and uses the same scale for all histograms within this thesis. Both sensors perform very similarly on the vertical axis although AIT Stereo Vision performs slightly better than Microsoft's Kinect. This is due to the larger field of view of the sensor and its larger range. During experimentation, we noticed that AIT Stereo Vision usually detects large parts of the ground in contrast to Microsoft's Kinect, as is see in figures 2.7 and 2.4 on page 18. We noticed that, in many cases, textured grounds e.g. fig. 2.4 and even untextured plain-colored grounds e.g. fig. 2.7 result in valid depth data. Mostly, an untextured wall does not result in valid data in comparison to the ground counterpart. The ground is in many cases significantly more dirty than the walls since people walk on it with dirty shoes[7] resulting in a pseudo-texture. Since the robot moves around the dirt, the walls turn invisible due

[7] Please note that in our lab, most people use slippers and not street shoes



(a) AIT Stereo Vision (Cumulative error seen on the plot on the left $\approx 62.1\%$, right $\approx 94.9\%$)

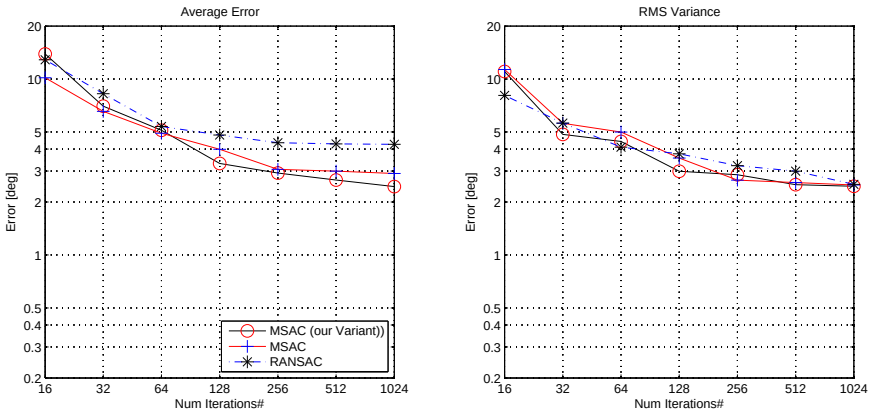


(b) Microsoft's Kinect (Cumulative error seen on the plot on the left $\approx 96.2\%$, right $\approx 97.9\%$)

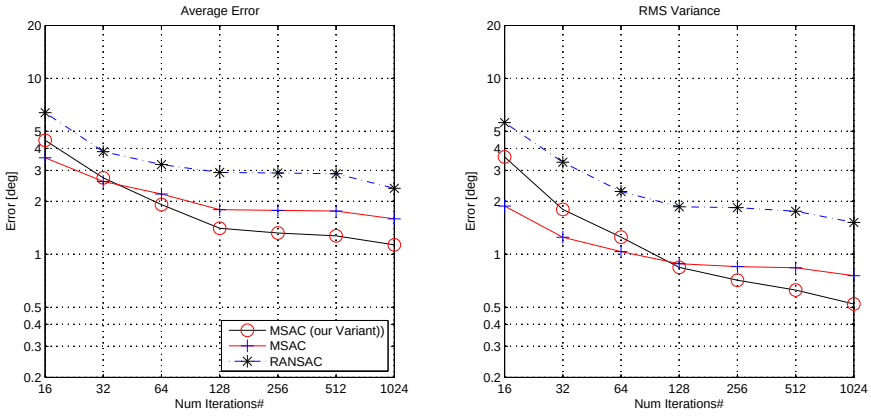
Figure 4.13: Histogram of the angular error on AIT Stereo Vision and Microsoft's Kinect

to motion blur. This overall results in that only about 10-20% wall-like parts of the environment are detected. This is reflected in the horizontal average error histogram of the AIT Stereo Vision. It *almost* fails with an almost uniform distribution while Microsoft's Kinect shows good results. While pitch and roll from the Manhattan system geometry is detected properly for both AIT Stereo Vision and Microsoft's Kinect, it gives random yaw angles for AIT Stereo Vision $\approx 10\%$ of the time[8] and $\approx 30\%$ plausible yaw angles within 5 degrees error range. About $\approx 40\%$ of the Manhattan system geometry estimates are rejected for AIT Stereo Vision (also included in the histogram), while about $\approx 5\%$ are rejected for Microsoft's Kinect.

[8] if very little wall segments are visible in the depth data



(a) AIT Stereo Vision



(b) Microsoft's Kinect

Figure 4.14: Evaluation of the accuracy of our MSAC and standard RANSAC/MSAC approaches. All plots use logarithmic scale for both axes

The cumulative average error is shown in figure 4.14 using different numbers of maximum iterations and using different estimators.. For comparisons, we also show the results of RANSAC methods using the same inlier-range as MSAC. As expected, all methods converge with an increasing number of iterations. The mean shift method shows a large improvement in Microsoft's Kinect while it has little effect on stereo vision. Variance proves this point for AIT Stereo Vision and Microsoft's Kinect. While AIT Stereo Vision converges within few iterations, it does improve with Microsoft's Kinect. The average error for AIT Stereo Vision is 2.3 degrees and 1.15 degrees for Microsoft's Kinect.

Now we consider the influence of noise on the Manhattan system geometry estimation, as is seen in fig. 4.15. We apply Gaussian noise on the depth data before the normal

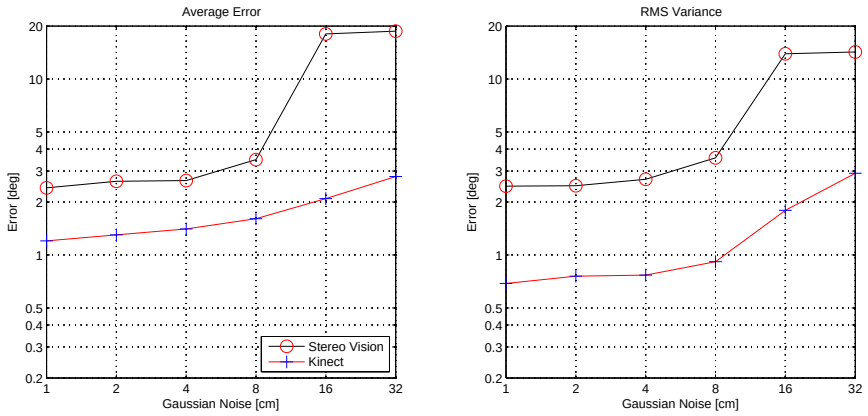


Figure 4.15: Evaluation of the accuracy of our MSAC approaches using arbitrary Gaussian noise on the 3D data . The noise is applied before normal vector calculation. All plots use logarithmic scale for both axis

vectors are calculated. Noise has little effect on Microsoft's Kinect; it has 32cm noise while the variance increases significantly with 16cm. AIT Stereo Vision almost fails at 16cm noise while it is relatively stable below 8cm. Similar to Microsoft's Kinect, the variance increases at 16cm and results in less certain estimates. One reason why noise has little influence is that we use a normal vector calculation based on integral images. Since we use a pixel window region based method for normal vector calculation, instead of using a depth range, the vectors are smooth. AIT Stereo Vision drops at 16cm because the depth data is less dense than Microsoft's Kinect; therefore, it can only be smoothed within that limit.

Last, we consider the stability of our approach regarding occlusion within the image, see 4.16. The idea is to use two masks to occlude certain parts, for example (e.g. 20%) in the image that we not be used for processing. We use two types of masks: one that occludes small non-connected parts which is referred to as "small" and one that occludes only connected parts, called "big", see fig. 2.10 on page 23 for details. Both masks are identical for each image and are generated for each amount of occlusion. One can see that the type of mask has little influence on the error due to the fact the MSAC method does not rely on local coherent segments like the 2D vision counterpart does. Both AIT Stereo Vision and Microsoft's Kinect fail after 70% occlusion. At 70%, both methods degreesenerate about $\approx 2.5deg$ degrees.

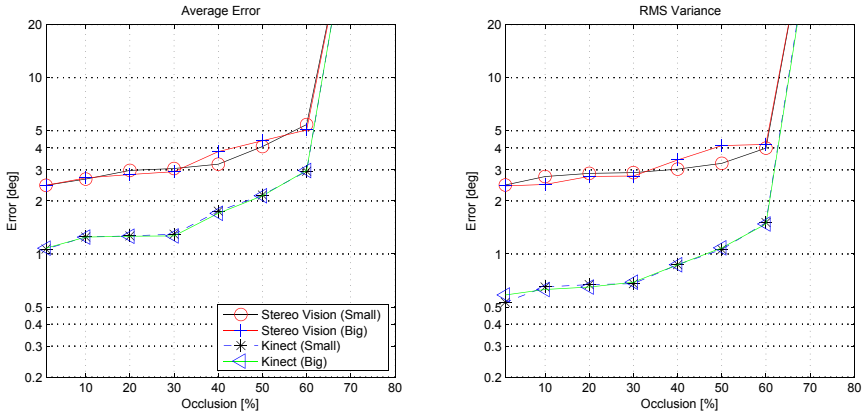


Figure 4.16: Evaluation of the accuracy of our MSAC approach using different two kinds artificial occlusion. The "small" mask occludes non-connected parts of the image using a filled circles. The "big" mask occludes connected areas. The plots use logarithmic scale for the error axis

4.3.2 Minimum Entropy Estimation

The main idea of our approach [39] is to use minimum entropy in histograms as metric. One advantage of using histograms is that it is relatively easy to estimate the Manhattan-like structure using the principle of minimum Shannon entropy. We estimate the relative camera orientation (roll, pitch, and yaw) with regard to the Manhattan system geometry so that the dominant structure is aligned to all three axes (X,Y and Z). One disadvantage is that we lose spatial information about the voxels.

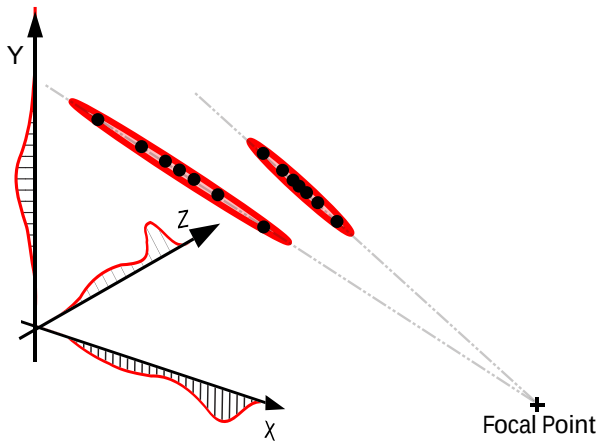


Figure 4.17: The uncertainty of the data, shown as red ellipsoids, is approximated using additional voxels

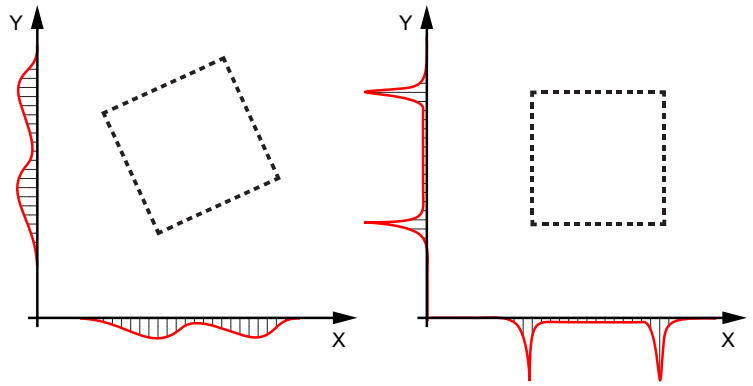


Figure 4.18: Basic idea of estimation using minimum entropy. Both figures contain the same data, but with a different configuration. The entropy of the histogram in the right figure is significantly smaller than the entropy on the left histogram

Similar to the previous chapter, the main issue with 2.5D data is how to cope with uncertainty. One way to deal with this is to use an ellipsoidal representation for uncertainty of the individual voxels; see chapter 4 for details. The mapping of the ellipsoids to the histograms is done by approximating the ellipsoids with additional voxels. The main idea is to approximate the density of the ellipsoid with additional sets of voxels in the fashion of particle filters. This approximation is sufficient as histograms for data processing are used. Since the minor and vertical axes of the ellipsoid are fixed (see fig. 4.6 on page 82), we only approximate the major axis with voxels. The length of the major axis is obtained by using the c_{onf} metric as shown in figure 4.3 on page 82. The additional voxels are drawn within the major axis with a simple Gaussian distribution. Additional voxels are drawn as follows: Let $m \in \mathbb{N}$ be the level of interpolated voxels $\sqsubseteq' = \{v_{-m}, .., v_m\}$ for one individual voxel v and let $v_j = \{-m, .., m\}$. The new interpolated voxel v is given using the Gaussian normal distribution as

$$v' = v + a^i c_g \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{v_w^2}{2\sigma^2}}$$

with c_g as normalizing constant and $v_w = \frac{v_j}{m+1}$. The term v_w prevents that new voxels are drawn at the borders with small m, e.g. $m < 2$. For the sake of simplicity we assume $\sigma = 1$ for all ellipsoids. Sinha et al. [75] have used a similar approach. An alternative approach is to sample the new voxels randomly using the Gaussian kernel.

The goal is to estimate the camera orientation using the principle of minimum entropy of histograms. First, three independent 1D histograms for X, Y and Z are built from

an arbitrary configuration (α, β, γ) i.e. the data is rotated first with γ yaw, then β pitch and α roll. Next, the Shannon entropy $H(X)$ of all three normalized histograms X_x, X_y, X_z is calculated with

$$H(X) = -\sum_{i=1}^{k} p(x_i) \log_{10} p(x_i)$$

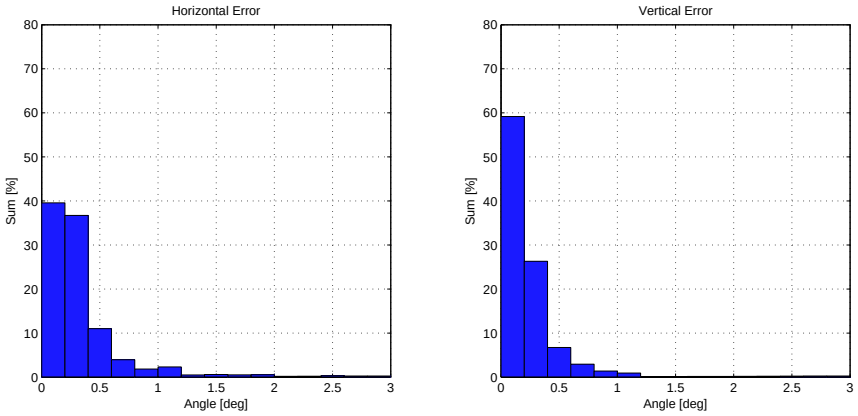
where k is the number of bins of the histogram and $p(x_i)$ is the value in the histogram at bin i. In the case of $p_i = 0$ for some i, the value of the corresponding summand $0 \log_{10} 0$ is taken to be 0, which is consistent with the limit $\lim_{p \to 0+} p \log_{10} p = 0$.

The Manhattan system geometry configuration estimate N is obtained with

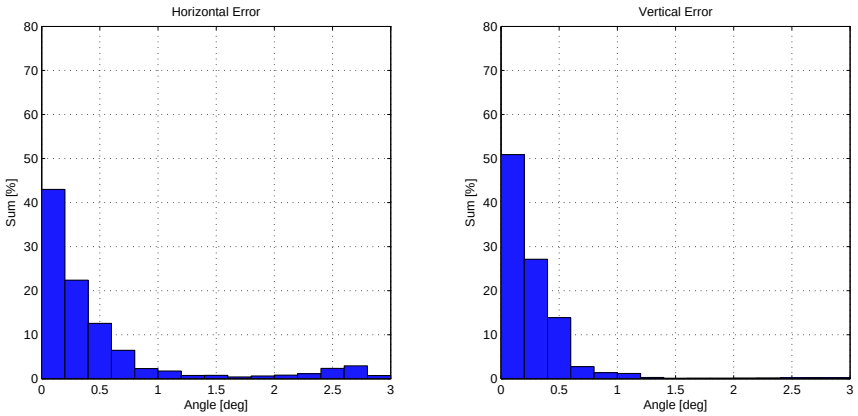
$$\mathsf{N} \quad = \quad \underset{X}{\operatorname{argmin}} H(X_x) + H(X_y) + H(X_z) \tag{4.16}$$

i.e. the configuration with the lowest overall entropy. Similar approaches have been used by Gallup et al. [87] for multi view reconstruction and Saez et al. [107] for vision-based SLAM. This kind of metric does not provide a framework to validate the plausibility of an estimate since it always estimates the "most likely solution". Due to that, we only use it to estimate the global Manhattan system.

Figure 4.18 depicts the overall concept: If there are parts in the image which are orthogonal to one of the X,Y,Z axes, such is the case in Manhattan world assumption, then it is possible to find a configuration using minimum entropy. It is not necessary that all parts in the image are Manhattan-like or directed to its main axis as long there is some structure orientated to the major axis. Please note that fig. 4.18 shows the concept in the 2D case for the sake of simplicity. We also want to emphasize that our approach can also deal with occlusions due to the fact we build histograms direct on voxel in contrast to many monocular, vision-based approaches [20, 75, 87].
One way to obtain an estimate is to use non-linear optimizers like the Levenberg-Marquardt algorithm [50]. This is typically used for *argmin* in the computer vision literature. The Levenberg-Marquardt algorithm depends on proper initialization to avoid getting stuck in local minima. Instead, we use traditional Monte Carlo Particle filters [7] to estimate and track the most believed estimate of minimum entropy, see page 125 for details. We use a particle filter to estimate and track the camera orientation (Manhattan system geometry configuration) using 128 particles in the (α, β, γ)



(a) AIT Stereo Vision (Cumulative error seen on the plot on the left $\approx 95.6\%$, right $\approx 96.9\%$)

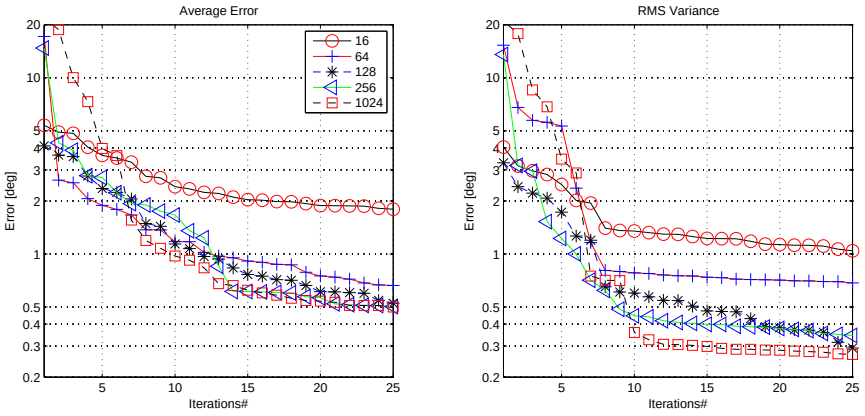


(b) Microsoft's Kinect (Cumulative error seen on the plot on the left $\approx 96.3\%$, right $\approx 97.9\%$)

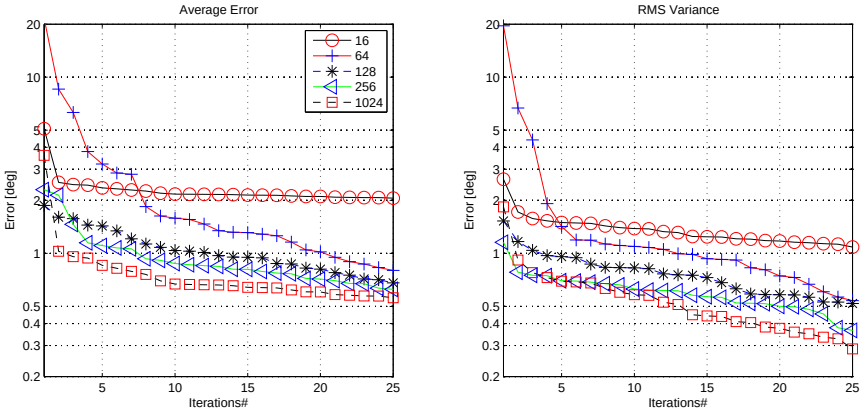
Figure 4.19: Histogram of the angular error on AIT Stereo Vision and Microsoft's Kinect

state space instead we initialize the particles in the Monte Carlo fashion. In this case, we favor robustness instead of accuracy. A possible extension is to move the particles according to the motion of the robot as motion model. In each iteration, 10% of the used particles are random in order to avoid local minima. We use 5cm for the histograms within 10m range which is a good trade-off between speed and accuracy. In a last step, we obtain the variance N_σ per axis from the particles, see [7] for details. The variance is later applied in the fusion of Manhattan geometry.

Our code runs on our test computer (see page 27) at 38ms for AIT Stereo Vision for 128 particles ($m = 2$) with **no** multi-threading. For Microsoft's Kinect, we need 42ms using all data. The bottleneck of our approach is the construction of the histograms



(a) AIT Stereo Vision

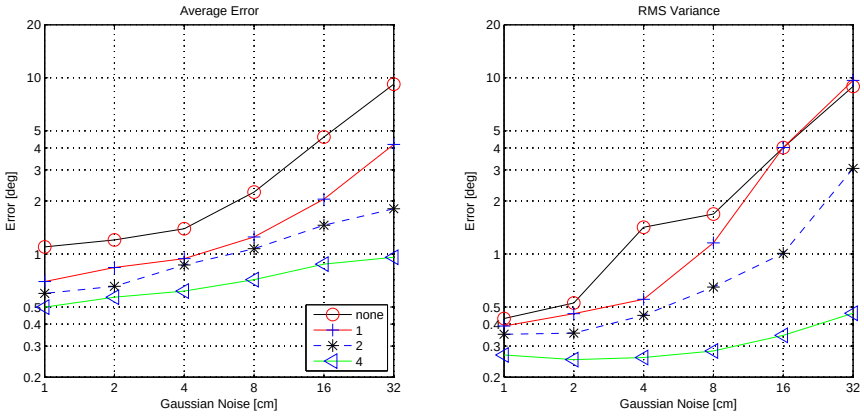


(b) Microsoft's Kinect

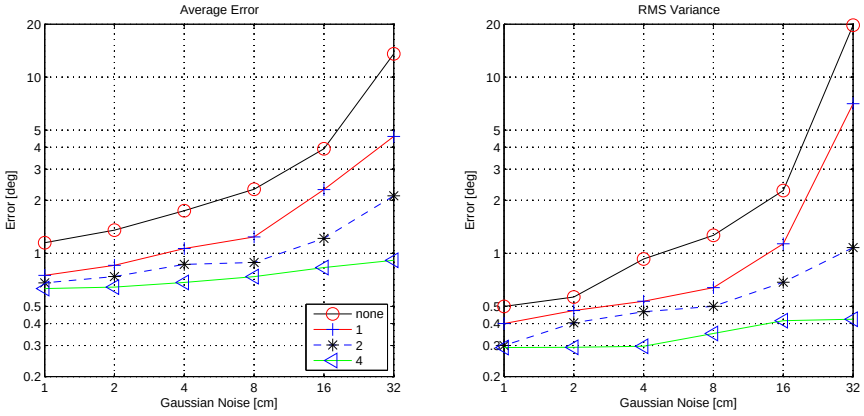
Figure 4.20: Evaluation of the accuracy of our minimum entropy approach using different number of particles. All plots use logarithmic scale for the error axis

per pixel and uses 90% of the computational power. 8% is used to calculate the rotation of the voxels and the rest is used to calculate the entropy.

The average error of both sensors is shown as a histogram in figure 4.19. Since we use particle filters, we evaluate only the error after an initialization phase of approximately 30 iterations, because we use a Monte Carlo-like initialization of the particles. One can see that AIT Stereo Vision performs better than Microsoft's Kinect in both vertical and horizontal error. Similar to the previous MSAC method, this is due to the larger field of view and the use of tracking. Both sensors show slight artifacts on the horizontal error within the 2.5 degrees range. This is due to the resampling method that uses exactly a noise range of 2.5 degrees for new particles i.e. spreading new particles within this range. In some rare cases, (i.e. $> 1\%$) it can cause the tracker to be stuck



(a) AIT Stereo Vision



(b) Microsoft's Kinect

Figure 4.21: Evaluation of the accuracy of our minimum entropy approach with voxel upsampling i.e. m and arbitrary Gaussian noise. All plots use logarithmic scale for both axis

in a local minima if the robot is not rotating. In contrast to the MSAC method, the minimum entropy method can cope with little wall information since it uses a Monte Carlo style tracker, see page 94. Figure 4.20 shows the particle tracker during the initialization phase. All trackers using at least 64 particles converge after 25 iterations with similar accuracy. Microsoft's Kinect converges faster with an increasing number of particles due to the more dense depth data while the variance converges faster with AIT Stereo Vision due to less amount of data. The average error[9] for AIT Stereo Vision is ≈ 0.5 degrees and ≈ 0.61 degrees for Microsoft's Kinect (128 particles, $m = 2$).

Next, we study the influence of Gaussian noise on our particle interpolation method,

[9]Please note that the average error for the ground truth data is 0.2 degrees

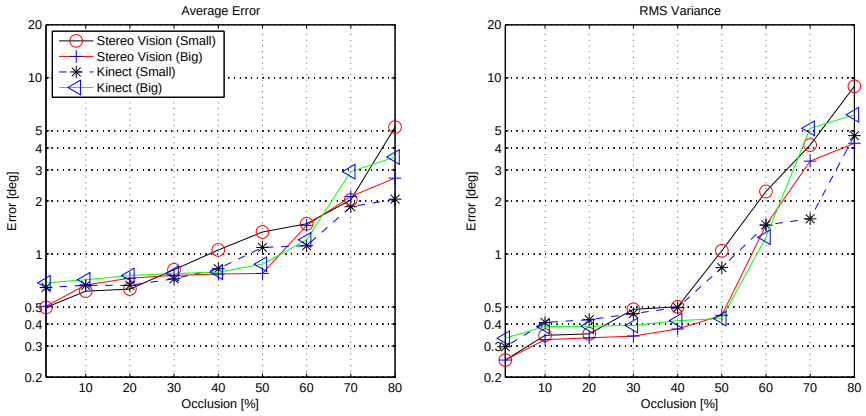


Figure 4.22: Evaluation of the accuracy of our minimum entropy approach using different two kinds artificial occlusion. The "small" mask occludes non-connected parts of the image using a filled circles. The "big" mask occludes connected areas. The plots use logarithmic scale for the error axis

see fig. 4.21. One can see the effect of an increasing m number of interpolated voxels according to their uncertainty. The overall error for 32cm Gaussian noise is less than ≈ 0.5 degrees with $m = 4$ in contrast to the 2.5 degrees for the previous MSAC method. Another improvement can be seen for an increasing m with no noise resulting in a lower average error. Although these improvements are less than 0.6 degrees for both AIT Stereo Vision and Microsoft's Kinect, it shows the robustness of explicit modeling of uncertainty. While an increasing m results in more interpolated voxels, the computing can still be done quite efficiently. The constraint within histogram generation is the rotating of the voxels according to their Manhattan system geometry configuration. As figure 4.17 suggests, only the center voxel and view point is rotated; then the voxels are interpolated. Using this technique, the runtime for construction of the histogram is only slightly influenced m $\frac{1}{4}\times$ for $m > 5$.

Last we consider the stability of our approach regarding occlusion, see 4.21(b). With MSAC, the idea is to use two masks to occlude certain parts (e.g. 20%) in the image that will not be used for processing in our method.. We use two types of masks: one that occludes small non-connected parts "small" and one that occludes only connected parts "big", see fig. 2.10 on page 23 for details. One can see that the type of mask has little influence on the error while the small mask has a slightly bigger influence. While the amount of occlusion influences only slightly the average error, it harms the variance with more than 50% occlusion. This occurs because the used histogram is

utilized less than with no occlusion. This results in a less certain (larger variance) estimate since we use minimum entropy.

4.4 CC-RANSAC Plane Estimations

The use of 3D planes as features from 3D data has become the de facto standard in robotics in recent time [91]. While many approaches use only 3D data from 2.5d sensors, we want to exploit the 2D component of the underlying data structure too. Our approach is as follows: First we assume a global/local Manhattan system geometry configuration[10]. We use normal vectors (see chapter 4.1 on page 77) for each pixel in the 2.5d grid and assign every vector the most plausible Manhattan system geometry. In the second step we use a constrained 1D RANSAC variant to extract planes within the Manhattan system geometry. Since we use 2.5d sensors we can exploit their properties to improve the overall robustness of the system. We directly estimate the error for the plane model of the 1D RANSAC in sensor space rather than in Euclidean space. Another advantage is that we can use the 1D RANSAC in combination with a connected component analysis to extract only continues (pixel wise connected) planes and extract their borders directly like on a 2D image.

The plane segmentation [26] exploits the fact that we use the data on a 2.5D grid by using a constrained RANSAC variant with connected competent analysis in the fashion of CC-RANSAC proposed by Gallo et al. [108]. The CC-RANSAC first estimates the parameters of the model using traditional RANSAC methods. Instead of counting all inliers within the model, CC-RANSAC counts only the inliers of the largest cohered area (of connected components), see figure 4.23. Since we use 2.5D, we can perform this analysis like on a 2D image in an efficient way. Gallo et al. [108] states that the use of the connected components analysis leads to $4-5\times$ less iterations than without. He also states that in a worst-case scenario, CC-RANSAC performs like a normal RANSAC method. Another property is that the use of the largest cohered area can be used as a metric, such as size/area [108], for the iterative estimation of all planes within the 2.5D data.

The estimation of the planes within the 2.5D data is done with a constrained 1D CC-RANSAC with the standard Hessian normal form $ax+by+cz+d=0, a,b,c,d \in \mathbb{R}$.

[10] For the sake of simplification we show the algorithm for the global Manhattan system, while the local Manhattan system is the same (only the non shared axis data is calculated)

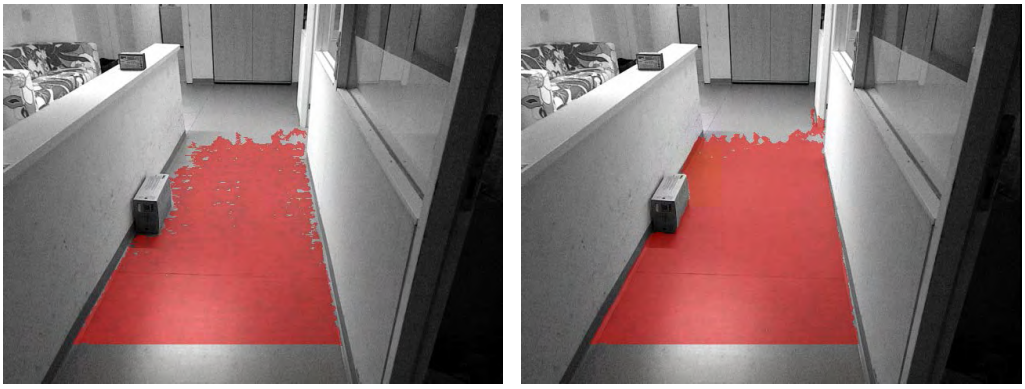
(a) CC-RANSAC (b) RANSAC

Figure 4.23: Comparison of Plane estimation using CC-RANSAC and standard RANSAC using normal vectors, *no* least square fit, 16 estimation iterations and 5cm distance threshold to the plane model

Since the global and local Manhattan system geometry are estimated in the previous steps, we can assign every normal vector (for each voxel) the most plausible relationship to one of the three major axes of either the global or the local one. In our case, the plausible relationship to an axis is the one with the lowest error. In this stage, we do not use a threshold for maximum error. The general assumption here is that the point belongs to a Manhattan geometry, so we need only one point to model the plane since the normal vector results from the Manhattan system instead of using the normal vector of the voxel. The random vector is drawn from a previous sorted list according to the vector's corresponding voxel confidence in depth perception $c_{onf} \in \mathbb{R}$, see chapter 4. The list has been sorted with an integer[11] *bucket sort* [104] in linear runtime from high to low confidence values. Additionally, we smooth the vector within a 3x3 grid using the confidence in the depth perception metric as weight. This results in a rough estimate as shown in figure 4.23(a) and is refined with a standard least square fit on the best estimate. Experiments have shown that the method of using a local (e.g. 3x3 grid) smoothed vector is more robust than smoothing a random number of vectors within the same geometry since there is no guarantee they belong to same plane. This results in the need of more iterations. Using equation 4.8 on page 89, we calculate that we need at least 32 iterations assuming a pessimistic probability of 20% that data points do belong to our model.

Since we use only one point for the CC-RANSAC, we need less iterations to estimate

[11] the c_{onf} values have been mapped to 1024 integer values



(a) Euclidean Distance (b) Disparity Distance (native Sensor space)

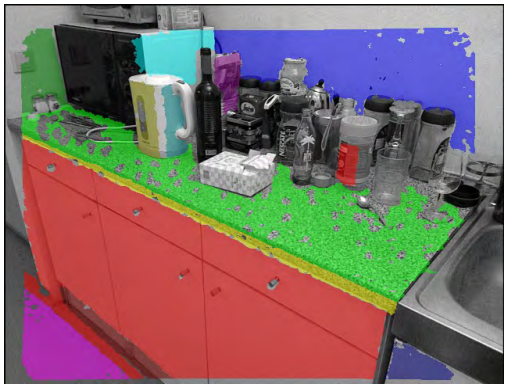
Figure 4.24: Comparison of distance metric using our CC-RANSAC approach on sample data. The Euclidean distance metric shows artifacts due to the non-linear depth scaling of the Kinect. This effect is typical for camera based sensors

planes within the dataset than with the Hessian normal form for planes. Experiments have shown that more than 70 percent of the data are planes with the global Manhattan system geometry and can be extracted within a few repeated segmentation steps in many cases.

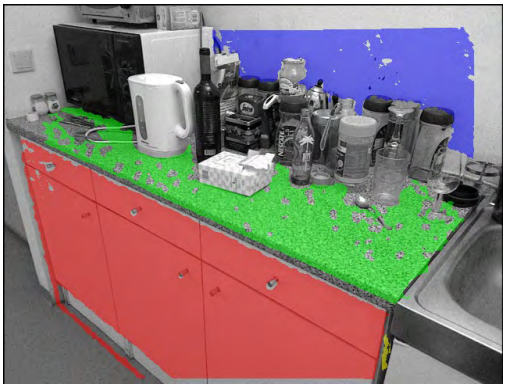
To improve the quality of segmentation, we measure the error of the data to the estimated model directly in the sensor space i.e. in disparity rather than using the Euclidean space, see figure 4.24. For instance, Microsoft's Kinect sensor space is a disparity map with 8 sub-pixel interpolation. We use 12 depth disparities as maximum error for our plane estimation. We use 16 for AIT Stereo Vision. The overall robustness is not influenced by working on the sensor space directly.
Multiple planes are estimated with the normal RANSAC style. First, we estimate the model with the most inliers and use it as an estimate if the number of inliers represent at least 10% of the entire image (also counting pixels with depth data). The samples are removed from the set and repeated as long as no more planes can be found. Figure 4.25 shows this process. Planes that have been found first are drawn[12] in red, then sequentially after that: green, blue, yellow, cyan, magenta. One can see that our methods show good results using only a low number (32) of iterations per plane.

Next, we consider the qualitative error of our plane segmentation in combination of the Manhattan room structure estimation. We use 100 randomly selected images from

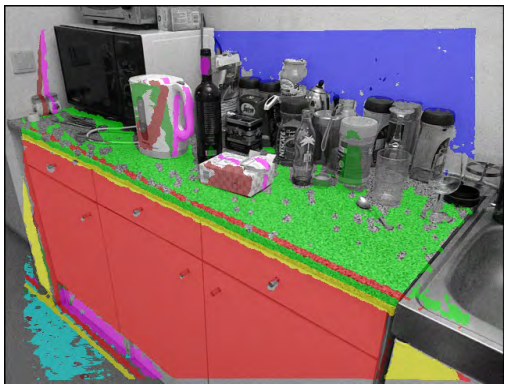
[12]We repeat the color palette for the sake of simplification



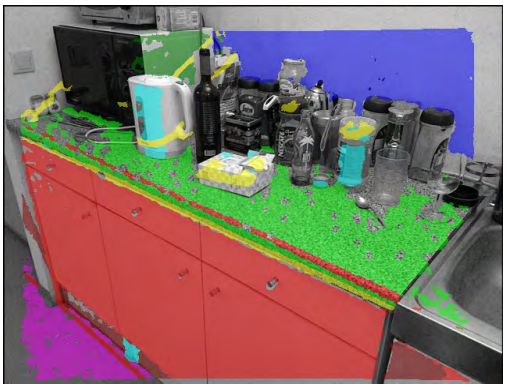
(a) 1D CC-Ransac (our Approch)



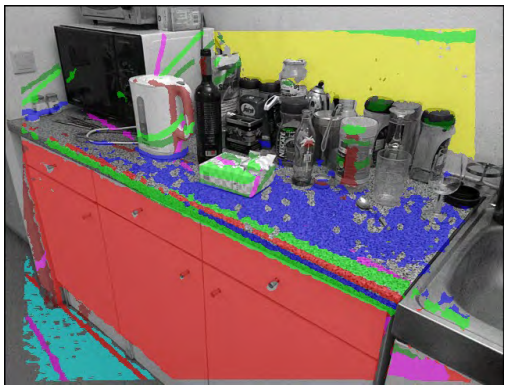
(b) CC-Ransac [108] (with Normal Vectors)



(c) MSAC [50] (with Normal Vectors)



(d) RANSAC [91] (with Normal Vectors)



(e) RANSAC [91] (without Normal Vectors)

Figure 4.25: Comparison of different plane segmentation methods in a cluttered scene from one of the tours with Microsoft's Kinect. The maximum iteration for all methods was fixed to 32 iterations. Note that all methods show similar results to our CC-RANSAC approach if enough iterations were used (see table 4.1) resulting in a much longer calculation time

the tour and generate ground truth for the dominant Manhattan structure. The error is estimated for all three major axes of the Manhattan system individuality and added together. In the case of the unconstrained RANSAC variants, we choose only

Table 4.1: Average Runtime for the different estimation Methods for estimating one valid model per Manhattan system.

	Runtime per Iteration [ms]	minimum Iterations
RANSAC	0.2	400
MSAC	0.2	250
CC-RANSAC	0.7	100
our Approch (1D CC-RANSAC)	0.7	32

planes that are plausible (i.e. within a certain threshold) to one of the three major X, Y, Z axis. Figure 4.26 shows a quantitative curve on the true/false positive rate relative to the number of iterations. One can see that all RANSAC variants converge at almost the same rate if we use enough iterations. The data has been smoothed with a 5bin Gaussian kernel for better visibility due to the relatively low number of samples.

Our code runs with 12.3ms (AIT Stereo Vision) / 22.1ms (Microsoft's Kinect) on our test computer (see page 26) for approximately 6 / 14 planes per image. The code is not optimized and uses only one CPU for comparable results. Table 4.1 shows the overall runtime of the evaluated RANSAC variants. Note that the estimation of our 1D CC-RANSAC is overall computationally more expensive than a MSAC, because we have to estimate the global Manhattan system first. In practice, our approach (with Manhattan system estimation) can be faster than MASC since we need less iterations before the error converges which results in a faster runtime. About 90% (90.6% are planes) of the data is compressed removing all estimating planes from the data.

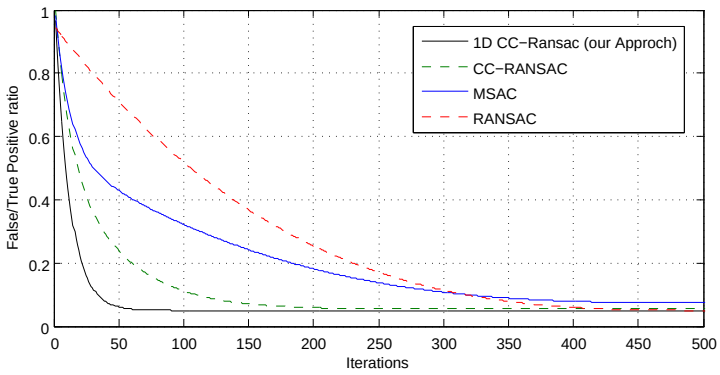


Figure 4.26: **Smoothed** average false positives to true positives ratio for plane segmentation. The lowest ratio is 0.05

4.5 Mean Shift Plane Hypotheses

The use of RANSAC-based plane estimation has a long history within robotics (see previous chapter) and became quite popular with the first 3D laser scanners, see [76] for details. The use of parametric plane models depends on the quality of the 3D data. It can be used with dense data e.g. from Microsoft's Kinect with about 80% data depth data per image, but it can be difficult with less dense depth data (around 30%) per image like AIT Stereo Vision resulting in few estimates. Depending on the implementation, this can result in plausible but false estimates like standard RANSAC. An example of this would be if a plane is detected within a door frame with no door (see fig. 4.23(b) on page 105)), or with Connected Component RANSAC (CC-RANSAC) with no estimates since the estimated planes (largest connected area) are too small. Instead of seeking parametric planes, we introduce an alternative concept [39]: The *plane hypotheses*. The idea is to describe an area with possible planes in which there exists a far weaker assumption than in the standard parametric plane model. We use clustering techniques instead and validate the plausibility of the data. This idea is similar to the approach of Furukawa et al. [20] and Sinha et al. Sinha et al. [75].

Once the Manhattan system geometry is known, we can generate plane hypotheses from voxel data. The voxel data is first interpolated with extra "upsampled voxel" (i.e. 8 per pixel) to model the uncertainty, see 4.3.2 on page 98 for details. This is done in three steps: First, generate individual 2D histograms on all possible X,Y,and

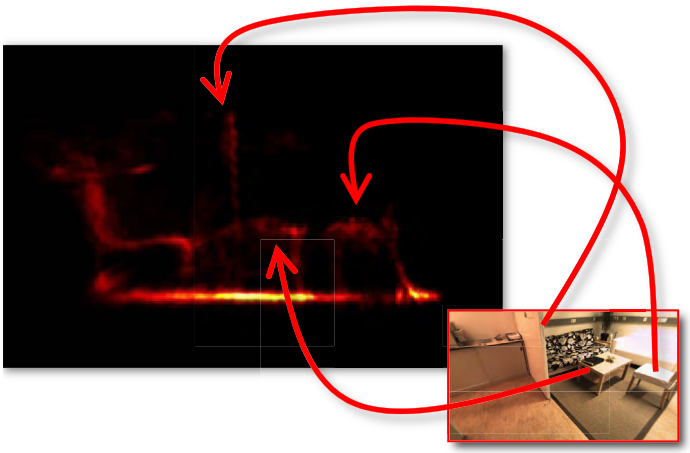
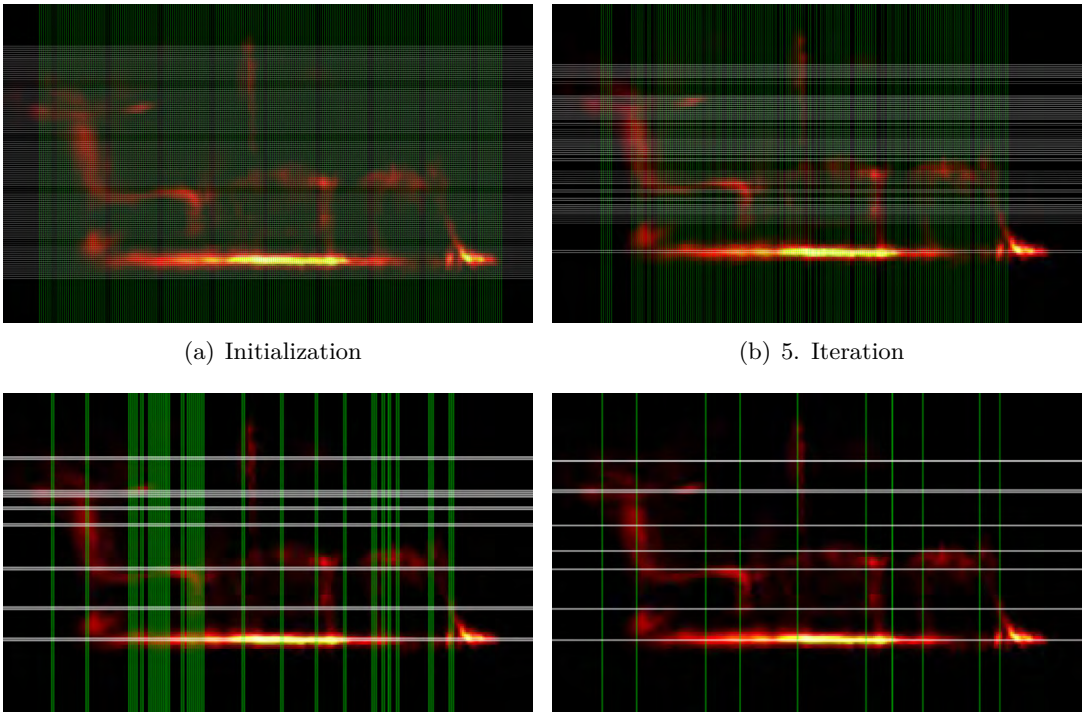


Figure 4.27: Overview of the data in a histogram from AIT Stereo Vision and real image. The orientation has been removed since the Manhattan system geometry is known



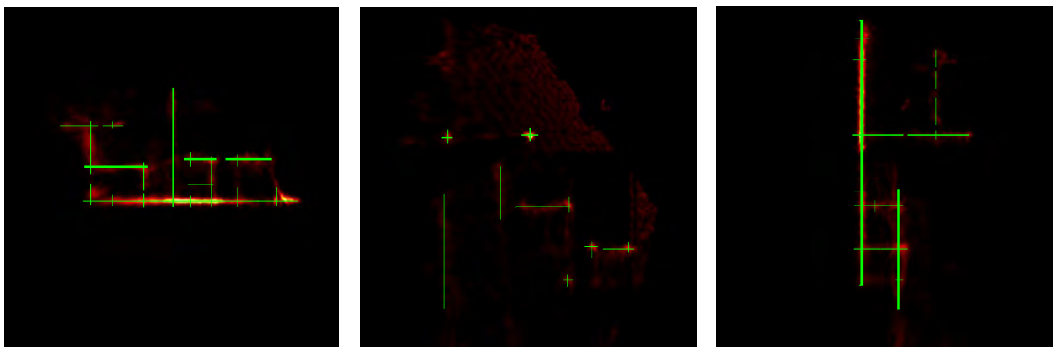
(a) Initialization (b) 5. Iteration

(c) 8. Iteration (d) 30. Iteration

Figure 4.28: Constrained mean shift clustering on the X/Y histogram of Figure 4.27

Z combinations i.e.X/Y, Z/X and Z/Y. In the next step, extract line segments in the 2D histograms and group them **into** planes in the last step.

The generation of the 2D histograms is straightforward: First, an inverse rotation is applied to the upsampled voxels using the known camera orientation. This aligns the voxels to one of the three major axes. Usually, not all parts of an image are aligned



(a) X/Y (b) X/Z (c) Y/Z

Figure 4.29: Edge grouping with hysteresis of Figure 4.27

to the major or dominant axes. However, here we assume that the majority of data is aligned to the dominant axes. For the histograms, we use a resolution of 5cm per bin. Please note that the resolution of 2.5d sensors does not scale linearly, and we therefore use voxel upsampling.

Using Manhattan-like environments, the planes are represented as vertical and horizontal lines in the histograms. Extraction of the planes is done all in two steps: First, we group segments in the histogram together using the mean shift algorithm [58] and edge grouping with hysteresis in the fashion of the canny edge detector [30].
Mean shift itself is a procedure for locating the maxima of a density function given discrete data x sampled from that function. In our case, we use it for detecting the modes of this density. This is an iterative method, and we start with an initial estimate x. Let a kernel function $\mathrm{K}(x_i - x)$ be given. This function determines the weight of nearby points for re-estimation of the mean. We use the Epanechnikov kernel

$$\mathrm{K}(x) = \begin{cases} 1 - ||x||^2 & if||x|| \leq 1 \\ 0 & if||x|| > 1 \end{cases}$$

on the distance to the current estimate,

$$\mathrm{K}(x_i - x) = e^{c||x_i - x||}$$

The weighted mean of the density in the window determined by K is

$$m(x) = \frac{\sum_{x_i \in N(x)} K(x_i - x)x_i}{\sum_{x_i \in N(x)} \mathrm{K}(x_i - x)}$$

where $N(x)$ is the neighborhood of x, a set of points for which $\mathrm{K}(x) \neq 0$. The mean-shift algorithm now sets $x \leftarrow m(x)$, and repeats the estimation until $m(x)$ converges to x or the maximum of iterations is reached. See page 38 and fig. 3.8 (pp. 38) for details.

Instead of applying the mean shift on the entire 2D histogram, we first reduce the 2D histograms to two 1D histograms and apply the mean shift separately (due to the Manhattan world assumption), see figure 4.28. We use a parameterized Epanechnikov kernel with a size that represents 7.5cm in the real world. In the next step we use the output of the mean shift clusters as input for the line boundary detection using the counterpart 1D histogram: For instance, we use an X/Y histogram. We apply the

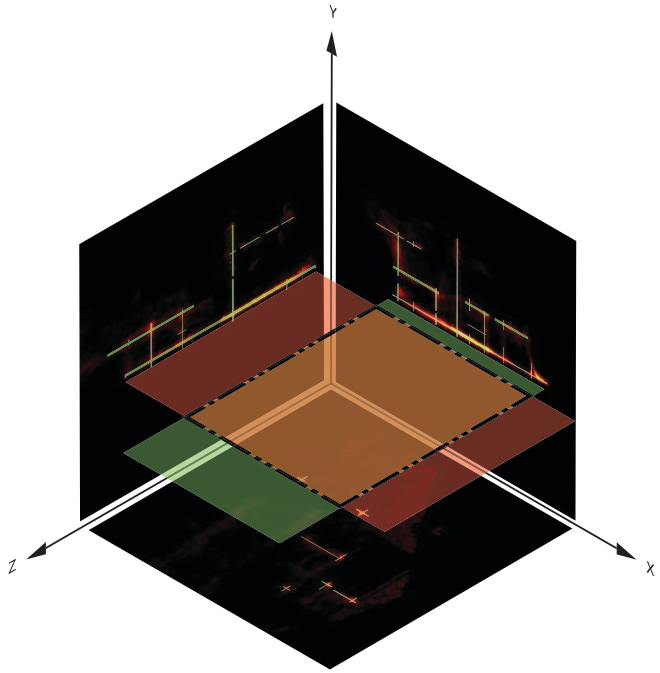


Figure 4.30: Principle of the plane hypothesis generation of Figure 4.27. The line segments of two orthogonal aligned histograms are used to generate planes. The intersected area is the actual output. Note that it is possible that more than one plane hypothesis is generated per one line

mean shift on the X 1D histogram to determine the height of the line and use the Y to estimate the boundaries of the line or lines. The boundaries are estimated in the hysteresis fashion as used in the canny edge detector. The algorithm assigns the labels "no edge" (0) "maybe edge" (1) and "edge" (2) to each pixel within one line. Instead of using only the pixels within one mean shift cluster, we use the maximum value within the variance of the mean shift cluster. That is to say in the case of a horizontal cluster "line" we also consider pixels that are "above" and "below" the line. The variance is calculated using simple backtracking, see [58] for details. Let T_m and $T_h, T_h < T_m$ be two thresholds. T_m denotes the "edge" threshold while T_h is the hysteresis threshold. Let q_i be pixel value from the cluster at position i and

$$C(p_i) = \begin{cases} 2 & if ||p_i|| \geq T_m \\ 1 & if ||p_i|| \geq T_h \\ 0 & if ||p_i|| < T_h \end{cases}$$

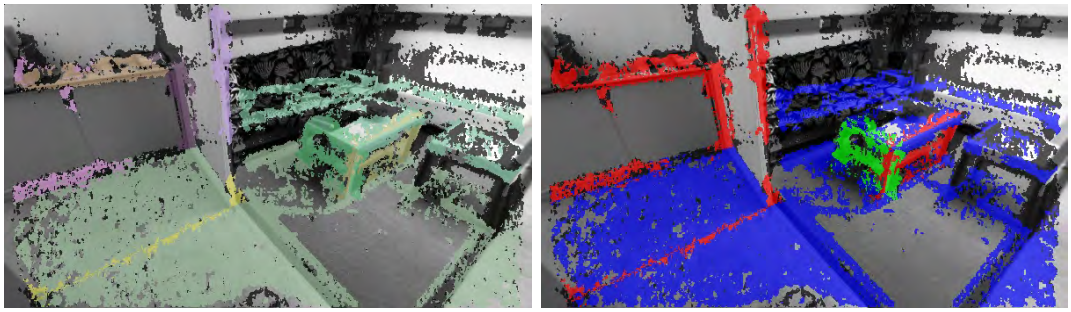
be a function that assigns labels to the pixels p. Next, the algorithm assigns the label

"edge" to labels with "maybe edge" if an "edge" pixel is nearby within the recursive function

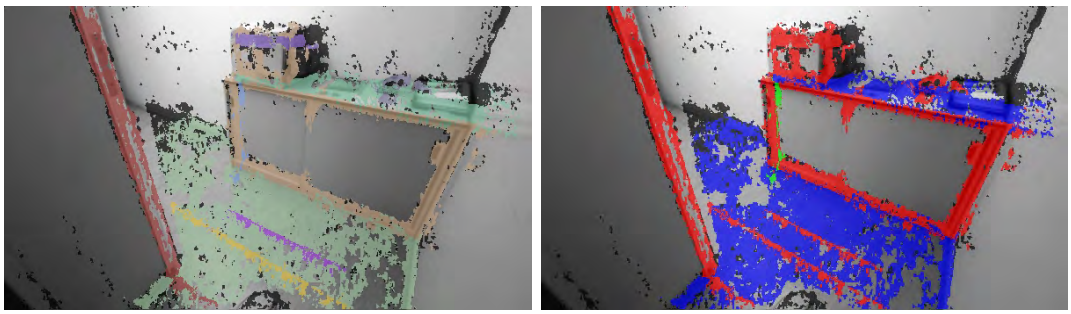
$$C'(p_i) = \begin{cases} 2 & if C(p_i) = 2 \\ 2 & if C(p_i) = 1 \wedge (C'(p_{i-1}) = 2 \vee C'(p_{i+1}) = 2) \\ 0 & if C(p_i) = 0 \end{cases}$$

Finally we group all "edge" pixels to line segments using simple run length encoding, see figure 4.29.

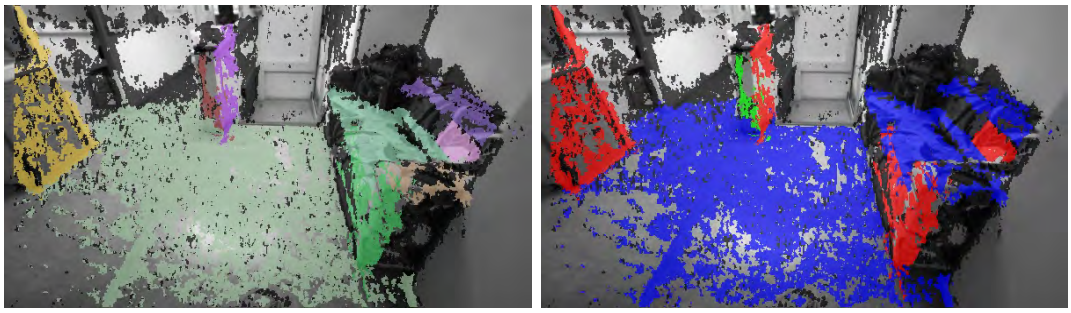
Now we group the line segments to the plane hypothesis. Since each line segment is aligned with one major axis (X, Y, or Z), we can generate plane hypotheses by projecting planes to the normal axes of the parent 2D histograms of two axes. For example, "horizontal ground planes" can be generated using the X/Y and Z/Y histograms and their corresponding line segments (for X and Y line segments with the same height from the ground). All "X" line segments of X/Y histogram are projected orthogonally to the Z axis. Next, we project the "Z" line segments of the Z/Y histogram and build planes the same way. The intersected area of two planes is used as the plane hypothesis in the fashion of plane sweeping methods [87]. The generation of the X and Y planes is done the same way; figure 4.30 depicts the overall concept.
Finally, we use a pruning strategy to remove plane hypotheses with no or little support from the upsampled voxels. First, we sort the plane hypotheses f according to the joint probability $p(f) = p(l_n, l_m)$ of the line pair l_n, l_m and the size (largest first). The individual $p(l_n)$ and $p(l_m)$ result from the mean shift clustering and are normalized weights, see [58] for details. Next, all plane hypotheses are evaluated by again projecting them back into the 3D state space, counting the number of inliers per plane using RANSAC. Planes with no support $count < 0.1\%$ are then removed from the set. This step removes 90% of all false planes. Another approximately $approx 8.5\%$ can be removed by utilizing plane sweeping: If a plane with a low probability occludes the visibility of a higher one, it is removed from the set. The visibility check is ego centered at the focal point. Figure 4.31 gives an example of the final labeling with and without Manhattan system membership. One can see that some labels, such as those displayed in fig. 4.31(a), are fragmented, but belong to the same physical structure. This is due to missing depth measurements on some structures like doors or walls resulting from a use of, for example, AIT Stereo Vision. These fragments are grouped later in the segmentation step.



(a) Living Room



(b) Corridor



(c) Lobby

Figure 4.31: Plane hypothesis from AIT Stereo Vision. The first row show the plane hypothesis in pseudo color coding, the second one with Manhattan system membership

Our code runs at 22ms (AIT Stereo Vision) / 36ms (Microsoft's Kinect) on our test computer (see page 27). About 60% is used for plane hypothesis pruning, while their generation only take up 5%. The use of mean shift is relatively fast (14%) since we use only three 1D histograms. The clear advantage is that the hypotheses are estimated within linear runtime since all operations are done on fixed-size histograms in contrast to our CC-RANSAC approach.

4.6 Related Work

The computer vision and robotics literature proposes different methods for the estimation of Manhattan system geometry system and features from 2.5d sensors. One difference among many approaches is the depth estimation itself. Besides traditional stereo-like configurations like AIT Stereo Vision or Microsoft's Kinect, common methods are structure from motion (SfM) and multi view stereo (MVS) from a ordered or unordered image sequence. Other approaches combine an active 3D sensor like a Lidar sensor with a 2D camera to obtain 2.5d data e.g. [109]. One key aspect of many approaches is the estimation of the Manhattan system geometry system itself and the other the extraction of features such as planes that are oriented to a geometry.

Hedau et al. [110] estimates a box-like structure within a room, which is a special case of the Manhattan system geometry assumption. The box-like structure is divided into left, right, ceiling and floor structures and is estimated separately from learned labels from 3D data. Although the approach works on a single 2D image, the strategy is related to stereo plane sweeping strategy; each sub structure is used as a view. This approach is similar to Hoiem et al. [111] which uses the same labels per image, but does not incorporate any Manhattan system geometry constraints. Gallup et al. [112] uses a vice versa strategy by estimating first Manhattan system geometry configuration and extracting planes using plane sweeping using depth data from structure in motion. The Manhattan system geometry is estimated from the structure of motion depth data using minimum entropy in histograms, which is similar to our approach. Gallup et al. [112] assumes that the vertical orientation within the images is known and it calculates only the yaw angle of the Manhattan system geometry using planar 2D histograms and the Levenberg-Marquardt algorithm [50]. The approach is extended by Pollefeys et al. [113] by introducing a visibility-based depth map fusion which includes the aspect of a moving observer and occlusion handling.

Furukawa et al. [20] estimates a rigid Manhattan system geometry system using normal vectors and spherical histogram on a unit Gaussian. Planes are extracted using Manhattan system geometry axis-oriented 1D histograms and mean shift clustering. Sinha et al. [75] uses a two-step method to estimate a Manhattan system geometry system from multi-view stereo. First, a vanishing direction (rather than the vanishing point) is estimated using mean shift clustering of 2D lines on a Gaussian sphere. In the

second Step, the lines with the same direction are estimated in 3D using triangulation and the Manhattan system geometry configuration is obtained. Planes are extracted in the next step similar to [75] based on the spares 3D features of the MVS, but using RANSAC instead of mean shift. The approach also incorporates uncertainty of the features in the 3D space using upsampled voxels which is similar to our approach. Manhattan system geometry axis-oriented 1D histograms for 2.5d plane estimation were also used by Zebedin et al. [114]. The approach extracts lines using region growing instead of mean shift or RANSAC and groups them into primitives using graph cuts. Chauve et al. [115] uses a similar approach using a piecewise planar surface metric for graph cuts. Vanegas et al. [116, 117] extends the idea of plane extraction by using shape primitives like wall, edges, etc., instead of using lines. Nan et al. [118] uses "SmartBoxes" instead of 1D histograms to estimate box-like structure instead of grouping lines to planes similar in strategy to [110]. Li et al. [109] extends this approach by adding 2D vision methods on the 2.5d depth data for edge detection.

A common technique with 3D point clouds is the use of robust estimators such as RANSAC [57], expectation maximization (EM) [68, 119] or J-Linkage [120] for parametric model fitting. The de facto standard is the use of point clouds with normal vectors and MSAC estimators Rusu and Cousins [91]. Gallo et al. [108] extends these kind of approaches by considering only estimated planes that are locally connected within an image-like grid. With these approaches, Manhattan system geometry constraints are not exploited. Holz et al. [93] exploits partially Manhattan system geometry with no rigid constrains by clustering first normal vectors with the same orientation and extracting planes in a second step. This can result in parallel planes within certain limits. Oehler et al. [121] use a similar approach using multi resolution core-to-fine strategy using Hough transformation on normal vectors and RANSAC based estimators for planes. Triebel et al. [119] estimates planes with RANSAC and groups parallel ones using expectation maximization for an unknown number of parallel structures. Another strategy to combine the IMU of a robot and depth data for partial Manhattan system geometry was proposed by Gutmann et al. [122].

4.7 Discussion

In this chapter, we presented algorithms for estimating global and local Manhattan systems, estimating planes and plane hypothesis. We showed that the explicit modeling of uncertainty with 3D data improves the accuracy. We use a dual strategy to deal with different kind of data. The first kind is based on RANSAC/MSAC methods and uses normal vectors as input. The basic drawback of this method is that it fails if no valid normal vectors can be computed. The other strategy is based on histograms, which show a higher robustness, but achieve less accuracy (Mean Shift Plane hypothesis). We use the second strategy in practice as a fallback if the first one fails while using it to improve the overall results by fusing them in the segmentation step. All presented methods work on both Microsoft's Kinect and AIT Stereo Vision but are not limited

Table 4.2: Summarized properties of the introduced algorithms in this chapter with non-multithreading and non-optimized code on our test computer (see page 27). The typical runtime is given for Microsoft's Kinect and is in average 2.3× slower than AIT Stereo Vision

	Normal vector calculation	Manhattan system estimation with MSAC	Manhattan system estimation with minimum Entropy	1D-CC RANSAC Plane Estimation	Mean Shift Plane hypothesis
Accuracy	o	+(++)	++	++	o
Robustness		+	++	+	++
Handling of Uncertainty	o	+	++	+	++
Estimating multiple model		yes	no	no	yes
estimation method	direct	MSAC	particle filter	RANSAC	mean shift
parametric Model		yes	no	yes	no
num model dimensions	2	3	3	1	2
Polynomial Complexity	O(n)	O(nm)	O(nm)	O(nm)	O(nm)
typical n	$\frac{640x480}{2}$	$\frac{640x480}{2}$	$\frac{640x480}{2}$	$\frac{640x480}{2}$	$3 \cdot 2 \cdot 200$
typical m		$169 \cdot 4$	128	$32 \cdot 32$	64
typical runtime (ms)	100	90	38	$0.7 \cdot 32$	34

to these. Experiments have shown that the histogram methods also work fine with the SwissRanger SR-3000 and the tilting laser scanners. As long as the uncertainty can be modeled, reliable features can be extracted and the data structure is based on 2.5d the methods can be applied.

The major drawback of our method's 3D process is that we depend on a proper estimated global/local Manhattan system geometry. While it is possible to detect a non-existing Manhattan system, it cannot always be guaranteed that room structure will be fully orthogonal. The use of a relaxed Manhattan constraint is one possible solution in that the major axes are almost 90 degrees orthogonal to each other. If no Manhattan system is found, both plane estimation and hypothesis methods are not able to generate any output. Another issue is the detection of a falsely estimated Manhattan system. At this point of our processing chain, we cannot validate a system since we detect it only from frame to frame and have no prior information of the motion of the robot. This is done in a later step. In practice we use fused estimates before feature extraction.

Table 4.2 gives an overview of the proposed algorithms and their properties. One can see that the non-parametric methods show an overall better robustness as is seen with AIT Stereo Vision. Since the MSAC Manhattan system geometry estimator depends on the quality of normal vectors, it has a lower accuracy than the minimum entropy method, because we use histograms based on integral images (see fig. 4.2(b)). Experiments have shown the accuracy is similar with the entropy estimator, if kd-tree-based normal vectors are used (see fig. 4.2(d)). One difference is still that the non-RANSAC methods do not depend on normal vectors. Note that the extra runtime for normal vectors is not included in the typical runtime. Only the mean shift based approach estimates multiple estimates and/or hypotheses in one process step, while the MSAC one needs multiple iterations. Also, the CC-Ransac has to be executed up to 32 times. All methods show different linear runtime, which depends always on the amount of depth data per image. Here we assume a VGA depth image of 50% for the sake of simplification.
We want to emphasize that the algorithms are suitable for use on a robot since they all run with linear runtime. Many algorithms can be implemented on a GPU to obtain a faster processing time, like normal vector calculation, Manhattan system estimation with MSAC and minimal entropy. CC-RANSAC is difficult to exceute on a GPU since

the connected component analysis consumes far too much memory (typically 4k per thread on the GPU 2013). CC-Ransac needs at least 900k. The RANSAC/MSAC & particle filter based estimation methods are suitable for anytime application.

5 | 2D/3D Fusion and Applications

Today many robots are equipped with a different set of actuators and sensors, both with different characteristics such as odometry, IMU, Laser Range scanners, etc. While the choice of actuators and sensors strongly depends upon the robot application, in many cases there is a need for *coherent* and *robust* sensor data interpretation. Coherent sensor interpretation helps with decision-making within behaviors. The robustness of data interpretation regarding noise and uncertainty is needed for overall stability of a system. A possible solution is the Bayesian sensor fusion using *probabilistic state estimation* by considering the system as a *dynamic process* if prior knowledge about the underlying system is available. A common strategy within robotics is the Markov chain [7] sensor fusion by incorporating sensor (e.g. laser data) data and actions (e.g. Odometry). Another common method that is widely used in computer vision is to use sensor data as a Bayesian graph and apply fusion using graph cuts [123].

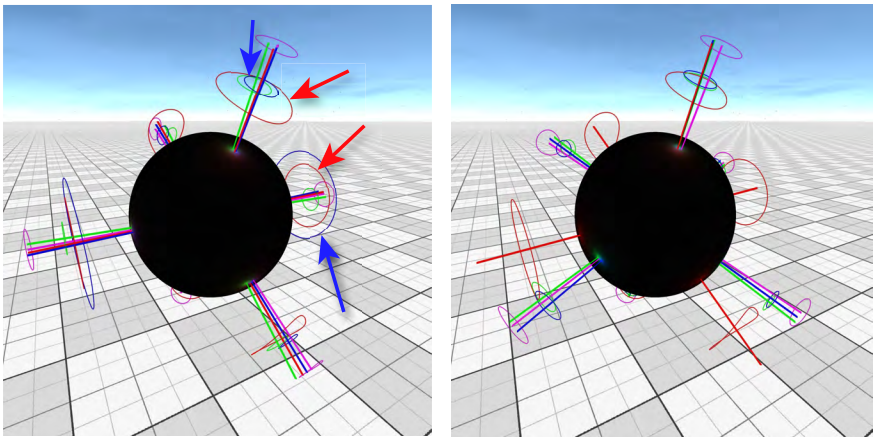
Within this work we present different approaches for Manhattan system geometry configuration estimation using the same 2.5d sensor data simultaneously. Some estimators are better than others are at giving results in some certain situations. For instance, the 2D Manhattan system geometry configuration estimator on a Microsoft's Kinect *can* be more accurate than a 3D estimator since the 2D image has 4.27× more resolution compared to the 3D data. In other cases, the 2D estimator is not effective because the robot is only observing a wall and only one line on the ground, while the 3D Manhattan system geometry configuration estimators will give results that are more accurate. So far, each estimator is fully independent form the other one and can run with a different runtime.

In this chapter the focus is also on the application side rather than only on pure sensor

processing as was handled in the previous chapters. In a first step, we apply sensor fusion to obtain a coherent Manhattan system geometry configuration. All obtained Manhattan system geometry estimates (global and local) are fused into one representation using Markov chain particle filters for tracking. The tracker also uses the robot's odometry in order to build a dynamic system and remove outliers. The result is a single Manhattan system geometry configuration that is used as input for the following modules. The next step is to combine 2D and 3D features in order to extract Manhattan system geometry . We use apply a common computer vision technique for this task such as superpixel over segmentation. The idea is to project all features into the 2D image and build a graph-like representation. The graph is reduced using the superpixels to individual sub-graphs for each feature that have different scales of the image. All graphs are combined into one graph using markov-random field (MRF) multi-label techniques and the Manhattan system geometry structure of the 2.5d data. An optional step is the use of geometric contained visual odometry and geometric constrained spatial-temporal Mapping. The basic idea is to consider only the translative motion of a robot since the rotation is known from the Manhattan system geometry. Visual odometry uses both 2D and 3D data, while the mapping is only applied to the depth data. With the mapping, we also apply Manhattan system geometry constraints on the map building itself by building joint maps for horizontal, vertical and non-Manhattan structure.

5.1 Manhattan system fusion

Within this thesis we proposed three different methods to estimate Manhattan system geometry configurations using either 2D images or 3D depth data from the 2.5d sensors. All methods work independently from each other and do not use techniques such as Markov chain tracking [7, 40, 106, 124] which expect the *minimum entropy* method, see section 4.3.2 for details. Each method delivers the *most believed* dominant Manhattan system geometry configuration if it can be detected. If available, the local Manhattan system geometry configurations are also delivered. Each axis of the estimated Manhattan system geometry has an associated uncertainty which is calculated according to the match of the estimated model with the data. In some cases, a local Manhattan-like structure is (mistakenly) estimated as the dominant one while an estimated local one reflects the true Manhattan structure. This can be the case with a settings/scences with a lot of clutter or if the robot is facing only a wall. In case of the



(a) Microsoft's Kinect (b) AIT Stereo Vision

Figure 5.1: Basic Concept of the Manhattan system geometry: The estimated Manhattan system geometry is shown in magenta as axis with its assigned uncertainty shown as ellipsoids (shown with the colored arrows in fig. 5.1(a)). The three inputs are shown in red for the MSAC method, green for the entropy method and blue for the 2D method. For the sake of better visibility, a Gaussian unit sphere is shown to illustrate the uncertainty of the three inputs

wall, about 90% or more represents one axis, and it is pure luck if the true dominant one can be found within the remaining 10%. Experiments have shown that in the most cases only one method[1] is able to detect a local Manhattan system geometry besides the dominant one, which is in 20% of all cases the true dominant one. This problem can be solved by applying tracking as sensor fusion and incorporating the robot motion if the Manhattan system geometry configuration was estimated properly before.

We adapted a method [37] that we originally proposed. The method was able to localize a *Nintendo Wii Controller* within a a-priori known environment. The method used multiple cues such as predefined beacons that were detected with the on-board Wii Controller IR camera and IMU data for estimating the 3D pose with all six degree of freedom using Bayesian particle filters. The system improved robustness by adding multiple cues from different sensor sources (and different sample rate). Each sensor measured three rotational axes (roll pitch, yaw) relative to its environment as three normal vectors i.e. N. The missing three translative (X, Y, Z) degrees of freedom were estimated by back-projecting the 2D image of the Wii camera to the a-priori 3D model of the environment, motion data and the fused rotational degrees of freedom.
For our adaption of the method, we only track the three rotational degrees of freedom

[1] two methods are able to detect multiple Manhattan system geometry

tracking with *only* the *dominant* Manhattan system geometry. We do not track the local Manhattan system geometry for the segmentation step that is applied after this step in the sensor chain. The translative ones are fused in the later segmentation steps of segmentation, visual odometry or mapping.

The basic idea is to track the *dominant* Manhattan system geometry configuration i.e. N using three orthogonal vectors $\vec{\mathsf{N}}_1, \vec{\mathsf{N}}_2, \vec{\mathsf{N}}_3$ while each of them points to particular X, Y, Z axis. Please note that we use the vector like an axis, which does not have orientation like normal vectors[2]. The vectors are rotated to an arbitrary roll pitch and yaw, which encode the Manhattan system geometry configuration. The Manhattan system geometry configuration is estimated using a variant of Markov Chain Monte Carlo filters to the data known as Monte Carlo Localization (MCL). MCL is a popular approach to self-localization of mobile robots introduced by Thrun et al. [125]. It is a Bayesian probabilistic method, in which the position of the robot is represented by a set of m weighted particles i.e. $\mathsf{N} = \{n_1, ..., n_m\}$. Each particle contains a "believed" position with an assigned probability π. The pose is estimated by using the observations as a likelihood function of the believed poses/states while MCL attempts to maximize the likelihood of the beliefs.

Let $\mathsf{N} \in (\alpha, \beta, \gamma)$ be an arbitrary Manhattan system geometry configuration with $\alpha, \beta, \gamma \in (\theta, \lambda, \varphi), \theta, \lambda, \varphi \in \mathbb{R}$. The three triples α, β, γ represent the rotational degrees of freedom in *Euler angles* where α is roll, β pitch and γ yaw. First, the three non-rotated vectors are created so each of one point one X, Y, Z axis. Then, the vectors are rotated according to each θ (α, β, γ) with first yaw, pitch and finally roll. We apply yaw first to avoid the *Gimbal lock* on this angle since it is the most dynamic angle for a moving robot on a flat ground. However, the extension to quaternion is straightforward. The remaining λ, φ are used to relax the Manhattan constraint which allows some tolerance to an almost orthogonal vector system, see fig 2.3 on page 17. The goal is to transform the vectors on a Gaussian unit sphere[3] to a meridian representation, which shows longitude and latitude. The two values are used as a delta offset to the resulting longitude λ and latitude φ. In order to have a bounded relaxation, we do not use λ and φ as longitude and latitude directly, but use a simple weighted cousins function e.g. $\dot{\lambda} = \zeta \cos \lambda$ with ζ being a normalizing constant. In practice, we use the same ζ

[2] the angle of an axis to a vector or other axis is $0 - 90deg$, see page 91 for details
[3] since the vectors are used as axis they do not have a length

for all angles so maximum delta is 5 degrees.

As with all Bayesian filters, MCL methods address the problem of estimating the state x [7] (we use the notion from Thrun et al. [7] for easier comprehension, instead of N) of a *dynamic system* from measurements and sensor/action readings. The control theory describes a *dynamic system* as an interaction model between a *controller* and its environment. Both entities interact with each other through *signals* y and *actions* u. *The signal* is the input of the controller and contains observations or measurements of the environment such as observational data like features extracted from images. The *action* is the output of the controller and is also a measurement such as data arising from accelerometers containing information about Wii motion. The Bayes filter assumes that the environment is *Markov*, which means that past and future data are conditionally independent if one knows the current state.
The implementation of a MCL requires two things: the motion model and the sensor model. The motion model is used to integrate the *actions* u to the current pose/state while the sensor model integrates the observations. The usual MCL algorithm works recursively in four different stages:

1. In the *prediction* stage, the motion model is used to integrate the *actions* u to all particles in that the particles are simply moved.

2. The observations are then used to *update* the weight π of the particles.

3. In the third stage, the weights of all particles are normalized to one weight.

4. In the last stage, the particles are *resampled* to get the posterior distribution.

Technically, the resampling discards particles with low weights and transforms them into a specific (random) particle with a high weight. In our implementation, we move to position of the new "offspring" particle in respect to the weight of the parent particle. This means that a low weight of the parent particle will result in a relatively high translation. Next, we give the models that are needed to implement the MCL filter.
The motion model $p(x_t|x_{t-1}, u_{t-1})$ represents the effects of action u_{t-1} on the robot's orientation to the Manhattan system geometry (here x_t) with respect to the orientation in the last time step. Our model is quite straightforward since we have odometry from the robot[4] itself. First, we use the rotational motion (speed) of the robot motion and apply to yaw $\gamma(\theta)$ directly by moving/rotating the particles, since the rotation of the

[4]The IMU is only used for ground truth

robot only affects yaw. We apply 10% Gaussian noise to the motion of each particle in order to cope with the uncertainty. The translative speed is applied to roll $\alpha(\theta)$ using 1% Gaussian noise. This is because we figured out that the ground tends to sag a bit, resulting in a slight jiggling motion of the robot. For pitch $\beta(\theta)$ we apply the acceleration of the translative motion part instead of the speed: We map $1\frac{m}{s} \equiv 10deg$ degrees with using 10% Gaussian noise. We apply noise on pitch since the robot tends to lean back when accelerating from a still position or lean forward when it stops due to its size and weight. Please note that we do not modify λ, φ since they result from the environment and not from the motion of the robot.

The *Sensor Model* is the heart of the MCL filter. It reflects the probability of measuring y_t if we assume that the state of the robot's orientation is xt. In our case, yt is a joint probability that results from the three different Manhattan system geometry estimation methods. Let $\Omega = \{\Omega_1, ..., \Omega_k\} \in (\mathsf{N}, \mathsf{N}_\sigma, \eta), \mathsf{N}_\sigma \in (\mathbb{R}, \mathbb{R}, \mathbb{R}), \eta \in \mathbb{R}$ be $k \in \mathbb{N}$ estimated Manhattan system geometry configurations (dominant and local ones) from the previous estimation methods (2D, MSAC, Entropy). N is the estimated Manhattan system geometry configuration as three vectors. For the sake of simplification, the delta longitude and latitude have already been applied on all $\mathsf{N}_{1,2,3}$). N_σ contains the variance in degrees for each $\mathsf{N}_{1,2,3}$ while η is used as a bias constant depending on the type of sensor and estimator. The bias η depends on the results of the previous sections, see table 5.1. For instance, the bias of MSAC is similar for both Microsoft's Kinectand AIT Stereo Vision (for the dominant and local Manhattan system geometry), since it is more likely that the true dominant one is mistaken as local one in contrast to the 2D method. Let $\vec{\mathsf{N}}_1, \vec{\mathsf{N}}_2, \vec{\mathsf{N}}_3$ be the mapped vectors from x_t^i from a particle i within the MCL filter. The idea is to sequentially match $\vec{\mathsf{N}}_1, \vec{\mathsf{N}}_2, \vec{\mathsf{N}}_3$ to each Ω and calculate a probability using a weight function based on the angular distance. So the mixed

Table 5.1: Different $_\eta$ bias values for Microsoft's Kinect and AIT Stereo Vision for the dominant and local Manhattan system geometry

	Microsoft's Kinect		AIT Stereo Vision	
	dominant	local	dominant	local
MSAC	1.0	1.2	1.8	2.0
Entropy	1.1	n/a	1.1	n/a
2D	1.2	1.8	1.0	1.5

probability (assuming[5] $k > 0$) is given with

$$p(y_t|x_t) = 1 - \prod_{j=1}^{3} \prod_{i=1}^{k} 1 + p(\mathsf{N}_j|\Omega_i) \tag{5.1}$$

with

$$p(\vec{\mathsf{N}}|\Omega) = \text{cauchy}(\ \underset{j\in(1,2,3)}{\text{argmin}} \vec{\mathsf{N}} \otimes \Omega(\mathsf{N}_j),\ \Omega(\mathsf{N}_{\sigma(j)})\Omega(\eta)\ ) \tag{5.2}$$

The operator $\otimes$ is a metric to calculate the angle of two axes using a *normalized* dot product. The normalization ensures that we use the vectors like axis. The result always ranges from $0-90$ degrees, since an axis does not have an orientation like a normal vector. The function $\text{cauchy}(x,\sigma), x, \sigma \in \mathbb{R}$ is given with

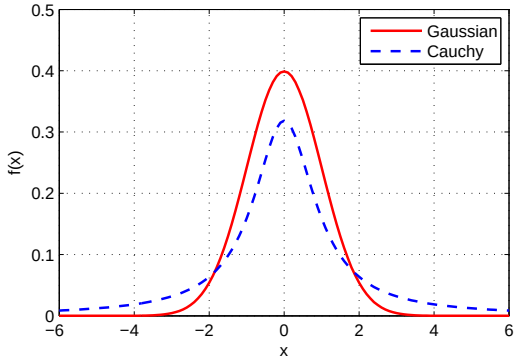


Figure 5.2: Comparison of the Gaussian (red) and Cauchy (blue) distribution with the same $\sigma = 1$. The Cauchy distribution is "less heavy" on its tails or is more flat at 3σ

$$\text{cauchy}(x,\sigma) = \frac{1}{\pi}\left(\frac{\sigma}{x^2+\sigma^2}\right) \tag{5.3}$$

as simplified Cauchy [50] function without the offset of the distribution. We do not use the Gaussian distribution here since the Cauchy distribution is more "relaxed" on its tails, see figure 5.2. The variance $\Omega(\mathsf{N}_{\sigma(j)})$ of the "best match" i.e. smallest angle, is multiplied with $\Omega(\eta)$ which is used as a bias depending on the sensor and estimation method. Figure 5.1 shows the overall concept of our approach: The red (MSAC), green (Entropy) and blue (2D) lines represent the estimated Manhattan system geometry Ω with their associated variances as circles[6]. The arrows on the left image point to the different variance for each axis and estimator. For instance, in figure 5.1(a) (blue arrows), the variance for the vertical Y axis and front horizontal Z axis is small for the 2D (blue)

[5] if $k = 0$ no sensor model is applied

[6] The variances seem to be distorted because they are also drawn on a slightly bigger sphere.

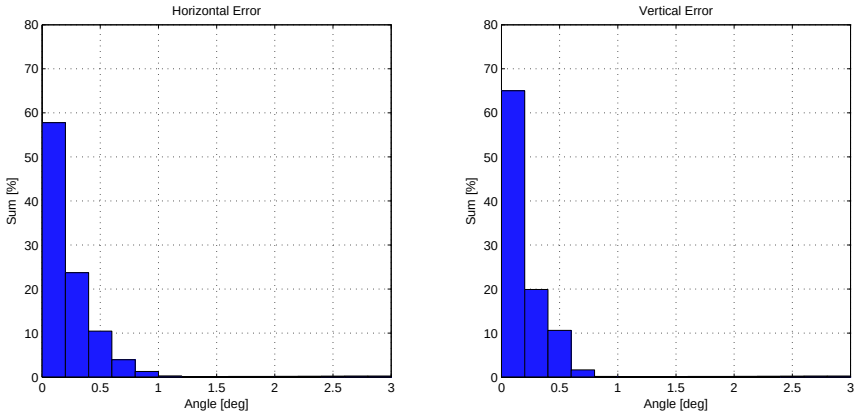
estimator, while it is far bigger on the X-axis. For the MSAC estimator (red arrows, fig. 5.1(a)), only the vertical Y-axis is bigger. The MCL estimated Manhattan system geometry is shown in magenta in the center of all Ω. Figure 5.1(b) shows an example of a falsely estimated Manhattan system geometry on the horizontal axis using MSAC on AIT Stereo Vision. In this case it has little effect on the MCL filters, because one of the three MSAC estimated axes is still estimated correctly. Another reason is that the MSAC estimator for stereo vision tends to give out a random horizontal axis if just the ground and no walls are seen in the stereo image, see 4.13(a) on page 94 for details.

Figure 5.1 also shows a plot of the cauchy distribution for each axis on the Gaussian sphere with its corresponding axis colors (the MCL estimated Manhattan system geometry is not shown) i.e. using the red color channel for MSAC etc. The plot has been normalized for better visibility. One can see the Ω estimators form a corona on the true Manhattan system geometry. Please note that both fig. 5.1(a) and 5.1(b) are not taken from the same scene and time stamp.

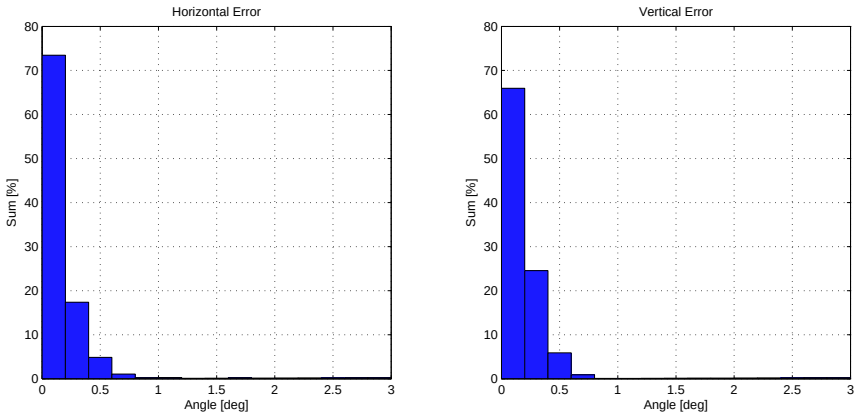
It belongs to best practices[7] that the robot stands still and then moves for initialization. This ensures a faster convergence of the particles, since the state space is reduced to rotational degrees of freedom only. The initialization of the system itself is done in "good old" Monte Carlo fashion; all hypotheses are randomly distributed in the state space $x_t \in \mathsf{N}$ except for the λ, φ to reduce the state space. Analogous to traditional Monte Carlo approaches, our state is obtained by building the weighted average of all particles. In the case of no available sensor information by the Manhattan system geometry estimators, the motion model is only applied to the particles. Here, the particles are just moved with the motion model and not resampled. For the normal resampling case, we add noise on all angles θ (roll, pitch and yaw) depending on the normalized weight in the sensor model. We use a simple linear ramp function to obtain the noise gain depending on the particle weight, i.e. 0.005 degrees for the best (highest weight/lowest error) and 0.2 degrees as the lowest for roll, pitch and yaw. We apply noise on λ, φ with slightly different strategy: Instead of applying the same noise level on all three axes, we choose two random axes as long as they're not identical. On the first one, we apply[8] $0.001 - 0.01$ degrees (first is highest weight) and $0.01 - 0.1$ degrees on the second one. We use this overall arbitrary strategy to avoid a chaotic

[7] It also works with a moving robot, but needs more time to converge

[8] Please note that we give the corresponding values in degrees, although λ, φ are used in a normalized cos bias function



(a) AIT Stereo Vision (Cumulative error seen on the plot on the left $\approx 98.7\%$, right $\approx 98.8\%$)



(b) Microsoft's Kinect (Cumulative error seen on the plot on the left $\approx 98.9\%$, right $\approx 97.9\%$)

Figure 5.3: Histogram of the angular error from both Microsoft's Kinect and AIT Stereo Vision from all tours using the proposed method. The error of the vertical and horizontal is shown as separate plots. One can see that the average error is significant lower than from all three previous proposed methods

dynamic within the 9 dimensional state space which is related to simulated annealing with particles filters [53].

We use 500 particles our experiments. Figure 5.3 shows an error histogram for both Microsoft's Kinect and AIT Stereo Vision separately for the vertical (y axis) and horizontal axis (X/Z axis) combined. One can see that the average error is significantly lower than for all previously proposed methods. Microsoft's Kinect performs overall better since MSAC only works robustly for vertical axes with AIT Stereo Vision. The average error for AIT Stereo Vision is 0.32/0.28 degrees (vertical/horizontal) and

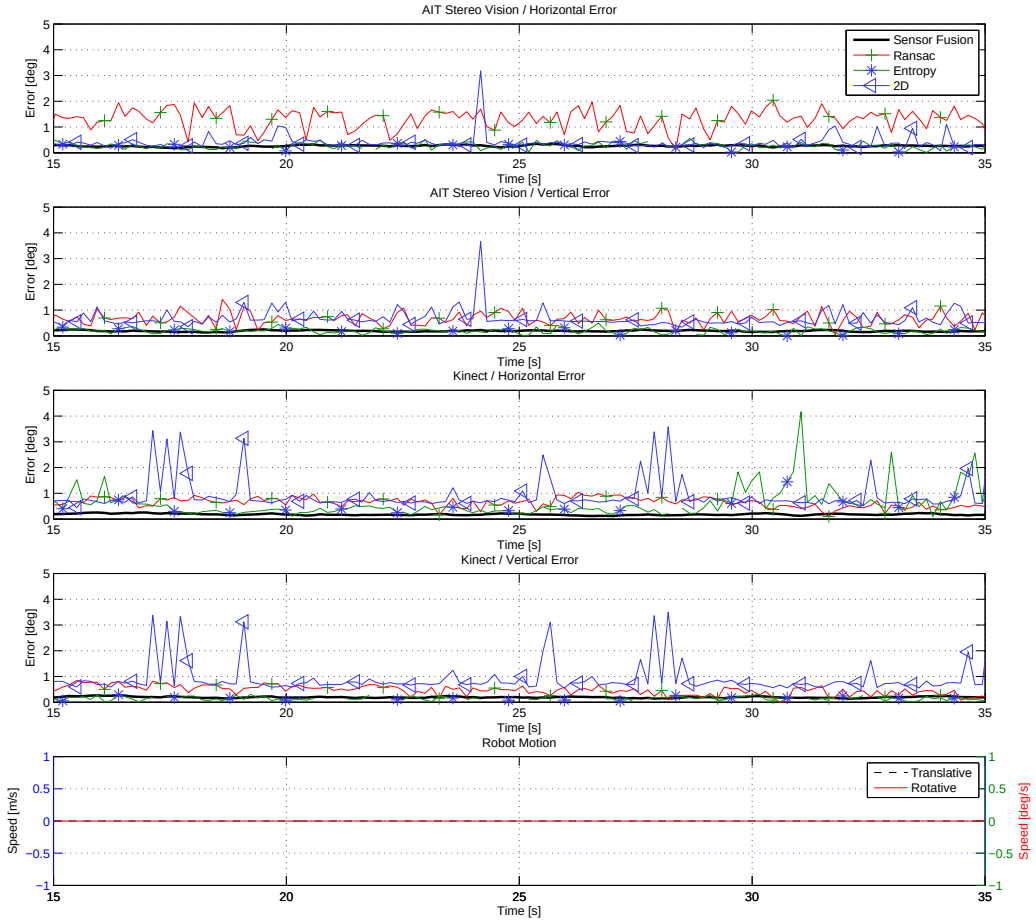


Figure 5.4: Example for Still Robot: The first two columns show the error AIT Stereo Vision the 3rd and 4th one Microsoft's Kinect. The last column shows the speed of the robot. The error per estimator is shown in the first four columns, for both the vertical and horizontal axes

0.23/0.22 degrees for Microsoft's Kinect. Please note that the average error for the ground truth data is 0.2 degrees.

Figure 5.4 gives an example for a non-moving robot: The robot is placed in a corridor with all Manhattan system geometry axes visible. One can see that the error for MSAC is relativity high for the horizontal axis with AIT Stereo Vision sensor while it is reasonable for the vertical ones: This is due to the ground being observed in stereo, but without having proper walls. With Microsoft's Kinect, we notice a low sensitivity with the 2D estimator (see the peaks) due to the auto exposure within Microsoft's Kinect. We notice that the brightness tends to "pump/burst" with Microsoft's Kinect

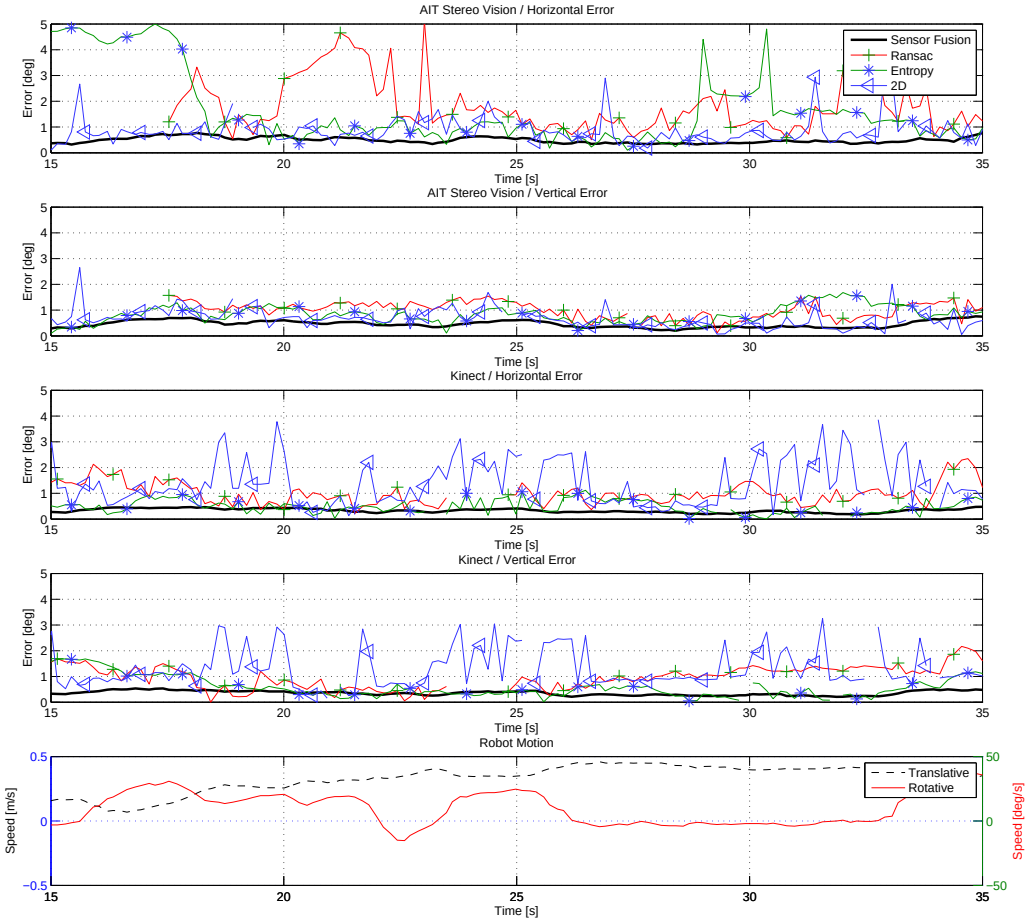


Figure 5.5: Example for Moving Robot: The first two columns show the error for AIT Stereo Vision. The 3rd and 4th one show the error for Microsoft's Kinect. The last column shows the speed of the robot. The error per estimator is shown in the first four columns, for both the vertical and horizontal axes

if the overall room illumination is low. The entropy estimator shows similar results for both sensors. Please note that we do not show the plots during initialization.

With figure 5.5, we show the more common case of a moving robot: The robot moves along a non-structured corridor, faces a white wall (15s), and turns into a room (20s). The robot moves all the way into the room with a slope, turns back (25s), and leaves the room (33s). One can see that MSAC has difficulty estimating the walls on the horizontal axis with AIT Stereo Vision while facing a white wall. The same is true for the entropy estimator since almost no 3D data is available. The Manhattan sys-

tem geometry is still tracked using the robot motion, 2D estimator, and the estimated vertical axis. Another reason is the "Gaussian noise" style behavior of the MSAC estimator, while the entropy estimator can be stuck in a local minima (15-18s) with the horizontal axis (i.e. yaw). With Microsoft's Kinect, we notice a similar behavior by the 2D estimator with the still robot except 2D is a bit more "noisy". Overall, Microsoft's Kinect performs better because it is an active sensor and just as sensitive to white walls as AIT Stereo Vision. Please note on the plots that not all estimators deliver output at each time step. This, for example, is shown by the 2D estimator with Microsoft's Kinect.

The code runs within 1ms on our test computer using an optimized multi-threaded code. This fast runtime is due to the few needed operations within the sensor model. The bottleneck is the calculation of the shortest angle between two axes for all particles. On average, $k = 3\ \Omega$ is used for calculation. 95% were dominant Manhattan system geometry configurations.

5.2 Oversegmentation

Before we dive into the actual segmentation, we consider first the *oversegmention* [55] of the 2D image. The extracted features such as planes, plane hypothesis and lines

(a) Quickshift [126] (b) Graph-based [127] (c) SLIC [128]

Figure 5.6: Comparison of different superpixel segmentation strategies using local pixel grouping 5.6(a), graphs 5.6(b) and deformable grid 5.6(c). Images are taken from [128] and show superpixels at two scales: large and small

are all parametric and are in either 2D or 3D. Each feature can be projected into a 2D representation using the original 2.5d information without loss of any information since we do *not* use a fully dense 3D model. The actual fusion of all features is done on a graph that is used like a 2D image. We do not apply a segmentation per pixel because of noise, resolution or runtime reasons. The idea is to use small groups of pixels with similar properties instead of applying a so-called oversegmention in order to increase robustness and runtime speed.

A common method in computer vision is the use of an oversegmentation with the so called ”superpixels” [126–131]. The use of superpixels has become quite popular in the computer vision literature within the last decade. The main objective is to locally merge pixels into meaningful clusters or ”superpixels”, pixels with similar color, texture, appearance or shading. With many approaches [127, 128, 130, 131] it is assumed that true object boundaries are mostly (but not necessarily) represented by the boundaries in the ideal case of the superpixels, if the object's size is large enough (> 5 pixels). In fact, the ideal case depends strongly on the parametrization of the superpixels. One common metric is the minimum or fixed number of pixels within a cluster, b_{max} for example. If the object size in the image is unknown, a superpixel ”pyramid” scale is used in many cases using different minimal number of pixels per cluster (with graph based methods e.g. [127, 129, 131]). Figure 5.6 shows examples for the three major superpixel strategies: (1) by clustering local pixels with similar appearance 5.6(a), (2) using a minimum spanning tree on the graph (the image is considered as a graph) 5.6(b) and (3) using a deformable grid with fixed size 5.6(c). One can see the difference in the boundaries of the superpixels in approximating the boundaries depending on using ”small” or ”large” scale superpixels.
In this work, we use the fast *Minimum Spanning Tree* method by Felzenszwalb [127] (5.6(b)) since it provides a good trade-off in quality and speed. It is almost the de facto standard with many computer vision approaches. Methods using clustering give similar results but are computationally more expensive (e.g. $\approx 16\times$ with Quickshift [126])). Methods with a deformable grid are similar in speed ($\approx 1.3\times$ slower with [128] and $\approx 1.2\times$ [130, 131])), but strongly depend on the position of the grid. For instance, if a transition of two structures (e.g. white wall and blue ground) is placed in the *center* of grid cell, it is not approximated by deforming the grid on it in many cases. This is an issue because, with our environment, we face the problem that most of the structure appears distorted in perspective since we use an camera setup that looks down like a

(a) $sq_{color}\alpha$ (b) $sq_{depth}\beta$

(c) $sq_{lines}\gamma$ (d) $sq_{color}\alpha + sq_{depth}\beta + sq_{lines}\gamma$

Figure 5.7: Using our Superpixel to apply oversegmentation on the corridor scene from Microsoft's Kinect using different cues, namely using color5.7(a)5.7(d), depth 5.7(b)5.7(d) and lines 5.7(a)5.7(d)

human. So, many transitions of structures are not placed near the *edge* of the grid cells at certain (real world) depths in the image[9], in contrast to the commonly used image databases used for image segmentation. Here a frontal view of the environment/objects is common, such as those that are used with the PASCAL[10] object databases.

The minimum spanning tree method proposed by Felzenszwalb [127] works as follows: First, the image is blurred with a Gaussian kernel τ similar to the well-known canny [30] edge detector. Next, the image is transformed into a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where each node $v_i \in \mathcal{V}$ reflects a pixel in the image and the edge $(\sigma, v_j, v_k)_i \in \mathcal{E}$ from node v_j to

[9] perspective is a common issue with image segmentation

[10] http://pascallin.ecs.soton.ac.uk/challenges/VOC/databases.html

v_k with weight $\sigma \in \mathbb{R}$. The edges are drawn for each node from its direct surrounding neighbors in all four directions: north, south, east, west. A non-negative weight σ is assigned to all edges of the graphs based on the absolute difference of the two neighbor pixels according to a key difference, color for example. The node v_i is used as a label and is set to "unassigned" for all nodes. All edges of the graph are sorted in ascending order using the weight as a metric. The following procedure is now applied to all edges $(\sigma, v_j, v_k)_i$ in the list (lowest σ first): if the two neighboring nodes v_j, v_k are both not assigned to a label, then both are set to a new unique label. In the event that one of the nodes is assigned, but the other one is not, the unset label is set to its neighboring one. In case both nodes are assigned to two different labels, the following procedure is used: first, the number of nodes that are assigned to the label are counted for both v_j, v_k. If both v_j, v_k exceed a certain number defined by bmax the edge is discarded. In the other case, the sum of all edges that have the same label for vj or vk for both nodes is calculated. The label with the highest sum is set to all nodes for vj , vk until only one label remains. This entire procedure is applied since no more edges are in the list. Felzenszwalb [127] uses an additional post-processing step to assign a label with a smaller number of nodes than b_{max} to the biggest label within the neighborhood within all nodes with the same label.

The original approach uses color to calculate s for an edge σ for an edge (σ, v_j, v_k)

$$\sigma = \sqrt{\mathrm{sq}(v_j, v_k, red) + \mathrm{sq}(v_j, v_k, green) + \mathrm{sq}(v_j, v_k, blue)}$$

with

$$\mathrm{sq}(v_j, v_k, c) = (\mathrm{B}(v_j)_c - \mathrm{B}(v_k)_c)^2$$

and $\mathrm{B}(v_k)$ as a function to obtain the color of an node in RGB. We extend this idea by adding two more images to improve the performance of the oversegmentation: depth information from the 2.5d sensor and the extracted lines from section 3.4 on page 50. For greater ease, we notate the color part without square root as $sq_{color} = \mathrm{sq}(v_j, v_k, red) + \mathrm{sq}(v_j, v_k, green) + \mathrm{sq}(v_j, v_k, blue)$. The actual square root is applied in a later step.

First, we introduce a new sq_{depth} term to incorporate depth cues in the oversegmentation. Experiments have shown that the use of Normal vectors from 2.5d is not efficient, since it can lead to artifacts for non-flat objects and corners. Instead, we use the po-

sition of the corresponding voxel (x_j, y_j, z_j) (z represents depth in the image) from a node v_j and the disparity value d_j . Note that it is possible that a d_j disparity value is not defined ($d_j = \infty$) because no depth data is available. So we define

$$sq_{depth} = \begin{cases} \eta \lfloor d_j - d_k \rfloor + (x_j - x_k)^2 + y_j - y_k)^2 & if d_j \neq \infty \wedge d_k \neq \infty \\ 0 & else \end{cases}$$

which prevents undefined values for missing depth data. η is a normalizing constant to scale the disparity to the voxel space equally within a close range i.e. 3m and depends on the sensor.

The last new term is sq_{lines} that uses the extracted lines from section 3.4. Each line contains an averaged sum of all pixel gradients from which the line is created. These come from, for example, the Gabor filter and polar histogram. The method is straightforward. We create a new image with the same size as the RGB image and draw all found lines using the averaged pixel gradients as color and then apply a Gaussian blur. The blur kernel uses half as much as the one from the oversegmentation image τ. The term is given using B′ as function to obtain pixel values from the newly created image with

$$sq_{lines} = (\mathrm{B}'(v_j) - \mathrm{B}'(v_k))^2$$

We put all parts together using additional gains α, β, γ that are used as bias depending on the sensor, for example:

$$\sigma' = \sqrt{sq_{color}\alpha + sq_{depth}\beta + sq_{lines}\gamma}$$

Note that each component is blurred by the same amount (i.e. RGB, depth and line image) before the graph is built. Depending on the sensor, we use a different set of gains, see table 5.2. For instance, with AIT Stereo Vision, depth data can be less reliable due to the passive nature of the sensor in contrast to Microsoft's Kinect,

Table 5.2: Gains used for individual components used in this work for superpixel

	α (RGB)	β (Depth)	γ (Lines)
Microsoft's Kinect	0.3	0.4	0.3
AIT Stereo Vision	0.5	0.2	0.3

while the color image by AIT Stereo Vision is more reliable despite its lower resolution. Figure 5.7 shows an example of each individual part and the result. One can see that the mixture model follows the corners of the walls (transition wall and glass) far better than RGB only.

The use of superpixels is a relatively slow computation in comparison with other components since it has to recalculate many pixels many times. On our test setup, our approach needed about $\approx 105ms$ for the AIT Stereo Vision and about $\approx 450ms$ for Microsoft's Kinect for one scale. This is due to the nature of the spanning tree algorithm that can only be partially parallelized since the output of the "next" output depends on the previous one. It should be noted that the original code from Felzenszwalb [127] is not optimized for speed. It uses many recursive functions and consumes 80% overhead just for calling functions and 15% merely for sorting the edges (measured with gcc profiler utility 4.7.2)

5.3 Segmentation using MRF based multi-labeling

So far, in this thesis we extracted individual features from either 2D or 3D data/cues from the 2.5d sensor. The individual Manhattan system geometry configurations were fused into one global Manhattan system geometry configuration in the previous step. All features, such as lines, planes, and the plane hypothesis were extracted with one configuration for the sake of consistency. The next step is to combine all features into an overall segmentation using recent technology in computer vision technology the Multi-label Markov Random Field (MRF) technique [123, 132]. The idea to assign each object/pixel/superpixel a unique label. This is in order to use a metric to minimize the number of labels by joining two or more labels into one. The segmentation is processed on the 2D image using the previous extracted superpixels (see fig. 5.8 since all features can be back-projected into the 2D image[11]. Now we want to cover the segmentation process in detail by defining all parts of it.

To label the pixels on a global level, one must consider the following factos: prior information about possible planes, hypotheses, orientation to a Manhattan system, 2D geometry and the relationship between neighboring superpixels simultaneously. We formulate the problem in a fully probabilistic framework when searching for a maximum

[11] This is possible because we use 2.5d data instead of dense 3D data

posterior (MAP) configuration of the Markov Random Field [123] for multi-labeling [132]: When dealing with a labeling problem we have a set of observations $\mathcal{P}$ (e.g. superpixel) and a set of labels $\mathcal{L}$ (e.g plane/line/orientation/etc hypotheses). The goal is to assign each observation $p \in \mathcal{P}$ a label $f_p \in \mathcal{L}$ such that the joint labeling f minimizes the objective label function $E(f)$. We assume a graph $\mathcal{G} = \langle \mathcal{P}, \mathcal{E} \rangle$ consisting of a discrete set $\mathcal{P}$ of objects and a set $\mathcal{E} \subseteq \binom{|\mathcal{P}|}{2}$ of pairs of those objects.

An instance of the Max-sum problem is denoted by the tuple $(\mathcal{G}, \mathcal{L})$, where the elements $D_p(f_p)$, $V_{pq}(f_p, f_q)$ and $h_L \dot{\delta}_L(f)$ of g are of alignment costs or qualities. The quality of a labeling f is defined

$$E(f) = \overbrace{\sum_{p \in \mathcal{P}} D_p(f_p)}^{\text{data cost}} + \overbrace{\sum_{pq \in \mathcal{N}} V_{pq}(f_p, f_q)}^{\text{smooth cost}} + \overbrace{\sum_{L \subseteq \mathcal{L}} h_L \dot{\delta}_L(f)}^{\text{label cost}}$$

where h_l is the non-negative label cost of label l, and $\delta_L(f)$ is the corresponding indicator function

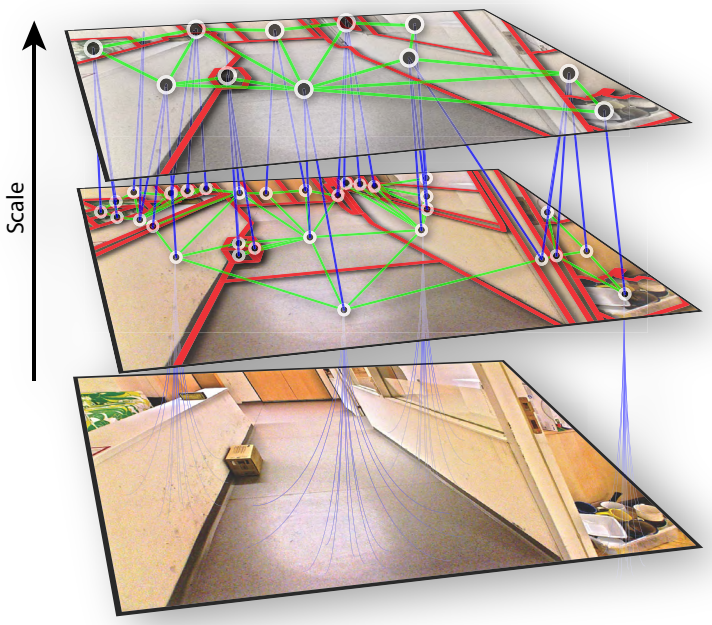


Figure 5.8: Multi-scale Segmentation of an image using different scales of pixel grouping. The image on the bottom is the lowest scale. This refers to all of the individual pixels. As the scale increases, more pixels are grouped into increasingly larger clusters. Note that a larger scale of an image does not necessarily represent a better oversegmention since the size of the objects in the image is previously unknown

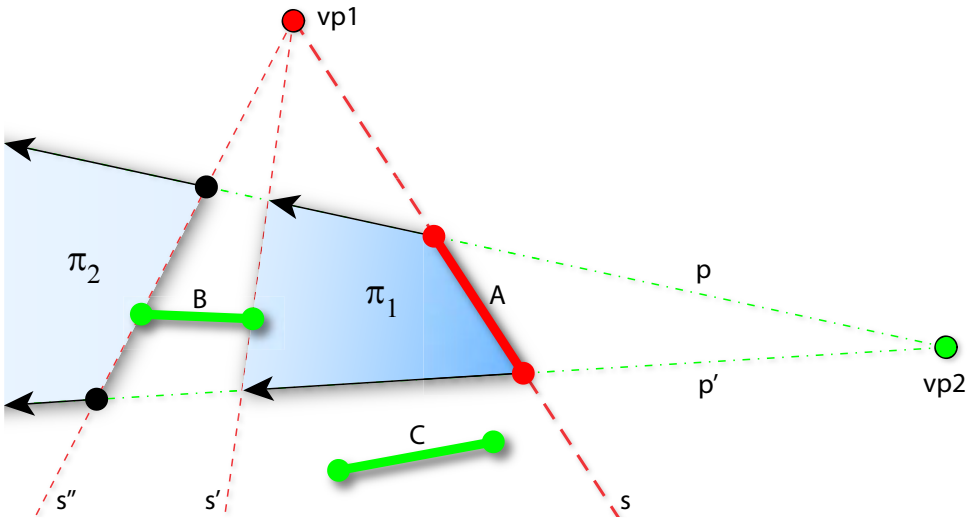


Figure 5.9: Principle of Vanishing Point Sweeping: The planes $\pi_1, .., \pi_n$ are constructed from the endpoints of line A (member of $vp1$) using $vp2$. The sweep is constructed so other lines (not belonging to $vp1$) are avoided within the planes

$$\delta_L(f) = \begin{cases} 1, & \text{if } \exists p : f_p = l \\ 0, & \text{else} \end{cases}$$

A common approach in the computer vision literature is to use three labels [88] for every major axis instead of using multiple labels per major axis. One reason for not using more labels is that the general labeling for more than three labels often leads to NP-hard solutions with a standard Max-flow min-cut graph [123, 132]. In this work, we use an MRF multi-label approach proposed by Delong et al. [132], which solves the multi-label problem within polynomial runtime for an arbitrary number of labels. The polynomial runtime is achieved by using a different strategy for the multi-labeling than common approaches like using a fixed number of labels. The goal is to use the MRF to reduce the number of labels by merging them using the $E(f)$ function. The strategy starts with a reasonable number of labels of all extracted plane and plane hypotheses and uses the *label cost* together with the *data term* as a metric.

All features are projected back into the image space for the use within the three cost terms. For planes and the (plane) hypothesis (chapter 4) we use a straightforward strategy for pre-segmentation: A voxel $v \in \mathbb{R}^3$ is assigned to a plane or hypothesis (i.e. $P_x = \{\rho_{x,1}, ..., \rho_{x,n}\}, P_y = \{\rho_{y,1}, ..., \rho_{y,n}\}, P_z = \{\rho_{z,1}, ..., \rho_{z,n}\}$) if it intersects the plane or hypothesis within certain thresholds. A voxel is not assigned exclusively to plane

since we use multi-label graph cuts. The distance, using the native sensor space, to the plane is the applied metric for the later graph cut using a weight function. For the extracted lines $\omega_n \in \Omega$ (chapter 3) we use a strategy based on the approach from Lee et al. [133], shown in figure 5.9. A common strategy is to group line segments to rectangular planes π such as [50, 88, 134] from two different vanishing points. In many cases, this method can lead to NP hard solutions and high dimensional search space since line segments can be occluded [135]. The idea is to construct plausible orientation maps instead of the actual planes. We "sweep" one orientation map using lines from two vanishing points in order to obtain a structure that can be oriented to the third redundant vanishing point. For instance, the sweep lines from the X and Y axis will result in a possible Z orientation if no Z sweep is within the same area. The algorithm is depicted in figure 5.9 for the example of a "sweep" for line A using its parent vanishing point $vp1$ and "sweep" vanishing point $vp2$. The first two lines p and p' are constructed so that they intersect the endpoints of the line A and end in $vp2$. Next a "sweep" line s is constructed so that it intersects at $vp1$ and overlaps line A. The idea is to "sweep" s away from $vp2$ (in our case to the left) until it intersects (here s') with a different endpoint of another line not belonging to $vp1$ and lying between p,p' or on p or p' (lines outside p,p' are not considered e.g. C). A new plane π_1 is constructed from the convex hull of all intersections from s,s',p,p'. The sweep line is moved further (here s'') until it intersects with another endpoint of a non-related line and the sweep line does not intersect (e.g. belonging to $vp2$ like in the figure). The sweep is continued until the sweep is outside of the image, such as in the figure where the sweep intersects with the left image border, and results in π_2. For the sake of simplification, the figure shows only the "sweep" away from $vp2$. In practice the sweep is proessed in both directions (not exceeding $vp2$). Let $P_{x,y}$ be a sweep from lines that belong to the x ($vp1$ in the example) vanishing point and the sweep vanishing point ($vp2$). A map of particular orientation, like Z , is obtained with $R_z = (P_{x,y} \cap P_{y,x})$. If it is exclusively Z, it is also added with $O_z = R_z \cap \neg(R_x \cup R_y)$ by excluding the maps for X and Y . However, the other maps are obtained the same way. Due to the selectivity of criterion, a pixel can belong to one of the three major orientations or to an unknown one.

Now we discuss the Graph entities: graph $\mathcal{G}$ is built upon the previous overly segmented image using two scales. The use of superpixels significantly reduces the number of objects in the graph compared to building the graph directly on the pixel grid. The

superpixels represent objects, with the set $\mathcal{P}$ in the graph and edges. The set $\mathcal{E}$ is established between every two neighboring superpixels. The number of nodes (labels) K is set to the number of observed planes or the hypothesis, along with four undetermined labels to mark ambiguous label assignments. These labels allow the solver to mark the places where there *is not enough* information to decide which plane the superpixel belongs to. The individual *undecided* labels are X, Y, Z and undetermined.

With some graph cut methods, the smoothness term is used to formulate the similarity of two neighboring graph entities i.e. superpixels. With multi-label graph cuts this term is encoded in the weight $\rho[p,q]$ of link of two neighboring $p,q \in \mathcal{G}$ graph entities. Similar to the other cost term, a low cost will lead to more often "graph-cuts," while a high value will lead to a higher number of connected graph entities. The weight function consist of three parts: color similarity, texture similarity and the orientation of overlapping pixels along the two superpixel borders. The color is similarly calculated in the Lab Color [136] space as a parametric model with the mean $\mu \in \mathbb{R}$ and variance value of $\sigma\mathbb{R}$ for all thee color channels as tuple $\mathcal{C} = \{l,a,b\}, l,a,b \in \{\mu,\sigma\}$ of all pixels within the superpixels. We do not use histograms due to the low pixel number seen with some superpixels. The similarity of the colors is obtained individually for each mean and variance per color channel, based on the Bhattacharyya distance [106]

$$\rho_{color}(p,q) = \prod_{p,q \in \mathcal{C}} -\ln\left(BC(p,q)\right)$$

with

$$BC(p,q) = \int \sqrt{p(x)q(x)}\, dx$$

which in our case is

$$D_B(p,q) = \frac{1}{4}\ln\left(\frac{1}{4}\left(\frac{\sigma_p^2}{\sigma_q^2} + \frac{\sigma_q^2}{\sigma_p^2} + 2\right)\right) + \frac{1}{4}\left(\frac{(\mu_p - \mu_q)^2}{\sigma_p^2 + \sigma_q^2}\right)$$

according to Coleman and Andrews Coleman and Andrews [137]. The texture similarity of two superpixels is estimated using the well-known *Local binary patterns* approach proposed by He and Wang [138]. The idea is encode a binary pattern within the surrounding pixels of a center pivot pixel. Each bit represents if it is either brighter or the same value with "1", or if it is darker with "0" than the pivot pixel. We use four neighborhood pixels (North, South, East, West) for the bit encoding which is due to the small sizes of some superpixels. The similarity is obtained by building histograms

on the pattern of both neighboring superpixels and using the Bhattacharyya *metric* [106]:

$$\rho_{texture}[p,q] = \sum_{u=1}^{16} \sqrt{p_u q_u}$$

with $\sum_{u=1}^{16} p_u = 1$ and $\sum_{u=1}^{16} q_u = 1$. The last term ρ_{edge} encodes how many pixels of the shared boundary of two pixels are directed to any of the three vanishing points (Manhattan system geometry in 2D projected back). This is the most important term since it indicates a possible split of two superpixels belonging to two different geometries. Let $\mathcal{C} = \{b_1, ...b_n\}, b_i = \{x, y\}, x, y \in \mathbb{R}$ be the pixels of the shared boundary. Let $\mathcal{E}' = (e_1, ..., e_m)$ be the gradients maps that have been introduced on page 54, see fig. 3.22. So we get

$$\rho_{edge}[p,q] = \frac{1}{n} \sum_{i=1}^{n} max_{u=1}^{m}(B(u, x_i, y_i))$$

using $B(u,x,y)x, y \in \mathbb{R}, u =\in \mathbb{N}$ as the function to obtain the value of pixel x, y of the e_u gradient image. The entire weight function is given as

$$\rho[p,q] = \alpha(1.0 - \rho_{color}) + \beta\rho_{texture} + \gamma\rho_{edge}$$

using $\alpha, \beta, \gamma \in \mathbb{R}$ as normalizing constants. For both AIT Stereo Vision and Microsoft's Kinect we choose $\alpha = \beta = \frac{1}{4}, \gamma = \frac{1}{2}$. In order to favor splits based on pixel gradient and geometry over color or texture.

The term $V_{pq}(f_p, f_q)$ describes the smoothness between two labels p, q as the cost to assign q to p. The function itself must be injective, $V_{pq}(f_p, f_q) \neq V_{qp}(f_q, f_p)$, which is necessary for our MRF variant by Delong et al. [132]. In our implementation we set cost function V_{pq} so that a label q from a plane or hypothesis is set to p of an another hypothesis. This is as long as both the hypotheses as well as the planes are assigned to the same orientation and have almost the same position $|pq|$. For example, if the plane or hypothesis is y-aligned, it's the height, and if p has less support by voxels of

the plane or hypothesis, i.e. inliers, then q:

$$V_{pq}(f_p, f_q) = \begin{cases} 0, & \text{if } p = q \\ 1, & \text{if } p_{orientation} \neq q_{orientation} \\ cost_{pq} \cdot \Delta_{pq} \cdot q_{inliers}, & \text{if } p_{orientation} = q_{orientation} \\ cost_{known}, & \text{if } p_{orientation} = \text{"undecided"} \\ cost_{undetermined}, & \text{if } p_{orientation} = \text{X, Y, Z} \\ cost_{undetermined}, & \text{if } q_{orientation} = \text{"undecided", X, Y, Z} \end{cases}$$

with $cost_{pq} \gg cost_{undetermined} > cost_{known} \geqq 1$. We use the distance of the plane and hypotheses position of the corresponding labels p, q as weight of a metric. For instance, this is used for

$$\Delta_{pq} = \begin{cases} 2.0 - \frac{|pq|}{c} & \text{if } |pq| \leq c \\ 0 & \text{otherwise} \end{cases}$$

a threshold c. We use $cost_{known} \geqq 1$ since we want to allow the MRF to remove false positives from the graph, i.e. false labeled x-axis oriented planes or hypotheses surrounded by other z-axis-oriented planes and hypotheses. The condition $p_{orientation} = q_{orientation}$ can combine labels with the same properties while the graph -based representation ensures that this is only applied if the source superpixels are near each other in the source image.

The data term $D_p(f_p)$ encodes the quality of assigning a label f from the set $\mathcal{L}$ to an object or superpixel p in the graph. The quality measures how the superpixel is oriented to a specific plane or hypothesis. We use a joint probability to encode both 2D and 3D features. The data term $D_{p3}(f_p)$ for the 3D features like planes or a hypothesis is as follows: for each f_p we count the number of voxels/pixels with the same label as f from the pre-segmentation. Note that the label f corresponds to the plane or hypothesis. Next, the number of pixels p_n is normalized to p_m and set as cost to

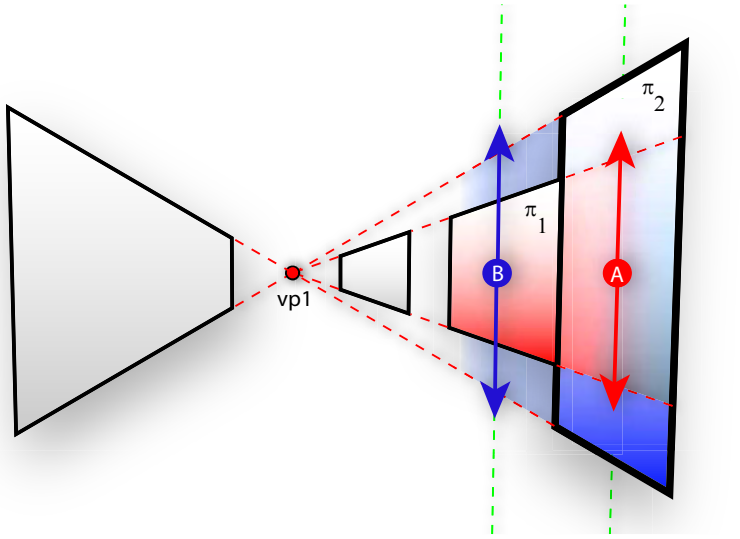


Figure 5.10: Principle of the 2D data cost metric: Both planes $\pi 1, \pi 2$ are projected (red dashed lines) to test if A, B are plausible members. The plausibility check for both A, B uses scan lines to search for the boundaries using the orientation map

$D_{p3}(f_p)$

$$D_{p3}(f_p) = \begin{cases} W(p_m) \cdot cost_{data}, & \text{if } W(p_m) \geq \lambda \wedge W(p_n) \geq \gamma \\ cost_{undetermined} & \text{if } f_{orientation} = X, Y, Z \\ \eta cost_{undetermined} & \text{if } f_{orientation} ='' \, unknwon'' \\ 0, & \text{otherwise} \end{cases}$$

With $cost_{data} \gg cost_{pq}$, $\eta = 0.1$ as normalizing constant and

$$W(p) = \begin{cases} p \cdot cost_{data}, & \text{if } f_p \text{ is "undecided"} \\ p, & \text{otherwise} \end{cases}$$

where λ and γ are thresholds and $cost_{data}$ is a normalizing constant that can prevent false positives if $cost_{data} > 1$. In our implementation we use $\lambda = 0.1$ and $\gamma = 10$ since we set the "minimum superpixel size" Felzenszwalbs [127] superpixels segmentation to 100 pixels. In our experiments $cost_{data} = 1.5$, which produces fewer but more definite labels.

The data term $D_{p2}(f_p)$ encodes the cost for the 2D features for the label f for superpixel p. First, we estimate the boundaries of the parent plane and hypothesis f since, with the exception of the undetermined labels, the label originates from it. The basic idea is to project the plane and hypothesis in 2D using its boundaries and its associated

vanishing point to estimate if a superpixel belongs to the plane. Figure 5.10 illustrates this idea; the plane π_1 is projected using vanishing point $vp1$, shown as dashed red line, to test if superpixel A is a possible member. "A" is a possible candidate since it's within the projected plane. A second test estimates the plausibility of the projected plane and the real "plane" underneath A. Since we do not extract "planes" in 2D, we use the orientation map instead to test if the border of the projected plane corresponds to the one present in the orientation map, using the orientation of the projected plane. This is processed with a scan line, shown as green dashed line, which is then oriented to the second vanishing point. In our case "A" is unlikely to be within the projected π_1 since the borders do not overlap. The same holds true for B and π_2 while A, π_2 and B, π_1 are plausible.

First, we obtain the minimum and maximum values for both plane dimensions, such as the two associated vanishing points, see fig. 5.10, in the Cartesian 3D space. To simplify, we assume only one dimension, seen with red dashed lines, while the algorithm is applied indeed onto both. We use a similar robust strategy like the image contrast enhancement seen with 3.1 on page 33. A cumulative histogram is built using the minimum and maximum values as boundaries. The new minimum value represents at least 1% of the data and 99% for maximum. The values are shown as red dashed lines and are oriented in 2D to the first vanishing point. After that, we construct a scan line for p using the second vanishing point and the geometric center of p. The purpose of the scan line is to search for boundaries in the orientation map. Instead of looking for a corner on the scan like an edge filter, we use a strategy that we originally proposed in [40]. This concept was to search for an edge using a sliding window and three histograms normalized with the sum of one: a model histogram h_{obj}, a foreground histogram h_{for} or and a background histogram h_{bck}. The histograms are built upon the entities of the orientation map; therefore, it has four bins, while the model histogram h_{obj} is simply the orientation of the plane/hypothesis itself. For example, this is seen if the orientation is X, because the only used bin is also "X". The idea is to slide with a window in both directions at point o of the scan line, shown as green dashed line, beginning at the superpixel center p itself. We build two histograms along the line at point o: one towards h_{for} and one backwards at h_{bck}. Both histograms are created from the name number of pixels (e.g. 20). The goal is to define a metric so that the foreground as well as the background histograms separate each other. Using the

previous Bhattacharyya metric we get

$$\rho[h_{obj}, h_{for}, h_{bck}] = \sum_{u=1}^{4} \sqrt{h_{obj}(u) h_{for}(u)} \cdot \sum_{u=1}^{4} \sqrt{h_{for}(u) h_{bck}(u)})$$

An edge on the scan line is considered as found if $\rho[h_{obj}, h_{for}, h_{bck}] < \nu$. In case h_{for}, h_{bck} are similar, meaning no corner, the Bhattacharyya metric will result in a value ≈ 1, if they are different in ≈ 0. We use $\nu = 0.1$ for both AIT Stereo Vision and Microsoft's Kinect. Both found corners are projected on the plane and the one with the smallest distance Δ_{hist} to the projected plane (π_1, π_2) boundaries is used; however, if no corner is found, the image corner is used. Let χ_f be the geometric center of the plane and χ_p the geometric center of the superpixel projected on the plane in the Cartesian space. The cost term is $D_{p2}(f_p)$ is obtained using the Δ_{hist} as a metric and biasing it using the distance χ_p, χ_f with

$$D_{p2}(f_p) = \text{cauchy}(\Delta_{hist}, \sigma_\Delta) \cdot \text{cauchy}(\chi_f - \chi_p, \sigma_{fp})$$

using the cauchy(x, σ) function from page 127. Since $\Delta_{hist}, \chi_f, \chi_p$ results from the projection on a plane, the distances are in the Cartesian space. We use $\sigma_\Delta = 0.1, \sigma_{fp} = 2$, which results in a strict scan line for edge detection and liberal handling during depth plane projection.
The final data cost is given with

$$D_p(f_p) = 1.0 - \psi D_{p2}(f_p) + \omega D_{p3}(f_p)$$

using $\psi, \omega \in \mathbb{R}$ as weight factors. Since AIT Stereo Vision tends to produce data with less dense depth, we bias the 2D feature with $\psi = 0.65, \omega = 0.35$ and for Microsoft's Kinect with $\psi = 0.35, \omega = 0.65$. For *undetermined* labels, we build a histogram on the orientation map from all pixels in p. The cost per histogram bin is set to the corresponding label using a normalizing cost of 0.1 to ensure that the undetermined labels are unbiased in the graph cut.

The *label cost* term $h_L \dot{\delta}_L(f)$ is used to penalize each unique label that appears in f within $E(f)$. We use the cost h_L to express the amount of certainty of the label f with a lower cost reflecting a higher amount of certainty. Since we use the MRF to minimize the number of labels by combining the costs of fused labels, a solution is created with minimal overall cost. We use the orientation maps to obtain h_L and

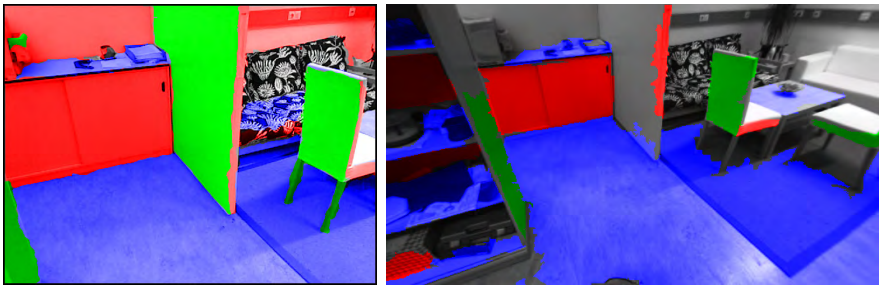
penalizing a label f by considering the consistency of the marked pixels, resulting from the planes/hypotheses, as well as the orientation map. Let $\mathcal{O} \in \{X, Y, Z, unknwon\}$ be n values of orientations from the map that overlap with the pixels that intersect with the plane or hypothesis within a certain threshold with label L. Let $f_O \in X, Y, Z$ be the orientation of the Label L. The idea is to count the orientations that violate the plane or hypothesis in order to penalize them. The term is given with:

$$h_L = \sum_{i=1}^{n} \begin{cases} \lambda & \text{if } O_i = unknwon \\ \eta & \text{if } O_i \neq f_O \\ 0 & \text{otherwise} \end{cases}$$

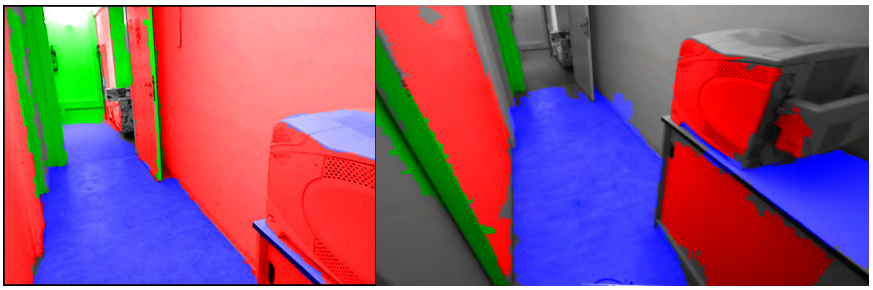
with $\lambda << \eta$ and $\eta = \frac{1}{n}$. We use $\lambda = max(0.01\eta, 0.0001)$ for both AIT Stereo Vision and Microsoft's Kinect. We use the same metric for the labels (X, Y, Z and undetermined) but use $\mathcal{O}$ for the entire image. We penalize the undetermined label less than the other ones (see λ) to ensure that the undetermined label is penalized correctly compared to the other true labels.

Figure 5.11 shows some sample results using our method. One can see that the use of orientation maps improves the results even for superpixels with no depth data, see page 20. In some cases, only the orientation can be extracted, but not the actual depth itself. We use a heuristic technique (from [50]) in order to estimate the depth for those planes by assuming that all planes are connected to each other. If the depth of a ground plane or a hypothesis is known, then the depth of the surrounding connected planes can also be obtained by assuming that they stand on the same ground. In practice we apply this heuristic technique if at least 90% of the pixels are directly connected to a ground plane that has been properly estimated.

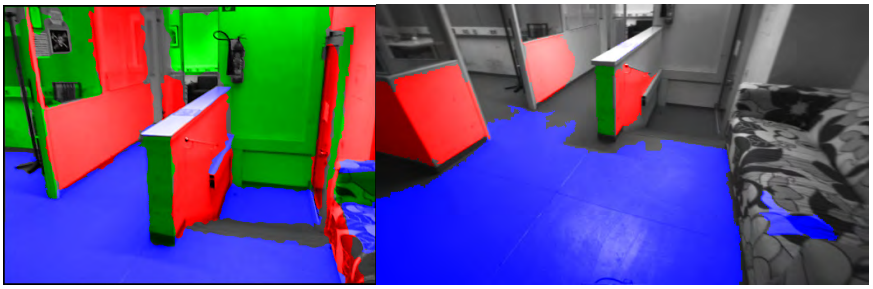
The algorithm proceeds with an average runtime of 480ms on Microsoft's Kinect and with 376ms on AIT Stereo Vision using **no** multi-threading. One limitation of our approach is the calculation of the data cost term. This is around 80% since we did not apply any optimizations on the histogram-based approach. The second limitation is that the graph cut itself has a 10% CPU load.



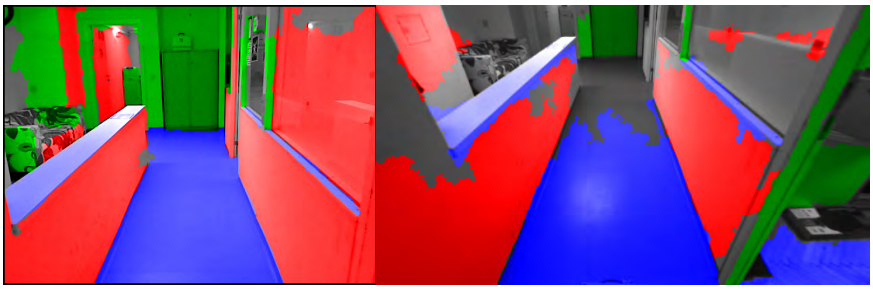
(a) Living Room



(b) Printer

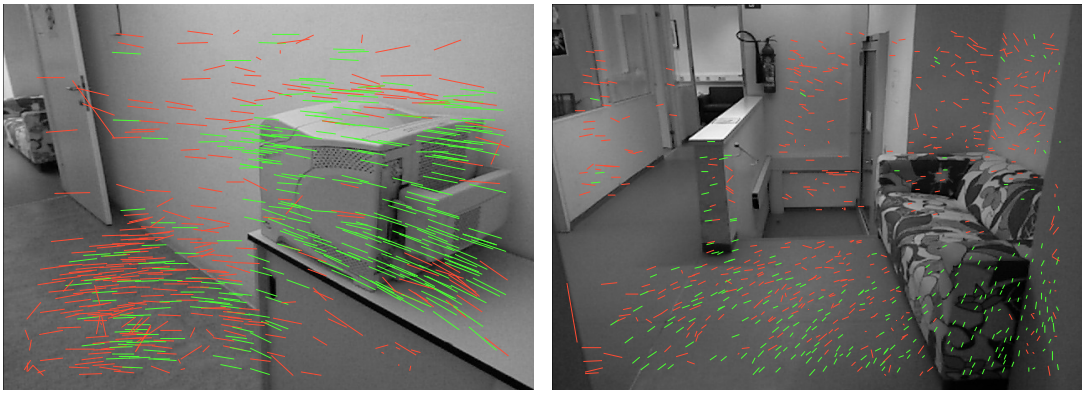


(c) Lobby



(d) Corridor

Figure 5.11: Sample pictures of our MRF multi-label segmentation method. The color encodes the orientation of the pixels. On the left is Microsoft's Kinect, while AIT Stereo Vision is seen on the right



(a) Rotative Motion (b) Translative Motion

Figure 5.12: Visual Odometry using KLT features and RANSAC motion estimation using Microsoft's Kinect. Outliers are shown in red, inliers in green

5.4 Applications

An optional step used as an independent application is geometricly-contained visual odometry and geometricly-constrained spatial-temporal mapping. The basic idea is to consider only the translative motion of a robot since the rotation is known from the Manhattan system geometry . Visual odometry uses both 2D and 3D data while the mapping is applied only to the data on depth. With the mapping, we also apply the Manhattan system geometry constraints on the map building itself by building joint maps for horizontal, vertical and non-Manhattan structures.

5.4.1 Geometric Constrained Visual Odometry

The visual odometry [22, 97, 101, 139] is used to estimate the relative translative motion of the robot toward the estimated Manhattan system geometry configuration. This occurs since the rotation is already obtained in the previous step using MCL. We use standard KLT [140] 2D features to track the motion using both 2D data on depth. The tracking on the 2D grayscale image is completed using the GPU within $>$ 1ms with Zach et al. [140] implementation. The depth estimation for each point is straightforward because we use the xyz coordinates from the depth data. The motion estimation is completed with a one-point RANSAC matching with known depth and correspondence per point. One tracked point is chosen as a model. For instance, this is seen with the relative motion of the point in 3D as well as with the previous point while the rotation is already removed. As with the standard RANSAC, the solution

with the most inliers is selected and implemented for motion estimation, see figure 5.12. Only tracked KLT points with valid depth are used. We use a maximum distance of 10cm for inliers, 80 iterations for AIT Stereo Vision, and 50 for Microsoft's Kinect. See Pöufs et al. [22] for implementation details. The entire runtime for our visual odometry is $> 1ms$ since only several iterations are necessary with RANSAC , with an average of 60 tracked KLT features with depth correspondence. A noticeable effect using KLT with AIT Stereo Vision is that the KLT features are also correspondent to depth with high certainty. While previous Manhattan system geometry estimators like the entropy method (see chapter 4.3.2 on page 97) rely on a dense depth map, the KLT tracker favors opposite segments like dirt on a white wall or ground, usually resulting in a very small amount of depth data. The output of the visual odometry is combined with the robots odometry using a standard extended Kalman filter and a differential drive model, which is the golden standard in robotrics [55, 95, 97, 101]. In the next step we describe how we use the visual odometry in order to build temporal maps (see next chapter for results).

5.4.2 Geometric Constrained Spatial-temporal Map

So far within this thesis we only used sensor data from a single time stamp. Sensors are limited within a certain range of depth perception and/or field of view. With our used sensors, the Microsoft's Kinect delivers more dense depth data than with AIT Stereo Vision, but is clearly more limited in field of view and depth range. One popular solution within robotics is the use of *spatial-temporal maps* [55, 141] similar to the *Kinect Fusion* [142] approach. The basic idea is to build a local map that has a date of expiry or is only used within a certain driving range of the robot. The map is maintained while to robot moves and data is added to it. These kind of maps are commonly used for robot navigation and obstacle avoidance [95] and have been utilized by Engel et al. [139] for monocular vision. It is additionally used in the popular open source software *Robot Operation Software* "ROS" (`www.ros.org`) developed and maintained by Willow Garage. Our idea is to store depth and 3D data from arbitrary waypoints, all 1m, which are thenapplied to build a map that is valid for 15m of robot movement. This can be used to project 3D data such as when the robot visits the same place again in order to complete parts of the depth data due to far range, absorbing materials or reflections. Another application is the offline sensor-processing with more accurate sensor data. This is seen with Steinbruecker's et al. [143] method for dense reconstruction without a running robot, which can be used as map for self-localization.

With our experiments, we noticed that a robot usually revisits a place within the same tour, like when entering and leaving a room. One example is shown in figure 5.13: the map from a previous run[12]) is projected back to the start position of the robot. See olufs et al. [22] for more details.

In one of the first steps, the 2.5d data is converted into a voxel representation including the previously obtained normal vectors, see chapter 4.1 on page 77. Then we remove the rotation of the three major axes from the voxels data such as the roll, pitch, and the yaw according to the previous fused Manhattan system geometry configuration. After this, all voxels are then labeled according to the most plausible relationship to Manhattan geometry, see figure 4.11 on page 88. For labels we use "X Axis", "Y Axis",

[12]only for the sake of better visibility

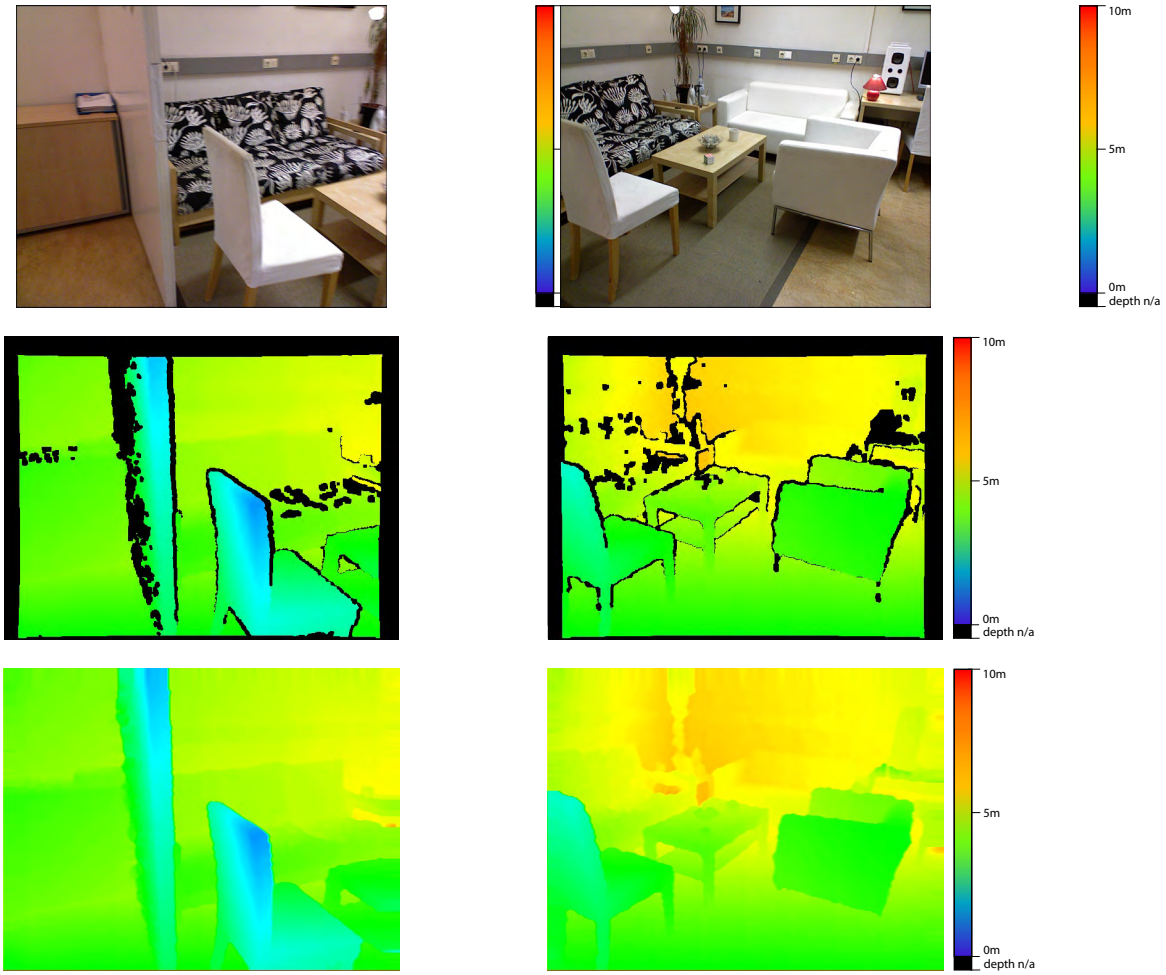


Figure 5.13: Examples for Spatial-temporal map projected back (bottom picture) to 2.5d using Microsoft's Kinect (top picture)

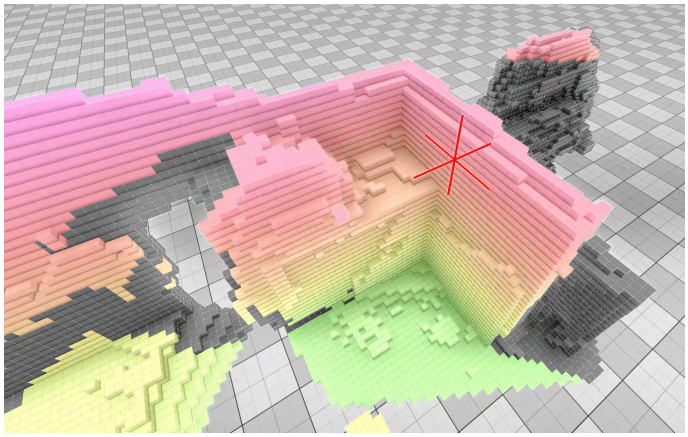


Figure 5.14: Sample visibility check for the current robot position shown as a red cross. The dark cells are not visible from the current position

"Z Axis" and "None" to demonstrate the relationship to an axis. This is estimated using the normal vectors of each point. If a normal vector has almost the same orientation to a major axis within a specific threshold, it is assigned to that axis. We use a threshold of 5 degrees. All normal vectors that do not belong to an axis are assigned as "None". We use an additional heuristic technique to ensure that the voxel dataset contains enough of a Manhattan-like structure. The heuristic technique simply counts the ratio λ of assigned voxel to the unassigned one ("None") and the ratio μ the axis with the most voxels to the second one. The voxel data is only used if $\lambda > 10$ and $\mu < 5$. This heuristic technique results in a rejection rate of 90% for AIT Stereo Vision because for most parts, only the ground is observed and for some parts of the wall, such as $\mu = 8$. Although furniture can usually be properly detected by AIT Stereo Vision , for the most parts only the ground parallel structure is observed. Experiments

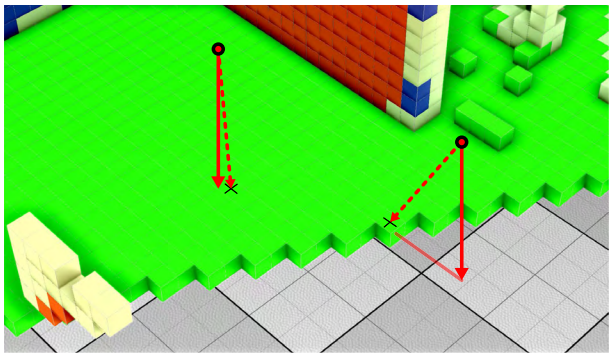


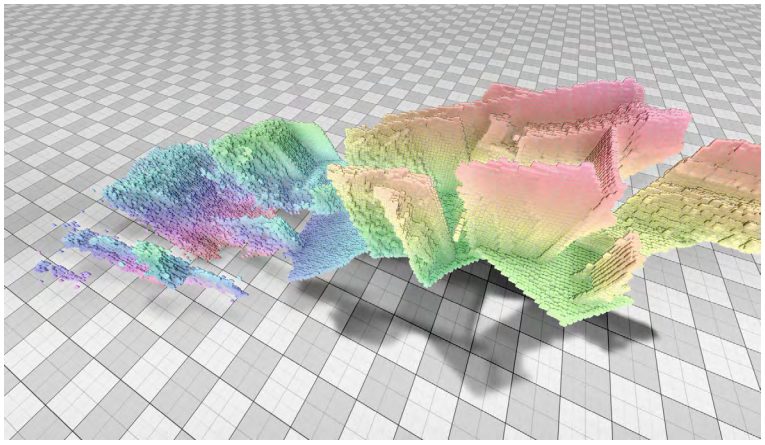
Figure 5.15: Principle of the facelets with our ICP variant. The dashed line is the Euclidean distance to the closed point, the solid one to the facelet. The red circle denotes the actual point (Y Geometry), the black cross shows the closed point in the octree grid

have shown that the ICP can only be applied if the thresholds are met; therefore, the approach is **only** used for Microsoft's Kinect.

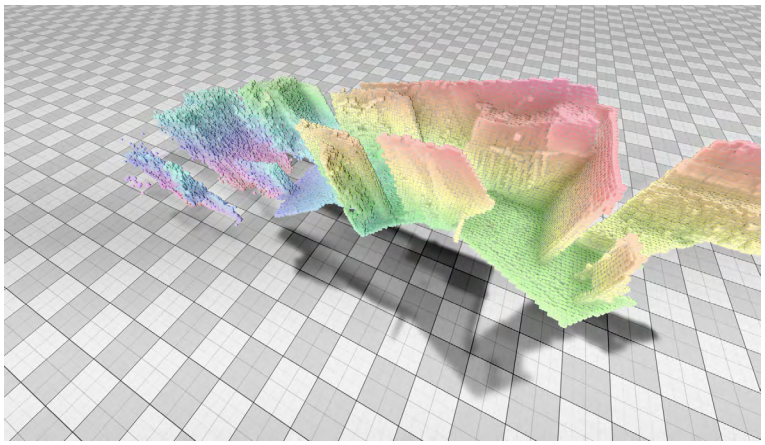
For the generation of the map we used a modified version of the well-known iterative closed Point ICP algorithm with additional RANSAC outlier rejection. ICP consists of three steps to register a set of samples to a model. First, the samples are associated with the model by the nearest neighbor (in Cartesian space) criteria. Then we estimate the transformation of the points to the model and finally the input points are transformed. The entire algorithm is repeated for a fixed number of iterations or the error converges. In our variant, we associate only samples to the models with the same label. We use the previously obtained visual odometry to move the robot in the ICP state space, since mapping is a continuous process and we use the last transformation of the previous step for initialization of the next one.

The map is represented using octrees with a fixed minimum resolution that is efficient regarding memory usage and computational power. Since the runtime of the ICP depends upon the amount of input, the map (or model) points, and the voxels, it is feasible to limit the size of the map instead of representing them in a kd-Tree-like structure [144]. We use the ICP with a maximum of 20 iterations and an octreemap with a resolution of 2.5mm. Due to this, we remove the rotation from the 2.5d data. The ICP converges within 5-7 iterations in many cases using visual odometry, seen in the previous chapter. We use the octree map implementation proposed by Wurm et al. [145].

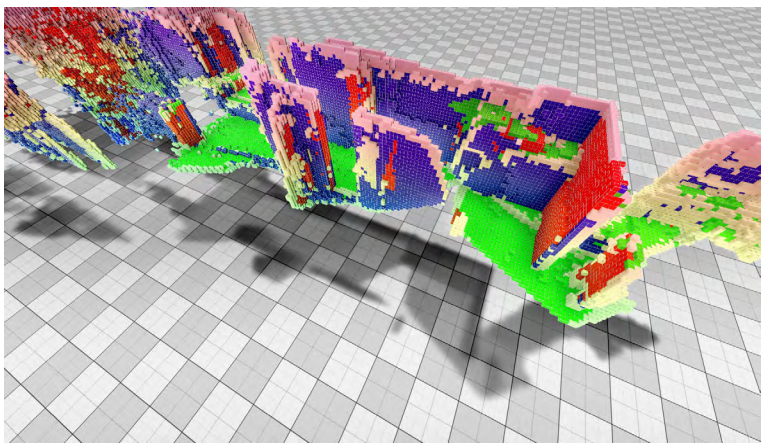
As an extension of the traditional ICP algorithm, we use an additional visibility check for each voxel relative to the current position, see figure 5.14. Since we use a voxel representation with octrees such a visibility check is feasible regarding runtime, similar to the 2D beam model used in Monte Carlo Localization. We use an exhaustive ray cast visibility check [136] implementation with a complexity of $O(n \log n)$ that is applied to all voxels. Alternatively, the usage of a sub-model with a visibility check will result in a smaller kd-tree for the nearest neighbor assignment, but is only an approximate measurement. Depending upon the size of the map, the visibility check can increase the point-to-point association up to 30 times since not all voxels have to be considered for the ICP. On the other hand, the runtime of the visibility check depends upon the size of the map. The usage of the visibility check adds 20% more runtime, but in



(a) standard ICP (PCL)



(b) standard ICP (PCL) with removed Roll, Pitch and Yaw



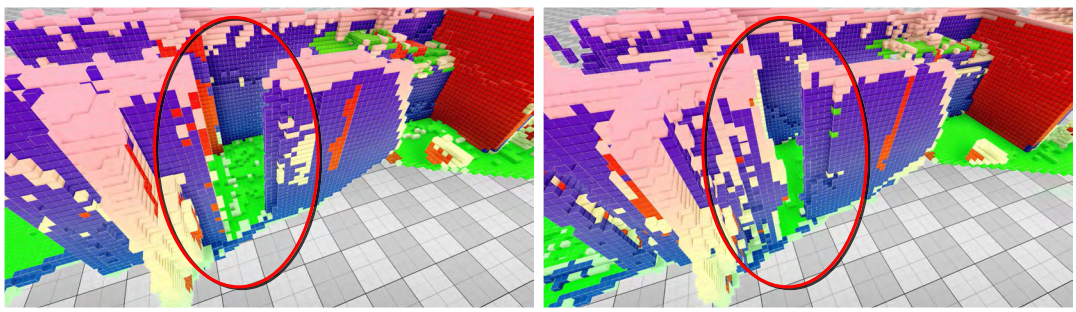
(c) our ICP variant

Figure 5.16: Generated map after 30s from figure 5.12 with 5mm cell size for better visibility. The unconstrained ICP shows an angular drift. Best viewed in color, height and geometric relationship is shown in color-coding with Microsoft's Kinect.

results in more defined maps.

Another extension is the usage of *surface facelets* [146, 147] for the nearest points assignment. The purpose is to utilize a surface model that is obtained from the model point cloud using factors such as triangulation [50] or marching cubes [148]. The points are then assigned to the closed points on the surface or the facelets, instead of to the points of the model. While ICP with surfaces is computationally expensive on dense point clouds, it is relatively easy to implement it with our ICP variant. All map points have labels according to their geometry. For instance, a point aligned to the Y-axis results in a Y plane within the same, or similar height, as the point, see figure 5.15. An exhaustive search like triangulation is not needed since all of the labels are already geometricly constrained. In the case of the label "None" we use no facelets due to performance issues such as average ICP usage. Facelets improve the mapping by making the far structure is more robust [147] and resulting in a smaller rotational error. It also improves the handling of points that are "not yet mapped" as long as a new observed structure is connected to previous observed structure.

The generated map of a sample tour, more than 32m, is shown in figure 5.16 (see figure 5.12 on 149 for a sample view from the robot POV using the ICP with and without geometric constraints. The visual odometry was used for all examples, without facelets and visibility check for comparable results. The ICP without the geometric constraints generates maps with the typical angular drift. This is average for non-holonomic robots and sensors with a relatively short sensing range. Holz et al. [149] state that the angular drift using the ICP can be reduced with a wide field of view. For example, a. 360 degree sensing up to 80m. The ICP with geometric constraints shows almost rectangular maps since the Manhattan geometry estimation reduces the angular drift significantly. One can see that non Manhattan-like structures are also



(a) with visual odometry (b) without visual odometry

Figure 5.17: Impact of visual odometry on the mapping process with Microsoft's Kinect

mapped and segmented.

Figure 5.17 shows the impact of visual odometry on the mapping process. While the door is visible with visual odometry, it almost vanished without the well-known ICP initialization problem. The major drawback of our method during the segmentation is that it depends upon a properly detected Manhattan configuration. While it is possible to detect a non-existing Manhattan system, it cannot be always guaranteed that room structure is fully orthogonal to each other. The use of a relaxed Manhattan constraint is one possible solution as to why the major axes are almost 90 degrees orthogonally to each other. Another issue that we encountered was open, or slightly open, doors. Since we currently only estimate the dominant Manhattan structure, doors are not yet segmented when they are not closed or aligned to another major axis. Depending upon the visible amount of the door in the data, namely more than 20%, the door will be segmented into small coherent areas.

Another drawback would be that there were artifacts in the map due to falsely classified areas. This is because of the normal vectors used for segmentation. For instance, a wall with a distance of 5m can produce inconsistent vectors due to limitations of the depth sensor resolution, while a wall in 2m shows reasonably normal vectors. Such artifacts do not influence the performance of the geometrically constrained ICP since we use independent maps for each geometry. If the 5m away wall is closer in the image, it will be correctly segmented. The artifacts can be removed from the map, since they are usually not cohered areas in the map. One other issue is that of round objects. They seem to be aligned with at least two geometries according to their normal vectors. Here, the use of a local geometry analysis can reject these kind of artifacts. Another issue is are corners and edges, see figure 4.11 on page 88. They are usually marked with "No Membership", because we use normal vectors from an integral image. One way to improve the results is to use the MRF segmented image instead, but this will result in a longer (not at frame rate) runtime.
The registration using ICP for all data points of the Kinect needs 250ms. We use an octree map with a resolution of 2cm. The update of the octree map is at 5ms relatively fast per iteration. The bottleneck is the ICP registration itself at 70% and 25% for the visbility check.

5.5 Related Work

The fusion of multiple sensors is a traditional discipline within robotics and computer vision literature e.g. Bayesian filters. One key aspect is the feature representation before and after the sensor fusion. While the robots odometry was widely used for mapping of laser data [7], the use of only one sensor gained interest within the last few years. One example is *visual odometry* [98] which incorporates sensor data over time and fuses it together by estimating a virtual camera movement. Another trend is building maps based on 2.5d sensors and visual odometry as is shown in [141, 142]. The approach from Fioraio and Konolige [150] uses ICP with bundle adjustment for frame-to-frame camera tracking using depth data from Microsoft's Kinect. Another method is used by Steinbrucker et al. [151] by using energy-based minimizer for frame-to-frame tracking.

Newcombe et al. [142] proposes the so-called "KinectFusion" approach which gained high interested in the robotics and computer vision literature. The idea is to build and maintain a volumetric voxel-grid using depth data from Microsoft's Kinect using voxels and normal vectors. The map is built by iteratively reconstructing the surface using a volumetric model and multi-scale ICP for alignment. The camera is tracked within the voxel-grid similar to visual SLAM approaches [98]. Newcombe et al. [142] uses a GPU implementation and is therefore limited to $10x10x10m^3$ in voxel grid size since the memory of the graphics card is used. Bylow et al. [141] extends the approach by using an error minimizer using signed distance functions instead of multi-scale ICP for alignment. Whelan et al. [152] uses additional IMU data from the 2.5d sensor and extends the approach by incorporating color information in the voxel grid.

With the previous approaches, no Manhattan system geometry system constraints were applied. Furukawa et al. [15] uses a reconstruction approach that first detects the Manhattan system geometry system from structure arising from motion depth data and extracts a structure hypothesis that is aligned to it. In a second step, the plane hypothesis is transferred into a volumetric voxel grid and a graph cut is used to extract the actual map. The graph cut uses a bias function to favor Manhattan system geometry structure over non-Manhattan system geometry structures. Another approach is proposed by Xiao and Furukawa [16] assuming that the Manhattan system geometry configuration is already known: The depth data is transformed into horizontal 2D

maps instead of using volumetric voxel grids. Each grid extracts a local plan using free space constraints to extract the ground plan. The individual 2D plans are stacked and transformed into a 3D map that fits to the Manhattan system geometry system constraints. The free space constraints were also used by Guo and Hoiem [153] to estimate the ground plane from a Manhattan system geometry system from Microsoft's Kinect using geometric and scene context knowledge.

The approach of Flint et al. [154] uses 2D, 3D and multi view cues for Manhattan system geometry scene understanding, for example in extracting the layout of the room with prior known Manhattan system geometry configuration. The idea is to use Manhattan system geometry primitives like convex, concave walls or occluded walls. Hypotheses about the room layout are built using the primitives which are extracted from the 2D image using vanishing points and 3D planes. The layout is obtained using dynamic programming and also incorporates terms like occlusion. This approach was adapted by Taylor and Cowley [155] using Microsoft's Kinect: Similar to our approach, an overestimation is applied and vertical planes are estimated using Manhattan system geometry constraints. Lee et al. [133] uses a similar approach to Flint et al. [154] based only on 2D images and plane sweeping with no resulting scale information. Ramalingam and Brand [134] showed a generalized version based on line grouping for a feature-based 3D reconstruction (also with no scale).

Another related technique to our approach is the work by Saxena et al. [135] with "Make3D". The approach extracts partially Manhattan system geometry constrained 3D structure with no scale from monocular images using learned classifiers from ground truth 3D data. The image is transformed first into a graph representation with explicit occlusion handling (assuming that structure can be occluded by other structure) which is obtained using superpixel overestimation. Saxena et al. [135] assumes that the ground plane is on the bottom of the image and does not explicitly model Manhattan system geometry in the classifiers giving labels like "ground", "sky", "wall" etc. The final structure is extracted using the linear programming MRF solver. Hoiem et al. [111] is similar, but uses different classifiers. Saxena et al. [156] also proposed a variant of the approach that also uses stereo vision for 3D structure estimation. Micusik et al. [157] uses a similar strategy to the one used in [135] by incorporating vanishing points from Manhattan system geometry instead of learned classifiers. A simplified graph representation is also used since only labels for the three major axis (X,Y,Z)

are used with no explicit occlusion handling. Liu et al. [158] extends this approach by introducing a bias term that favors box-like structures within the image. Silberman et al. [159] uses an approach that adds conceptional knowledge about a room layout and furniture to the graph e.g. that tables have in the most cases a certain height.
Some authors also exploit Manhattan system geometry constraints for traditional SLAM. Peasley et al. [160] uses a variant of position slam that incorporates the Manhattan system geometry system assumption by biasing only rotations of the robot within 90 degree steps. As a result, a square like map is obtained with little to no angular drift.

5.6 Discussion

In this chapter, we presented methods for the fusion of the Manhattan system geometry configurations to a coherent representation, over-segmention based on 2.5d data and the final segmentation using graph cuts. We showed that the use of traditional particle filters can improve the robustness and accuracy of the Manhattan system geometry estimation using all estimated configurations and the robot's odometry. While the sensor model of the particle filter for fusion is tuned for noise removal, the odometry is used as a motion model. The previously proposed Manhattan system geometry estimators tend to have Gaussian noise in their estimates due to limitations in image or depth resolutions. For instance, the 2D estimator still relies on edge detection, which strongly depends on the robot's motion (ego-motion), illumination and structure of the room itself in the instance where only low contrast edges are visible in some places. The 3D MSAC estimator based on normal vectors is limited in its accuracy on the quality of the normal vectors. Both used sensors use the principle of disparates for depth estimation; therefore, depth values tend to be quantize and not smooth resulting in quantized normal vectors. The minimum entropy Manhattan system geometry estimator on the other hand uses a particle filter underneath for estimation. In order to incorporate the noise from the estimates we use a Cauchy function, since it is not so heavy on its tails in contrast to the standard Gaussian function. This allows use of a more narrow Cauchy (so smaller Ω) function than with Gaussian ones. The robustness is increased by incorporating the associated variance of each Manhattan system geometry configuration estimate with the robot's motion. We also showed that using uncertain Manhattan system geometry configuration such as using MSAC Normal vector estimation with AIT Stereo Vision can still improve the overall performance. One

pitfall with this approach is that we assume that at least one estimator can detect a Manhattan system geometry system from the data. With our experiment, we figured out that in average ≈ 2.4 configurations are estimated in the entire dataset.

With our approach for segmentation, we use a graph-based representation that is obtained from multi-scale oversegmentation: The usage of superpixels for oversegmentation has become the de facto standard in computer vision in the last years according to a variety of sources [112, 135, 156–158, 161]. Our heuristic technique also incorporates the depth data and 2D geometry features rather than only the plain 2D image or 3D planes. Our experiments have shown that unconstrained superpixels at higher scales are mostly unused by the graph cut for segmentation since they tend to group wall segments with ground segments for images with weak contrast. However the graph cut seem to favor superpixels from a lower scale. The key feature for the multi-label graph

Table 5.3: Summarized properties of the introduced algorithms in this chapter with non-multithreading and non-optimized code on our test computer (see page 27). The typical runtime is given for Microsoft's Kinect and is 1.5× slower than AIT Stereo Visionwith oversegmention

	Manhattan system sensor fusion	Superpixel Oversegmentation	Segmentation using MRF based multi-labeling	Geometric Constrained Visual Odometry	Geometric Constrained Spatial-temporal Map
Accuracy	++	+	++	++	+
Robustness	++	+	++	+	++
Handling of Uncertainty	+	o	+	o	+
estimation method	particle filter	spanning tree	Graph	RANSAC	ICP
				cut	
parametric Model	no	yes	yes	yes	no
num model dimensions	3	3	3	1	2
Polynomial Complexity	O(mn)	O(n log n)	O(n log m)	O(m log n)	O(n log n)
typical n	3	$1280x1024$	800	50	$\frac{640x480}{2}$
typical m	500		12	60	
typical runtime (ms)	> 2	450	480	> 1	250

cut is the orientation of the superpixel segments rather than just the planes. This strategy turned out the be quite robust, since it also allows us to incorporate 2D cues (using orientation maps) about the orientation together with the extracted planes. The use of the multi-label graph cut to minimize the number of segments turned out to be quite efficient, since only segments with the same orientation can belong to each other. The major drawback of our approach is that we rely on proper oversegmentation and correctly estimated Manhattan system geometry.

We also presented applications for geometrically constrained visual odometry and geometrically constrained mapping using ICP. Although the focus was not on these two topics, we showed the potential of the Manhattan system geometry assumption. The use of the assumption showed both a major speed up of standard approaches like the geometrically constrained ICP. Since the rotational degrees of freedom are known, only the translative degrees of freedom need to be considered. We extended the idea of ICP by considering the membership of a voxel to a Manhattan system geometry in the matching process by maintaining four separate maps, those of X, Y, Z and "unknown" and adapt the facelet concept from point-to-facelet to point-to-geometry. We also showed that the map can be further improved using our adapted visual odometry. However the system relies on the proper detection of the Manhattan system geometry much like the previous approaches and can only be used for Microsoft's Kinect, since the depth AIT Stereo Vision is not dense enough and is rejected by the system. Another aspect is that we only incorporate the data from frame-to-frame which is commonly done with Microsoft's Kinect mapping [141, 142, 150, 152], but can be improved with PoseSLAM approach from Vanegas et al. [116] who incorporates the Manhattan system geometry in the loop-closing step.

Table 5.3 gives an overview of the proposed algorithms and their properties. One can see that some of used methods are either high in complexity and small on n, or vice versa. The major bottleneck here is the overestimation in two scales.

6 Conclusion

With this thesis we demonstrated a robust and fast perception system that detects indoor room structure using two different cost efficient 2.5d sensors: AIT Stereo Vision and Microsoft's Kinect. We exploited a certain property of man-made structure that holds true for many indoor environments called the **Manhattan system geometry** assumption. With our research, we figured out that this assumption is far more useful than expected. On one hand, this assumption is quite strict and limiting. On the other hand, this assumption reduces the search space for things such as plane fitting[1] It turned out that the assumption enables a high robustness for noise and occlusion. We can formulate it as a metric and easily use it to reject non-Manhattan system geometry and give the most plausible estimate from the given data. One major drawback of our approach is that we rely on a correct estimation of Manhattan system geometry for all processing steps. While our approach can reject non-Manhattan system geometry, it can still fail in some special cases, an example of such a case being if the Manhattan system geometry is at least 30% not visible. From the outset, we focused on the aspect of having multiple Manhattan system geometry configurations within the environment, by introducing the *global* and *local* Manhattan system geometry configurations. While the global Manhattan system geometry configuration is almost always found, the local ones are hard to detect due to the strict Manhattan model. In many cases, local Manhattan system geometry structures are very small and are hard to detect (see fig. 3.28(b) on page 64). Doors are usually aligned with the major axis within certain limits. Therefore we only use them in sensor fusion to prevent that a local one is considered as a global one. We do not use the local ones in the segmentation, since the cases are rare within our database. Another role is the relaxation of the Manhattan system geometry configuration once it has been estimated. This step is always a trade-off in robustness and accuracy. It is necessary, due to almost no structure in our environment being 100% straight.

[1] assuming the planes are oriented to the Manhattan system geometry

(a) Cafe im Kunsthaus (b) Hundertwasserhaus

Figure 6.1: Hunderwasser's art design in Vienna.

Our approach works with many standard homes e.g. see figure 2.1 on page 14, as long the majority of the structure is aligned to the three major axes and the surfaces are planar. However, this does not always holds true for some cases like museums or designer apparent. Some artists like *Friedensreich Hundertwasser* (1928-2000) prefer "everything than normal" structures in architecture. Some examples are his works in Osaka or Vienna, which can be viewed in figure 6.1. One key element is that almost nothing is flat (see fig. 6.1(a)) and everything seems to be a large-scale puzzle (6.1(b)). The parts of the puzzle seem to be arbitrary too and do not follow any continuity laws, pattern or scale. It is very unlikely that the true Manhattan system geometry configuration can be found with our approach in 2D, since no lines or structures are parallel or 3D since nothing is flat/planar. Another issue is that our segmentation approach would result in bogus results, because the puzzle-like structure will confuse the superpixels and the smoothness term of the graph cut.

6.1 Contributions

Direct robust estimation of Manhattan system geometry configuration
The Manhattan system geometry configuration estimation is the biggest part of this work and applies independent estimators which only use 2D or 3D data. These are *vanishing point detection* in 2D, *minimum entropy* method using 3D histograms and *normal vector* MSAC estimation. The overall intention from the beginning was to use multiple estimators in parallel in order to increase the robustness. As our experiments

have shown, some estimators work with some sensors better than with other ones. For instance, the 2D estimator & minimum entropy 3D estimator work best with AIT Stereo Vision, while the 3D normal vector estimation works best for Microsoft's Kinect. The MSAC method gives decent results on the horizontal error with AIT Stereo Vision, but still improves the result on the vertical axis, when all estimates are fused together into one system. The reverse example holds true for Microsoft's Kinect and the 2D method. All methods have been tested by various benchmarks, for example number of iterations for MSAC or occlusion masks. We showed that our system can cope with an amount of occlusion up to 80% and a Gaussian noise up to 32cm with the 3D data. The key for this performance in 2D is the explicit modeling of occlusion of structure and noise within the image. The strategy of using many simple (but not primitive-like pixels) features turned out to be quite robust if a proper grouping strategy is applied. Another key is also the method of the estimation of the vanishing point itself: we directly estimate a valid system rather than searching for three individual vanishing points.

Explicit modeling of uncertainty

The key for 3D data processing is the explicit modeling of uncertainty in depth perception of the sensors. We used the strategy of a depth confidence value per pixel within the native sensor space. The Depth confidence for AIT Stereo Vision was based on the similarity of the best match to the second best one, while Microsoft's Kinect used a heuristic technique based on the characteristics of the sensor. As a result, we were able to also use "uncertain"[2] depth values for AIT Stereo Vision, as well as being able to reject interpolated depth values from Microsoft's Kinect. Our results show that even standard methods can achieve better results using this simple technique.

Fast feature extraction using geometrically constrained extractors

Using feature extraction, we also exploit the Manhattan system geometry assumption which makes it faster and more robust. For instance, we extract lines in 2D vision using directed garbor filters. Since we only extracted lines that are oriented to the previously extracted vanishing point, we only use one (instead of multiple, for example 64) filter since the orientation is previously known. For 3D vision, we use robust histogram methods that would be too computationally taxing with six degree of freedom instead of three; only the translative degree of freedom remains. This similarly holds

[2] resulting in more dense data

true for the 1D CC-RANSAC plane estimation; we can use a 1D estimator with a connected component analysis with six degrees of freedom, but this is also extremely computationally taxing (resulting in more iterations).

Robust 2.5d Segmentation and Applications

The graph-based segmentation is sufficient, since it unites all the data from each source in one output sensors space. The usage of the 2D sensor space for segmentation turned out to be optimal since all 2D and 3D sources can be back-projected. The usage of 2D features in a two-step method significantly improves the quality of the segmentation for AIT Stereo Vision. With Microsoft's Kinect, we also achieve a higher segmentation quality since the 2D image has a higher resolution (4.2×) than the 3D depth data.
The usage of the Manhattan system geometry assumption is not only limited to image segmentation within robotics. We showed that the concept can also be applied for 3D mapping using geometrically constrained ICP methods. Since an estimate per voxel can be assigned to a Manhattan system geometry, we can build constrained maps such as having one map for each axis. Another benefit is the usage facelet interpolation instead of a point-to-point method with standard ICP. Here too we gain a speed increase due to the Manhattan system geometry assumption.

6.2 Future Steps

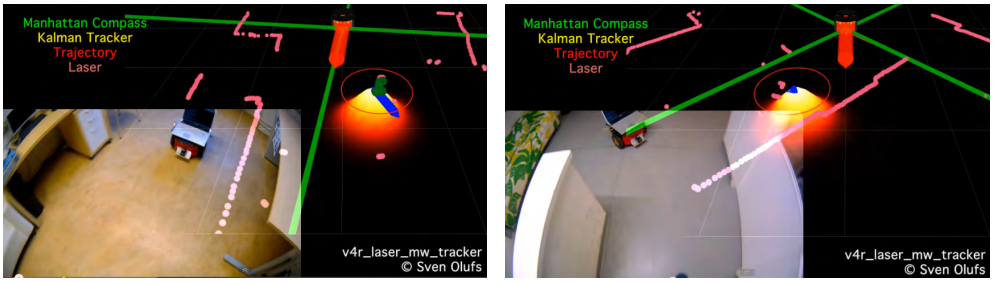
Sophisticated relaxation of Manhattan system geometry configuration

The use of one relaxation, that is to say, tolerance within the Manhattan system geometry within one axis, is adequate, but can be improved by allowing a relaxation per image or depth region. It is unlikely that walls that aren't perfectly straight[3] are also parallel. We made the trade-off allowing a bigger relaxation after Manhattan system geometry configuration estimation by way of using 5 degree tolerance

Unified 2D and 3D Manhattan system geometry estimation

With our approach, we use Manhattan system geometry configuration estimators independently from each other and fuse them in a post-processing step. However the estimation from 2D and 3D simultaneously can further improve the robustness according to Barinova et al. [78] using graph cuts. One key issue is how to balance 2D and 3D features for estimation some lines in contrast to many normal vectors. Since the

[3] within the $1deg$ range as "non-perfect"



(a) Living Room (b) Corridor

Figure 6.2: Example of using Manhattan system geometry for behavior (http://youtu.be/E_QTV7QCBWs). The robot follows the human (bottom of the picture) only within the four major orientations + four 45 degree rotated ones. The green line shows the estimated system and the red arrow shows the control trajectory.

2D vision estimator and normal vector-based estimation use both a MSAC estimator within the same state space (Gaussian Sphere), they can be used to estimate the same Manhattan system geometry using both sources. For instance, each source can be used to estimate one (non-identical) axis of the three[4] normal axes XYZ and to construct a valid Manhattan system geometry.

Using Manhattan system geometry in the loop

Within this thesis, sensor processing was only standalone and was not used for any actual behavior, e.g. [17] for car navigation. However, parts of this work like the minimum entropy estimator and 1D CC-RANSAC have been adapted for simple navigation[5] using a Hokuyo laser scanner and a "Pionner DX3" robot. Figure 6.2 gives an example: First, the Manhattan system geometry configuration is estimated. It is shown with the green cross. Next, the walls are detected and removed from the dataset using the same 1D CC-RANSAC methods used for 3D. The robot follows everything including the human that is "not wall" using a Kalman filter (shown as ellipsoid) within 8 major directions (4 main + 4 sub directions) instead of a straight line if the user is more than 1m away from the robot. For instance, if the robot were to follow a straight line in fig. 6.2(b), it would crash into the wall on the left. Using these simple techniques allows the robot to follow the human without bumping into the environment while the needed computational power is significantly lower than for the 2.5d sensor. Other high-level applications like object search have to be investigated further.

[4] only two axes are needed, because the third one is redundant
[5] See http://youtu.be/E_QTV7QCBWs

Bibliography

[1] Jon Agirre Ibarbia, Jennifer Stack, Geoff Pegman, and Thilo Zimmermann. Deliverable d3.1.2: Guide for start-ups. Technical report, European Robotics Coordination Action, Diamant Building, Boulevard A. Reyers 80, 1030 Brussels, 2011.

[2] Céline Ray, Francesco Mondada, and Roland Siegwart. What do people expect from robots? In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3816–3821, 2008.

[3] *Towards a Comprehensive Chore List for Robots*, Tokyo, Japan, 2013. ACM Press.

[4] A. E. Brain, A. Macovski, M. E. Hoff, A. B Novikoff, G. H Ball, and C. P. Bourne. Graphical data processing research study and experimental investigation. Technical report, AI Center, SRI Internationa, 333 Ravenswood Ave, Menlo Park, CA 94025, Nov 1960.

[5] Rodney Brooks. Intelligence without representation. *Artificial Intelligence*, 47: 139–159, 1991.

[6] D. Murray and J. J. Little. Using real-time stereo vision for mobile robot navigation. *Autonomous Robots*, 2(8):161–171, 2000.

[7] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, Cambridge MA, first edition, 2005.

[8] R. B. Rusu, A. Sundaresan, B. Morisset, K. Hauser, M. Agrawal, J. Latombe, and M. Beetz. Leaving flatland: Efficient real-time three-dimensional perception and motion planning. *Journal of Field Robotics, Special Issue: Three-Dimensional Mapping*, 26(10):841–862, 2009.

[9] Danica Kragic and Markus Vincze. Vision for robotics. *Foundations and Trends in Robotics archive*, 1(1):1–78, 2009.

[10] D. Burschka and G.D. Hager. Vision-based 3d scene analysis for driver assistance. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2005.

[11] P. Mayer, G. Edelmayer, G.J. Gelderblom, M. Vincze, P. Einramhof, M. Nuttin, T. Fuxreiter, and G. Kronreif. Movement -modular versatile mobility enhancement system. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2007.

[12] Markus Vincze. HOBBIT - towards a robot for aging well. In *Proc. of the ICRA 2013 Workshop on Human Robot Interaction (HRI) for Assistance Robots and Industrial Robots*, Karlsruhe, May 2013.

[13] Markus Vincze, Walter Wohlkinger, Sven Olufs, Peter Einramhof, and Robert Schwarz. Towards Bringing Robots into Homes. In *Joint Conference of the 41st International Symposium on Robotics (ISR 2010) and the 6th German Conference on Robotics (ROBOTIK 2010)*, Munich, Germany, 2010.

[14] Yingze Bao, Min Sun, and Silvio Savarese. Toward coherent object detection and scene layout understanding. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, San Francisco, California, 2010.

[15] Yasutaka Furukawa, Brian Curless, Steven M Seitz, and Richard Szeliski. Reconstructing building interiors from images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 80–87. IEEE, 2009.

[16] Jianxiong Xiao and Yasutaka Furukawa. Reconstructing the world's museums. In *European Conference on Computer Vision (ECCV)*, pages 668–681. Springer, 2012.

[17] Dmitri Dolgov and Sebastian Thrun. Detection of principle directions in unknown environments for autonomous navigation. In *Robotics: Science and Systems (RSS)*, Zurich, Switzerland, June 2008.

[18] Alex Flint, Christopher Mei, Ian Reid, and David Murray. Growing semantically meaningful models for visual slam. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010.

[19] Sven Olufs and Markus Vincze. Room-Structure estimation in Manhattan-like Environments from dense 2.5D range data using minimum Entropy and Histograms. In *IEEE Workshop on Applications of Computer Vision (WACV)*, Hawaii, U.S.A., 2011.

[20] Y. Furukawa, B. Curless, S. Seitz, and R. Szeliski. Manhattan-world stereo. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[21] J. Coughlan and A. L. Yuille. Manhattan world: orientation and outlier detection by bayesian inference. *Neural Comput.*, 15(5):1063–1088, 2003.

[22] Sven Olufs and Markus Vincze. Towards efficient Semantic Real time mapping of Man-made environments using Microsoft's Kinect. In *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Puket, Thailand, 2011.

[23] P. H. S. Torr and A. Zisserman. Mlesac: A new robust estimator with application to estimating image geometry. *Computer Vision and Image Understanding*, 78: 2000, 2000.

[24] Sven Olufs and Markus Vincze. Visual IMU in Manhattan-like Enviroments from 2.5D data. In *Seventeenth International Symposium on Artificial Life and Robotics (AROB 17th 2012)*, Beppu, Japan, 2012.

[25] Sven Olufs and Markus Vincze. Efficient Semantic mapping of Man-made environments using Kinect. In *IEEE Proc. of the ICRA 2011 Workshop on Active Semantic Perception and Object Search in the Real World (ASP-AVS-11)*, San Francisco, U.S.A., 2011.

[26] Sven Olufs and Markus Vincze. Real time Manhattan-like Structure segmentation from Kinect with constrained 1D CC-RANSAC. In *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR2011)*, Kyoto, Japan, 2011.

[27] Robert Schwarz, Sven Olufs, and Markus Vincze. Merging line segments in 3D using mean shift algorithm in man-made environments. In *34st AAPR/OAGM Workshop*, Zwettel, Austria, 2010.

[28] Sven Olufs and Markus Vincze. Robust Single View Room Structure Segmentation in Manhattan-like Environments from Stereo Vision. In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, 2011.

[29] Sven Olufs and Markus Vincze. Towards robust room structure segmentation in manhattan-like environments from dense 2.5d data (best paper award). In *IEEE 11th International Conference on Control, Automation and Systems (ICCAS)*, Seoul, South Korea, 2011.

[30] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8:679–714, 1986.

[31] J. W. Weingarten, G. Gruener, and R. Siegwart. A state-of-the-art 3d sensor for robot navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004.

[32] Jana Kosecka and Wei Zhang. Video compass. In *European Conference on Computer Vision (ECCV)*, 2002.

[33] Nicolas Guilbert, Fredrik Kahl, Karl Astrom, Magnus Oskarsson, Martin Johansson, and Anders Heyden. Constraint enforcement in structure and motion applied to closing an open sequence. In *Asian Conference on Computer Vision*, 2004.

[34] Peter Einramhof, Sven Olufs, and Markus Vincze. Experimental evaluation of state of the art 3d-sensors for mobile robot navigation. In *31st AAPR/OAGM Workshop*, Schloss Krumbach, Austria, 2007.

[35] Markus Vincze, Sven Olufs, Peter Einramhof, and Horst Wildenauer. Roboternavigation in Büros und Wohnungen (In German). *Elektrotechnik und Informationstechnik (e&i)*, 1-2(1-2):25 – 32, 2008.

[36] S. Olufs and M. Vincze. An Efficient Area-based Observation Model for Monte-Carlo Robot Localization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2009.

[37] S. Olufs and M. Vincze. An Intuitive Inexpensive Interface for Robots using the Nintendo Wii Remote. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2009.

[38] Kristian Ambrosch, Martin Humenberger, and Sven Olufs. Embedded Stereo Vision. In Ahmed Nabil Belbachir, editor, *Smart Cameras*, chapter 8. Springer, London, first edition, 2010. ISBN 978-1-4419-0953-4.

[39] Sven Olufs and Markus Vincze. Robust Room-Structure estimation in Manhattan-like Environments from dense 2.5D range data. In *IEEE Proc. of the ICRA 2010 Workshop on Semantic Mapping and Autonomous Knowledge Acquisition*, Taipei, Taiwan, 2010.

[40] Sven Olufs, Paul G. Plöger, and Markus Vincze. Probabilistic Shape Vision for Embedded Systems (Best Paper Award). In *IEEE 8th International Conference*

on Ubiquitous Robots and Ambient Intelligence (URAI), Incheon, South Korea, 2011.

[41] Markus Vincze, Walter Wohlkinger, Aitor Aldoma Buchaca, Sven Olufs, Peter Einramhof, Kai Zhou, Ekaterina Potapova, David Fischinger, and Michael Zillich. Roboternavigation in Büros und Wohnungen (In German). *Elektrotechnik und Informationstechnik (e&i)*, 1(129):42 – 52, 2012.

[42] Sven Olufs and Markus Vincze. Semantic Segmentation in Manhattan-like Environments from 2.5D data. In *IEEE Eighteenth International Symposium on Artificial Life and Robotics (AROB 2013)*, Daejeon, South Korea, 2013.

[43] Markus Vincze, Sven Olufs, Walter Wohlkinger, Peter Einramhof, Robert Schwarz, and Karthik Mahesh Varadarajan. A Situated Approach to Scene Understanding and Object Categorization for Domestic Robots. *Journal of Intelligent & Robotic Systems*, in press.

[44] Ole Bruun. *An Introduction to Feng Shui*. Cambridge University Press, 2008. ISBN 9-78052168-217-6.

[45] Stephen E. Palmer. *Vision Science*. MIT Press, 1999. ISBN 978-0262161831.

[46] J. Wagemans, J. Feldman, S. Gepshtein, R. Kimchi, P.A. van der Helm J.R. Pomerantz, and C. van Leeuwen. A century of gestalt psychology in visual perception: Ii. conceptual and theoretical foundations. *Psychol Bulletin*, 6 (138):1218–1252, 2012.

[47] H. Wildenauer and A. Hanbury. Robust camera self-calibration from monocular images of manhattan worlds. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2831–2838, June 2012. doi: 10.1109/CVPR.2012.6248008.

[48] Sudipta N. Sinha, Drew Steedly, Richard Szeliski, Maneesh Agrawala, and Marc Pollefeys. Interactive 3d architectural modeling from unordered photo collections. *ACM Trans. Graph.*, 27(5):159:1–159:10, December 2008. ISSN 0730-0301.

[49] Tat-Jen Cham, A. Ciptadi, Wei-Chian Tan, Minh-Tri Pham, and Liang-Tien Chia. Estimating camera pose from a single urban ground-view omnidirectional image and a 2d building outline map. In *IEEE Conference on Computer Vision*

and Pattern Recognition (CVPR), pages 366–373, June 2010. doi: 10.1109/CVPR.2010.5540191.

[50] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[51] Alex Teichman, Stephen Miller, and Sebastian Thrun. Unsupervised intrinsic calibration of depth sensors via slam. In *Proceedings of Robotics: Science and Systems*, Berlin, Germany, June 2013.

[52] Martin Humenberger, Christian Zinner, and Wilfried Kubinger. Performance evaluation of a census-based stereo matching algorithm on embedded and multi-core hardware. In *Proccedings of the International Symposium on Image and Signal Processing and Analysis*, volume 6, 2009.

[53] Arnaud Doucet. *Sequential Monte Carlo methods in practice*. Statistics for engineering and information science. New York, NY [u.a.]: Springer, 2001. ISBN 0-387-95146-6.

[54] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[55] C H Chen, L F Pau, and P S P Wang. *Handbook of Pattern Recognition & Computer Vision*. World Scientific - Singapore New Jersey London Hong Kong, 2nd edition, 2001.

[56] David Douglas and Thomas Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 10(2):112–122, 1973. ISSN 00083127.

[57] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[58] Y. Cheng. Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8), 1995.

[59] J.-P. Tardif. Non-iterative approach for fast and accurate vanishing point detection. In *International Conference on Computer Vision (ICCV)*, pages 1250–1257, 2009. doi: 10.1109/ICCV.2009.5459328.

[60] J.-C. Bazin and M. Pollefeys. 3-line ransac for orthogonal vanishing point detection. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4282–4287, Oct 2012. doi: 10.1109/IROS.2012.6385802.

[61] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, third edition, 2009. ISBN 978-0136042594.

[62] J. G. Daugman. Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters. *Journal of the Optical Society of America A*, 2(7):1160âĂŞ–1169, 1985.

[63] Rajiv Mehrotra, Kameswara Rao Namuduri, and Nagarajan Ranganathan. Gabor filter-based edge detection. *Pattern Recognition*, 25(12):1479–1494, 1992.

[64] Rabia Jafri and Hamid R Arabnia. A survey of face recognition techniques. *JIPS*, 5(2):41–68, 2009.

[65] Antonio Torralba, Kevin P Murphy, and William T Freeman. Using the forest to see the trees: exploiting context for visual object detection and localization. *Communications of the ACM*, 53(3):107–114, 2010.

[66] Marisa Carrasco. Visual attention: The past 25 years. *Vision research*, 51(13): 1484–1525, 2011.

[67] P. Denis, J. H. Elder, and F. Estrada. Efficient edge-based methods for estimating manhattan frames in urban imagery. In *European Conference on Computer Vision*, pages 197–210, 2 2008.

[68] A. P. Dempster, N. M. Laird, and Rubin D. B. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977. ISSN 00359246.

[69] G. Schindler and F. Dellaert. Atlanta world: an expectation maximization framework for simultaneous low-level edge grouping and camera calibration in complex man-made environments. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages I–203–I–209 Vol.1, June 2004. doi: 10.1109/CVPR.2004.1315033.

[70] Marco Nieto and Luis Salgado. Simultaneous estimation of vanishing points and their converging lines using the {EM} algorithm. *Pattern Recognition Letters*, 32(14):1691 – 1700, 2011. ISSN 0167-8655.

[71] F. Schaffalitzky and A. Zisserman. Planar grouping for automatic detection of vanishing lines and points. *Image and Vision Computing*, 18(9):647 – 658, 2000. ISSN 0262-8856.

[72] F.M. Mirzaei and S.I. Roumeliotis. Optimal estimation of vanishing points in a manhattan world. In *International Conference on Computer Vision (ICCV)*, pages 2454–2461, Nov 2011. doi: 10.1109/ICCV.2011.6126530.

[73] Roberto Toldo and Andrea Fusiello. Robust multiple structures estimation with j-linkage. In *European Conference on Computer Vision (ECCV)*, 2008.

[74] Baojiang Zhong, Dongsheng Xu, and Jiwen Yang. Vertical corner line detection on buildings in quasi-manhattan world. In *Image Processing (ICIP), 2013 20th IEEE International Conference on*, pages 3064–3068, Sept 2013. doi: 10.1109/ICIP.2013.6738631.

[75] S. Sinha, D. Steedly, and R. Szeliski. Piecewise planar stereo for image-based rendering. In *International Conference on Computer Vision (ICCV)*, 2009.

[76] Dorit Borrmann, Jan Elseberg, Kai Lingemann, and Andreas Nüchter. The 3d hough transform for plane detection in point clouds - a review and a new accumulator design. *Journal 3D Research*, 2(2), March 2011. ISSN 2092-6731.

[77] Bo Li, Kun Peng, Xianghua Ying, and Hongbin Zha. Vanishing point detection using cascaded 1d hough transform from single images. *Pattern Recognition Letters*, 33(1):1 – 8, 2012. ISSN 0167-8655.

[78] Olga Barinova, Victor Lempitsky, Elena Tretiak, and Pushmeet Kohli. Geometric image parsing in man-made environments. In *Computer Vision–ECCV 2010*, pages 57–70. Springer Berlin/Heidelberg, 2010.

[79] M Antunes and J.P. Barreto. A global approach for the detection of vanishing points and mutually orthogonal vanishing directions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1336–1343, June 2013. doi: 10.1109/CVPR.2013.176.

[80] M. Hornacek and S. Maierhofer. Extracting vanishing points across multiple views. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 953–960, June 2011. doi: 10.1109/CVPR.2011.5995396.

[81] A. Ames. The ames demonstrationsin perception. *New York: Hafner Publishing*, 1952.

[82] C. Jennings and D. Murray. Stereo vision based mapping and navigation for mobile robots. In *IEEE International Conference on Robotics and Automation*, pages 1694–1699, 1997.

[83] Q. Yu, H. Araujo, and H. Wang. Stereo-vision based real time obstacle detection for urban environments. In *Int. Conf. on Advanced Robotics*, 2003.

[84] Robert Sim and James J. Little. Autonomous vision-based exploration and mapping using hybrid maps and rao-blackwellised particle filters. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2006.

[85] S. Thrun, D. Fox, and W. Burgard. A real-time algorithm for mobile robot mapping with application to multi robot and 3d mapping. In *International Conference on Robotics & Automation*, San Francisco, CA, USA, 2000.

[86] P. Elinas and J.J. Little. omcl: Monte-carlo localization for mobile robots with stereo vision. In *Proceedings of Robotics: Science and Systems*, pages 373–380, 2005.

[87] D. Gallup, J. Frahm, P. Mordohai, Q. Yang, and M. Pollefeys. Real-time plane-sweeping stereo with multiple sweeping directions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.

[88] B. Micusik and J. Kosecka. Piecewise planar city modeling from street view panoramic sequences. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[89] A. Flint, C. Mei, D. W., Murray, and I. D. Reid. A dynamic programming approach to reconstructing building interiors. In *European Conference on Computer Vision (ECCV)*, Crete, Greece, 2010.

[90] A. Gupta, A. A. Efros, and M. Hebert. Blocks world revisited: Image understanding using qualitative geometry and mechanics. In *European Conference on Computer Vision (ECCV)*, 2010.

[91] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9–13 2011.

[92] Hernan Badino, Daniel Huber, Y. Park, and Takeo Kanade. Fast and accurate computation of surface normals from range images. In *IEEE International Conference on Robotics and Automation (ICRA)*, May 2011.

[93] Dirk Holz, Stefan Holzer, Radu Bogdan Rusu, and Sven Behnke. Real-time plane segmentation using rgb-d cameras. In *RoboCup Symposium*, 2011 2011.

[94] Paul Viola and Michael J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 2(57):137–154, 2004.

[95] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Introduction to Autonomous Mobile Robots*. MIT Press, second edition, 2011. ISBN 9780262015356.

[96] D. Scaramuzza, F. Fraundorfer, and R. Siegwart. Real-time monocular visual odometry for on-road vehicles with 1-point ransac. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4293–4299, 2009.

[97] B. Clipp, Lim Jongwoo, J.-M. Frahm, and M. Pollefeys. Parallel, real-time visual slam. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[98] Andrew J. Davison, Ian Reid, Nicholas Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2007.

[99] Georg Klein and David Murray. Parallel tracking and mapping for small AR workspaces. In *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, 2007.

[100] S. Thrun. A probabilistic online mapping algorithm for teams of mobile robots. *International Journal of Robotics Research*, 20(5):335–363, 2001.

[101] D. Schleicher, L.M. Bergasa, R. Barea, E. Lopez, M. Ocaa, J. Nuevo, and P. Fernandez. Real-time stereo visual slam in large-scale environments based on sift fingerprints. In *IEEE International Symposium on Intelligent Signal Processing*, 2007.

[102] Kurt Konolige, Giorgio Grisetti, Rainer Kummerle, Wolfram Burgard, Benson Limketkai, and Régis Vincent. Sparse pose adjustment for 2d mapping. In

IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2010.

[103] Martin Humenberger, Christian Zinner, Michael Weber, Wilfried Kubinger, and Markus Vincze. A fast stereo matching algorithm suitable for embedded real-time systems. *Computer Vision and Image Understanding*, 114(11):1180 – 1202, 2010. ISSN 1077-3142.

[104] Rober Sedgewick. *Algorithms in C*. first edition, 1992. ISBN 3-89319-376-6.

[105] M. Zuliani, , C.S. Kenney, and B.S. Manjunath. The multiransac algorithm and its application to detect planar homographies. In *Image Processing, 2005. ICIP 2005. IEEE International Conference on*, volume 3, pages III–153–6, 2005.

[106] D. Comaniciu, V. Ramesh, and P. Meer. Kernel-based object tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25:564–575, May 2003.

[107] J.M. Saez and F. Escolano. Entropy minimization slam using stereo vision. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2005.

[108] Orazio Gallo, Roberto Manduchi, and Abbas Rafii. CC-RANSAC: fitting planes in the presence of multiple surfaces in range data. *Pattern Recognition Letters*, 2011. ISSN 0167-8655.

[109] Yangyan Li, Qian Zheng, Andrei Sharf, Daniel Cohen-Or, Baoquan Chen, and Niloy J Mitra. 2d-3d fusion for layer decomposition of urban facades. In *International Conference on Computer Vision (ICCV)*, pages 882–889. IEEE, 2011.

[110] Varsha Hedau, Derek Hoiem, and David Forsyth. Recovering the spatial layout of cluttered rooms. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1849–1856. IEEE, 2009.

[111] D. Hoiem, A.A. Efros, and M. Hebert. Recovering surface layout from an image. *IJCV*, 75(1), 2007.

[112] David Gallup, J-M Frahm, Philippos Mordohai, Qingxiong Yang, and Marc Pollefeys. Real-time plane-sweeping stereo with multiple sweeping directions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8. IEEE, 2007.

[113] Marc Pollefeys, David Nistér, J-M Frahm, Amir Akbarzadeh, Philippos Mordohai, Brian Clipp, Chris Engels, David Gallup, S-J Kim, Paul Merrell, C. Salmi, S. Sinha, B. Talton, L. Wang, Q. Yang, H. Stewenius, R. Yang, G. Welch, and H. Towles. Detailed real-time urban 3d reconstruction from video. *International Journal of Computer Vision*, 78(2-3):143–167, 2008.

[114] Lukas Zebedin, Joachim Bauer, Konrad Karner, and Horst Bischof. Fusion of feature- and area-based information for urban buildings modeling from aerial imagery. In David Forsyth, Philip Torr, and Andrew Zisserman, editors, *Computer Vision âĂŞ ECCV 2008*, volume 5305 of *Lecture Notes in Computer Science*, pages 873–886. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-88692-1. doi: 10.1007/978-3-540-88693-8_64.

[115] A-L Chauve, Patrick Labatut, and J-P Pons. Robust piecewise-planar 3d reconstruction and completion from large-scale unstructured point data. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1261–1268. IEEE, 2010.

[116] Carlos A Vanegas, Daniel G. Aliaga, and Bedrich Benes. Automatic extraction of manhattan-world building masses from 3d laser range scans. *Visualization and Computer Graphics, IEEE Transactions on*, 18(10):1627–1637, Oct 2012. ISSN 1077-2626. doi: 10.1109/TVCG.2012.30.

[117] Carlos A Vanegas, Daniel G. Aliaga, and Bedrich Benes. Building reconstruction using manhattan-world grammars. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 358–365. IEEE, 2010.

[118] Liangliang Nan, Andrei Sharf, Hao Zhang, Daniel Cohen-Or, and Baoquan Chen. Smartboxes for interactive urban reconstruction. *ACM Transactions on Graphics (TOG)*, 29(4):93, 2010.

[119] R. Triebel, W. Burgard, and F. Dellaert. Using hierarchical em to extract planes from 3d range scans. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2005.

[120] Roberto Toldo and Andrea Fusiello. Image-consistent patches from unstructured points with j-linkage. *Image and Vision Computing*, 31(10):756–770, 2013.

[121] Bastian Oehler, Joerg Stueckler, Jochen Welle, Dirk Schulz, and Sven Behnke.

Efficient multi-resolution plane segmentation of 3d point clouds. In *Intelligent Robotics and Applications*, pages 145–156. Springer, 2011.

[122] Jens-Steffen Gutmann, Masaki Fukuchi, and Masahiro Fujita. 3d perception and environment map generation for humanoid robot navigation. *The International Journal of Robotics Research*, 27(10):1117–1134, 2008.

[123] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother. A comparative study of energy minimization methods for markov random fields with smoothness-based priors. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(6):1068 –1080, jun. 2008. ISSN 0162-8828. doi: 10.1109/TPAMI.2007.70844.

[124] Andrew Blake and Michael Isard. *Active Contours*. Springer, 1998.

[125] Sebastian Thrun, Dieter Fox, and Wolfram Burgard. Monte carlo localization with mixture proposal distribution. In *AAAI/IAAI*, pages 859–865, 2000.

[126] A. Vedaldi and S. Soatto. Quick shift and kernel methods for mode seeking. In *Proceedings of the 12th European Conference on Computer Vision (ECCV)*, 2008.

[127] P. Felzenszwalb and D. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2), 2004.

[128] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34 (11):2274–2282, 2012.

[129] G. Mori. Guiding model search using segmentation. In *International Conference on Computer Vision (ICCV)*, 2005.

[130] A. Levinshtein, A. Stere, K. Kutulakos, D. Fleet, S. Dickinson, and K. Siddiqi. Turbopixels: Fast superpixels using geometric flows. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12), 2009.

[131] Michael Van den Bergh, Xavier Boix, Gemma Roig, Benjamin de Capitani, and Luc Van Gool. Seeds: Superpixels extracted via energy-driven sampling. In *Proceedings of the 12th European Conference on Computer Vision (ECCV)*, 2012.

[132] A. Delong, A. Osokin, H. Isack, and Y. Boykov. Fast approximate energy minimization with label costs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, San Francisco, California, 2010.

[133] David Changsoo Lee, Martial Hebert, and Takeo Kanade. Geometric reasoning for single image structure recovery. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[134] Srikumar Ramalingam and Matthew Brand. Lifting 3d manhattan lines from a single image. In *International Conference on Computer Vision (ICCV)*, 2013.

[135] Ashutosh Saxena, Min Sun, and Andrew Y. Ng. Make3d: Learning 3-d scene structure from a single still image. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2008.

[136] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics - Principles and Practice*. Second edition, 1997. ISBN 0-201-84840-6.

[137] Guy Barrett Coleman and Harry C Andrews. Image segmentation by clustering. *Proceedings of the IEEE*, 67(5):773–785, 1979.

[138] Dong-Chen He and Li Wang. Texture unit, texture spectrum, and texture analysis. *Geoscience and Remote Sensing, IEEE Transactions on*, 28(4):509–512, Jul 1990. ISSN 0196-2892. doi: 10.1109/TGRS.1990.572934.

[139] J. Engel, J. Sturm, and D. Cremers. Semi-dense visual odometry for a monocular camera. In *International Conference on Computer Vision (ICCV)*, Sydney, Australia, December 2013.

[140] C. Zach, D. Gallup, and J.-M. Frahm. Fast gain-adaptive klt tracking on the gpu. In *CVPR Workshop on Visual Computer Vision on GPU's (CVGPU)*, 2008.

[141] Erik Bylow, Jürgen Sturm, Christian Kerl, Fredrik Kahl, and Daniel Cremers. Real-time camera tracking and 3d reconstruction using signed distance functions. In *Robotics: Science and Systems (RSS)*, volume 9, 2013.

[142] Richard A Newcombe, Andrew J Davison, Shahram Izadi, Pushmeet Kohli, Otmar Hilliges, Jamie Shotton, David Molyneaux, Steve Hodges, David Kim, and

Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pages 127–136. IEEE, 2011.

[143] F. Steinbruecker, C. Kerl, J. Sturm, and D. Cremers. Large-scale multi-resolution surface reconstruction from rgb-d sequences. In *International Conference on Computer Vision (ICCV)*, Sydney, Australia, 2013.

[144] Andreas Nüchter, Hartmut Surmann, Kai Lingemann, Joachim Hertzberg, and Sebastian Thrun. 6d slam with application in autonomous mine mapping. In *IEEE International Conference on Robotics and Automation (ICRA)*, New Orleans, USA, 2004.

[145] K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems. In *Proc. of the ICRA 2010 Workshop on Best Practice in 3D Perception and Modelling for Mobile Manipulation*, Anchorage, AK, USA, May 2010.

[146] Yang Chen and Gérard Medioni. Object modelling by registration of multiple range images. *Image Vision Computing*, 10:145–155, April 1992. ISSN 0262-8856. doi: 10.1016/0262-8856(92)90066-C.

[147] L. Armesto, , J. Minguez, and L. Montesano. A generalization of the metric-based iterative closest point technique for 3d scan matching. In *IEEE International Conference on Robotics and Automation (ICRA)*, May 2010.

[148] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Computational Graphics*, 21: 163–169, August 1987. ISSN 0097-8930.

[149] Dirk Holz and Sven Behnke. Sancta Simplicitas – On the efficiency and achievable results of SLAM using ICP-Based Incremental Registration. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1380–1387, Anchorage, Alaska, USA, May 2010.

[150] Nicola Fioraio and Kurt Konolige. Realtime visual and point cloud slam. In *Proc. of the RSS 2011 Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, 2011.

[151] F. Steinbrucker, Jürgen Sturm, and Daniel Cremers. Real-time visual odometry from dense rgb-d images. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 719–722. IEEE, 2011.

[152] Thomas Whelan, Hordur Johannsson, Michael Kaess, John J Leonard, and John McDonald. Robust real-time visual odometry for dense rgb-d mapping. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5724–5731. IEEE, 2013.

[153] Ruiqi Guo and Derek Hoiem. Support surface prediction in indoor scenes. In *International Conference on Computer Vision (ICCV)*. IEEE, 2013.

[154] Alex Flint, David Murray, and Ian Reid. Manhattan scene understanding using monocular, stereo, and 3d features. In *International Conference on Computer Vision (ICCV)*, pages 2228–2235. IEEE, 2011.

[155] Camillo J Taylor and Anthony Cowley. Parsing indoor scenes using rgb-d imagery. *Robotics*, page 401, 2013.

[156] Ashutosh Saxena, Jamie Schulte, and Andrew Y Ng. Depth estimation using monocular and stereo cues. *IJCAI*, 7, 2007.

[157] B. Micusik, H. Wildenauer, and M. Vincze. Towards detection of orthogonal planes in monocular images of indoor environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2008.

[158] Xiaoqing Liu, Olga Veksler, and Jagath Samarabandu. Graph cut with ordering constraints on labels and its applications. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8. IEEE, 2008.

[159] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *European Conference on Computer Vision (ECCV)*, pages 746–760. Springer, 2012.

[160] Brian Peasley, Stan Birchfield, Alexander Cunningham, and Frank Dellaert. Accurate on-line 3d occupancy grids using manhattan world constraints. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5283–5290. IEEE, 2012.

[161] Varsha Hedau, Derek Hoiem, and David Forsyth. Recovering free space of indoor scenes from a single image. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2807–2814. IEEE, 2012.