Designing an Autonomous Exploration Architecture for an Indoor Quadcopter

Jens Nyman

Promoter: prof. dr. ir. Benjamin Schrauwen
Supervisors: Ken Caluwaerts, Tim Waegeman

Thesis submitted in partial fulfillment of a Master Degree
in Electrical Engineering

Electronics and Information Systems department
Department head: prof. dr. ir. Jan Van Campenhout
Faculty of Engineering and Architecture
Ghent University
Academic year: 2011-2012

UNIVERSITEIT
GENT

Preface

Already having a huge interest in robotics, I was immediately enthusiastic when I first heard about research results with small quadcopters. I remember watching the amazing quadcopter experiments of the University of Pennsylvania on YouTube. Consequently, I was happy to see that Reservoir Lab offered a thesis about this subject and I am very thankful that I was given this opportunity.

During my work on this project, I felt hugely supported by my supervisors Ken Caluwaerts and Tim Waegeman, on whom I could always rely for support, helpful insights and sympathy. They deserve a big thanks for all their time and effort and for their help with the NIPS paper. I would also like to thank my promotor Benjamin Schrauwen for the advice and encouragements I received from time to time and for letting me go to the SPPRA conference.

As I was frequently working at reservoir lab, I always enjoyed the company of the staff and the students working there. Therefore, I thank the people from reservoir lab for the cheerful conversations and my fellow students for the great atmosphere.

Further, I'm grateful to my friends and family for their support. My mother especially deserves a word of thanks for frequently helping me, not the least by helping me correcting this work. Lastly, I would like to thank Ilse, who was always there for me.

Jens Nyman
Ghent, June 17th 2012

Summary

This work addresses the problem of autonomous exploration with a low-cost quadcopter. Low-cost devices have a number of advantages like the low deployment threshold, e.g. for usage in dangerous environments. However, the absence of more expensive sensors such as a laser range finder makes the exploration task a lot more challenging.

We propose a novel architecture for indoor exploration based on four important concepts: video based localization and mapping, application of a room based methodology, wall detection based on a room model and 3D visualization by mapping textures on walls.

For *simultaneous localization and mapping* (SLAM), we conducted a thorough study of the available algorithms and opted for *Parallel Tracking And Mapping* (PTAM). We extended this implementation with some improvements, including support for multiple cameras.

The *room based methodology* is apparent throughout this work and states that all rooms in a building should be treated separately. This allows for much more scalable mapping and easier visualization.

The PTAM algorithm generates a map of landmarks, which can be represented as a point cloud. We want to fit walls onto these points for visualization. Because of a lack of adapted line fitting techniques in literature, we constructed a layered room model enabling us to optimize the wall locations via the Expectation Maximization algorithm. Results show that this approach outperforms other state-of-the-art techniques.

The generated set of walls is visualized in 3D. To give the user an impression of what the room looks like, textures are mapped onto the walls and the floor, using PTAM information and camera frames.

Keywords: quadcopter, autonomous exploration, indoor, SLAM, wall detection

Designing an Autonomous Exploration Architecture for an Indoor Quadcopter

Jens Nyman

Supervisors: ir. Ken Caluwaerts, ir. Tim Waegeman
Promoter: prof. dr. ir. Benjamin Schrauwen

Abstract— **Autonomous exploration of buildings has many applications, especially in dangerous environments. Recently, the research community has shown an increasing interest in using quadcopters for this task. Our work focusses on creating a map of a set of rooms, using only a single camera as long-range sensor. Because of this very limited sensor hardware, the cost of the platform is reduced, which has a number of advantages like a low deployment threshold. We propose a novel architecture for indoor exploration. The important aspects of this architecture are video based localization and mapping, which we extended with multi-camera support, a room based methodology, meaning that we map every room separately, wall detection by applying EM to a layered room model and 3D visualization by mapping textures on walls.**

Keywords— **Quadcopter, autonomous exploration, indoor, SLAM, wall detection**

I. Introduction

THIS work addresses the problem of autonomous exploration with a low-cost quadcopter. The final goal is to enable a quadcopter to map an entire building without human interaction, using limited hardware. A possible application might be to gather information about damaged buildings after natural disasters. We use the *Parrot AR.Drone* as hardware platform, which has a single webcam as main long-range sensor.

Using only a webcam is a constraint with a high impact on the localization, mapping and visualization. Alternatives like laser range finders make the exploration task significantly easier. However, this platform choice also has a number of interesting advantages towards future use. First, this platform can be miniaturized because it has no large components. Moreover, the low-cost keeps the threshold for deployment low. A fire department may be reluctant to send 1000 dollar quadcopters with laser scanners into a burning home, but a simple 100 dollar quadcopter might be acceptable. Lastly, the low-cost makes it affordable to build a swarm of these robots. Swarms can speed up exploration considerably because they can investigate multiple spaces in parallel [1, 2].

In recent years, the research community has shown an increasing interest in quadcopters. Grzonka et al. [3] showed fast mapping with a laser range finder. Bachrach et al. [4] proved autonomous exploration on a similar platform. The area of low-cost quadcopter exploration has been less researched. Weiss et al. [5] implemented *Parallel Tracking and Mapping* (PTAM) by Klein and Murray [6] on a quadcopter equipped with a single webcam. They also created a textured 3D representation from the PTAM map [7].

II. Solution architecture

We propose a novel architecture for indoor exploration based on four important concepts:

- Video based localization and mapping
- Application of a room based methodology
- Wall detection based on a room model
- 3D visualization by mapping textures on walls

The room based methodology is apparent throughout this work and states that all rooms in a building should be treated separately. This allows for much more scalable mapping and easier visualization.

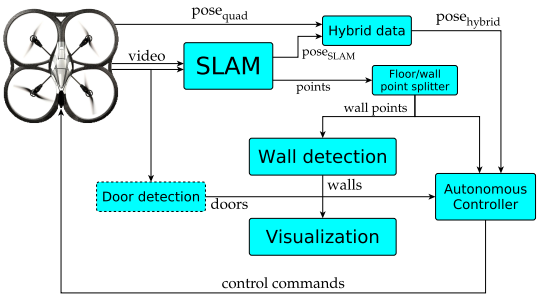


Fig. 1: Complete system architecture. The actual system architecture does not yet contain a door detector.

Figure 1 shows the complete system architecture. The most important blocks (SLAM, wall detection and visualization) will be discussed in further detail.

III. Simultaneous localization and mapping

We conducted a thorough study of the available video based *Simultaneous Localization And Mapping* (SLAM) algorithms, where we compared some software packages including EKFmonoSLAM [8] and PTAM [6] from a practical perspective. After careful deliberation, we chose PTAM as base algorithm.

Further, we integrated and evaluated the extension for VTOL airframes by Jama and Schinstock [9]. We also implemented some PTAM extensions, including support for multiple cameras. For this multi-camera extension, we developed a novel calibration procedure that uses CMA-ES [10].

IV. Wall detection

The PTAM algorithm generates a map of landmarks, which can be represented as a point cloud. We want to fit walls onto these points for visualization. Because of a lack of adapted line fitting techniques in literature, we constructed a room model enabling us to optimize the wall locations via the Expectation Maximization algorithm.

Inspired by the EM for Gaussian Mixtures implementation, we created a wall-sample model suiting the situation. To further enhance room detection, we created a layered room model, acting as prior for the wall placement.

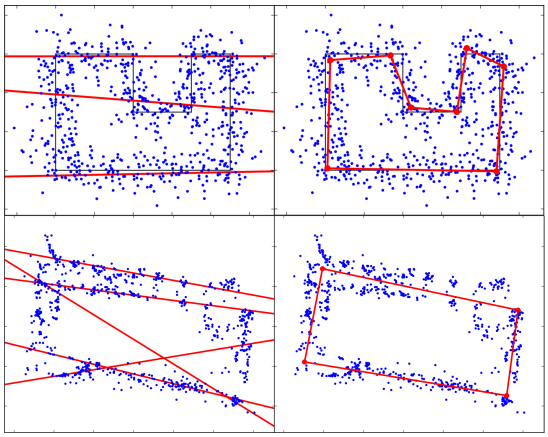


Fig. 2: Wall detection comparison. Left column: statistical learning approach [11], right column: our approach. Upper row: computer generated test case, lower row: real world dataset recorded in rectangular room.

Figure 2 shows an extract of an extensive comparison, in which we compare our algorithm with the Hough transform [12, 13] and the statistical learning approach [11], which is a recent state-of-the-art line fitting technique. As the extract suggests, our approach outperforms the others.

V. Visualization

The generated set of walls is visualized in 3D. To give the user an impression of what the room looks like, textures are mapped onto the walls and the floor, using PTAM information and camera frames. Figure 3 shows an example visualization, applied to multiple rooms.

VI. Conclusions

By implementing the largest part of the architecture, we were able to uncover the remaining barriers for reaching the final goal. We found that the bottleneck of our system is the SLAM algorithm. We also achieved some promising results in the individual blocks. The wall detection method turns out to perform better than the (to our knowledge) available algorithms and the multi-camera calibration procedure works well and can be applied to other problems as well.

Fig. 3: 3D visualization of three rooms.

References

[1] A.T. Hayes, A. Martinoli, and R.M. Goodman, "Comparing distributed exploration strategies with simulated and real autonomous robots," in *Proc. of the Fifth Int. Symp. on Distributed Autonomous Robotic Systems DARS-00*, 2000.

[2] I. Rekleitis, G. Dudek, and E. Milios, "Multi-robot collaboration for robust exploration," *Annals of Mathematics and Artificial Intelligence*, vol. 31, no. 1, pp. 7–40, 2001.

[3] S. Grzonka, G. Grisetti, and W. Burgard, "Towards a navigation system for autonomous indoor flying," in *IEEE International Conference on Robotics and Automation (IRCA)*. Ieee, 2009, pp. 2878–2883.

[4] A. Bachrach, R. He, and N. Roy, "Autonomous flight in unknown indoor environments," *International Journal of Micro Air Vehicles*, vol. 1, no. 4, pp. 217–228, 2009.

[5] M. Blosch, S. Weiss, D. Scaramuzza, and R. Siegwart, "Vision based mav navigation in unknown and unstructured environments," in *IEEE international conference on Robotics and automation (ICRA)*. IEEE, 2010, pp. 21–28.

[6] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *ISMAR 2007. 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*. IEEE, 2007, pp. 225–234.

[7] S. Weiss, M. Achtelik, L. Kneip, D. Scaramuzza, and R. Siegwart, "Intuitive 3d maps for mav terrain exploration and obstacle avoidance," *Journal of Intelligent & Robotic Systems*, vol. 61, no. 1, pp. 473–493, 2011.

[8] A.J. Davison, "Real-time simultaneous localisation and mapping with a single camera," in *Ninth IEEE International Conference on Computer Vision. Proceedings*. Ieee, 2003, pp. 1403–1410.

[9] M. Jama and D. Schinstock, "Parallel tracking and mapping for controlling vtol airframe," *Journal of Control Science and Engineering*, vol. 2011, 2011.

[10] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary computation*, vol. 9, no. 2, pp. 159–195, 2001.

[11] T.J. Chin, H. Wang, and D. Suter, "Robust fitting of multiple structures: The statistical learning approach," in *IEEE 12th International Conference on Computer Vision, 2009*. IEEE, 2009, pp. 413–420.

[12] P.V.C. Hough, "Method and means for recognizing complex patterns," Dec. 18 1962, US Patent 3,069,654.

[13] R.O. Duda and P.E. Hart, "Use of the Hough transformation to detect lines and curves in pictures," *Communications of the ACM*, vol. 15, no. 1, pp. 11–15, 1972.

Contents

Abbreviations

CMA-ES	Covariance Matrix Adaptation Evolution Strategy
DTAM	Dense Tracking And Mapping
DoF	Degrees of Freedom
EKF	Extended Kalman Filter
EM	Expectation Maximization
GMM	Gaussian Mixture Model
IMU	Inertial Measurement Unit
PCA	Principal Component Analysis
pdf	Probability Density Function
PTAM	Parallel Tracking And Mapping
PTAMM	Parallel Tracking And Multiple Mapping
PTAMMV	PTAMM for VTOL airframe
RANSAC	RANdom SAmple Consensus
RGB-D	Red Green Blue - Depth
ROS	Robot Operating System
SLAM	Simultaneous Localization and Mapping
SSE	Sum of Squared Errors
UAV	Unmanned Aerial Vehicle
VTOL	Vertical TakeOff and Landing

Introduction | 1

Imagine a disaster happening in a building. This might for example be a fire or an earthquake. You, as a crisis manager, want to investigate the building to look for survivors or to make a damage report. However, it's not safe to allow people to enter the building (because of radiation, danger of collapse, ...).

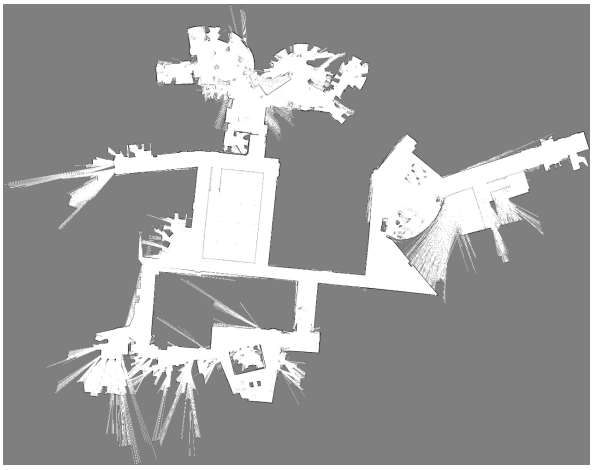


Figure 1.1: 2D grid map generated by a laser range finder [1].

In such a situation, you might have the tendency to send in a ground robot. Ground robot exploration is an established research domain. A typical approach would be to use a laser range finder to perform localization and create a 2D grid map of the environment (like Figure 1.1). While obtaining a high mapping precision, these robots have the disadvantage that they can't pass obstacles like stairs and furniture. Suppose that all glass windows are broken, but the door still is closed. In this case, a ground robot can't even enter the building.

To increase movement flexibility, one could switch from ground robots to **flying robots**. A suitable flying robot would for example be a quadcopter, because it can stay quite stable in the same position and its motion control is relatively easy. Flying robots of course don't have trouble with stairs. It has even been shown by Mellinger et. al. that quadcopters can fly through very small windows [2] (Figure 1.2). It should be noted that switching to flying robots means increasing the dimensionality of the localization problem. This is because a flying robot has more degrees of freedom than a ground robot.

Further, we want to avoid using **heavy or expensive sensors**. Heavy sensors are a problem because the payload capacity of a quadcopter is very limited. Moreover, such sensors

(a) (b) (c)

Figure 1.2: Quadcopter flying through a window [2]. Note that an external motion capture system is used for localization.

prevent the robot from scaling down. Small robots, however, have the advantage of being even more flexible. Excluding expensive sensors ensures that the robot platform remains cheap. It is easier for cheap robots to make it into commercial applications, but the major advantage is that it is affordable to build a swarm out of cheap robots. Swarms can speed up exploration considerably because they can investigate multiple spaces in parallel [3,4]. Even when price restrictions lead to less robust exploration of an individual, the entire group may turn out to generate a more robust and richer map.

When entering a building, the robot will easily lose its network connection. Hence, it is not always possible to steer the robot by hand. In the case of a swarm, it is infeasible to manually steer all robots, even with perfect network connectivity. So, as a final restricting factor, we desire the robot to **explore autonomously**.

1.1 Problem statement

The final goal of this and further work is to enable a quadcopter with limited sensors to visualize an entire building without human interaction.

The requirement of using **limited sensors** means that we don't use expensive sensors like laser range finders. Instead, we rely on simple webcams, IMUs and gyroscopes.

Excluding **human interaction** is the hardest constraint. It requires that every subsystem works autonomously and is robust enough. A subsystem should only fail rarely and be able to recover quickly.

Determining the nature of the **visualization** is part of the problem. We refined this definition to creating a room based map of the environment, consisting of textured walls (like in Figure 1.3). This decision is motivated in chapter 4.

1.2 Solution overview

The core elements of our solution are:

Figure 1.3: Example of textured walls, created by a ground robot with a laser scanner [5].

- Localization and mapping, based on video *(chapter 5)*
- Obstacle avoidance with acquired map *(section 7.2)*
- Room based methodology *(section 4.1)*
- Wall detection and visualization *(chapters 8 and 9)*

The rest of this work is organized as follows: Chapter 2 discusses previous work in the field. Chapter 3 explains in more detail the available hardware. Chapter 4 gives an overview of the entire architecture. Chapter 5 gives an extensive study of SLAM algorithms, including our improvements and chapter 6 explains how the SLAM data is post-processed. In chapter 7, we discuss how we implement the intelligence for autonomous flight. A new algorithm for wall detection is presented in chapter 8 and chapter 9 shows how a room is visualized. Concluding remarks are given in chapter 10.

State-of-the-art | 2

The topics of this thesis are not confined into one research area. In this chapter, we give a broad view on previous work in the main domains. Whenever necessary, subsequent chapters will describe the state-of-the-art of more narrowly defined problems.

2.1 Localization and mapping

The problem of simultaneous localization and mapping (SLAM) is one of the key technologies for autonomous robotics and is a research area on its own. SLAM addresses the problem of a robot seeking to construct a map of the environment, while at the same time localizing itself in the map. Almost every SLAM algorithm makes use of *landmarks*. These are recognizable objects, surfaces, corners etc. in the environment that are used to localize the robot.

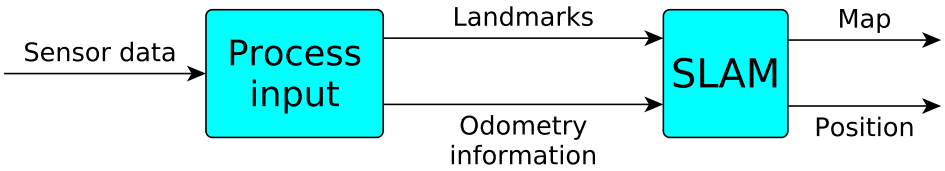


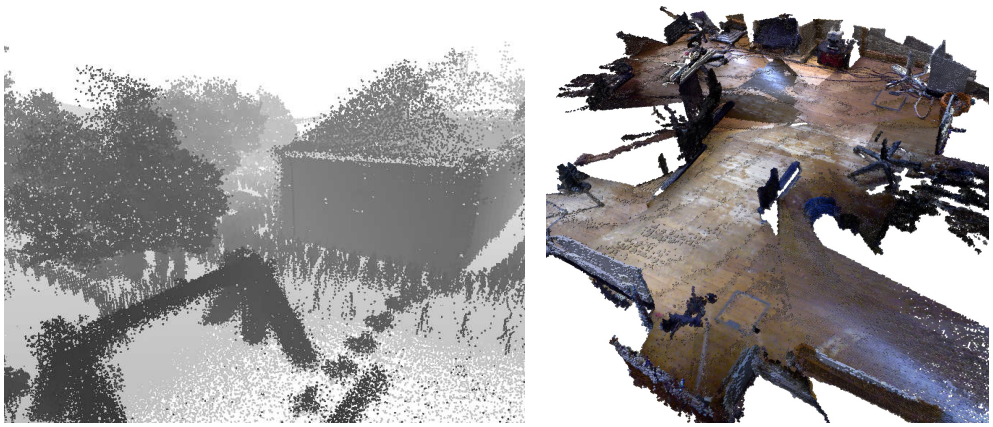
Figure 2.1: SLAM usage.

Figure 2.1 show a typical framework for performing SLAM. Many theoretical works only consider the SLAM block and assume given landmarks and odometry information [6–8]. Important progress on this part will be discussed. It should however be noted that the nature of the landmarks heavily depends on the available sensors. We will discuss what has been shown with different sets of sensors.

2.1.1 Sensor selection

Before assessing which sensors are appropriate, it is important to realize that one first has to define the number of degrees of freedom. A typical ground robot has three degrees of freedom (x,y position and angle at which robot is turned) and needs 2D landmarks [9, 10]. A flying robot has six degrees of freedom (x,y,z position and the roll, yaw and pitch angles) and needs 3D landmarks [11–13].

For ground robots, a laser range finder is often used to create accurate occupancy grids [9,10, 14]. 6D SLAM however, requires sensors with higher dimensionality. The main approaches are the use of a 3D laser scanner, regular cameras or a combination of both.



(a) (b)

Figure 2.2: Map visualizations. (a) SLAM6D of Nüchter et al.: Point cloud visualization of the map [15]. (b) RGB-D SLAM of Endres et al.: Occupancy grid visualization of the map [16].

Nüchter et al. [15,17] use a laser scanner to create detailed 3D maps of the environment. The algorithm tries to generate a consistent 3D point cloud map which is shown in Figure 2.2(a).

Endres et al. [16, 18] use an RGB-D camera like the Microsoft Kinect (Figure 2.3(a)). They combine depth image techniques with pairwise feature matching in the RGB images. Figure 2.2(b) shows an example of an occupancy grid map generated by this algorithm.

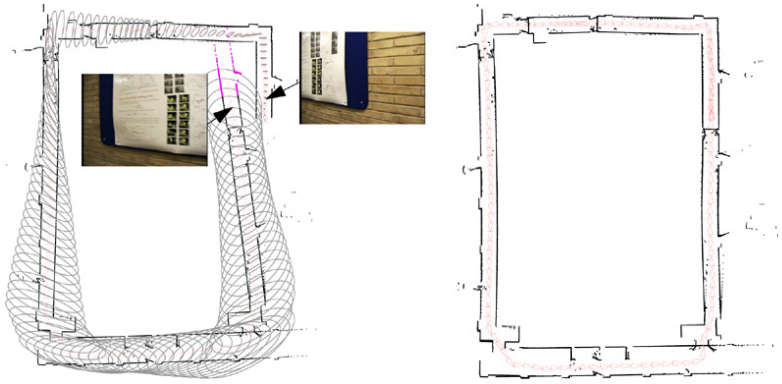
(a) (b)

Figure 2.3: Input devices. (a) Microsoft Kinect, picture provided by Microsoft. (b) ATRV rover Dala equipped with stereo camera [19].

Depth cameras can thus generate accurate maps, but they are usually quite expensive. Scalable and stable solutions have however also been shown with stereo pair cameras. [19] and [20] use two regular cameras in parallel as stereo pair (Figure 2.3(b)). A large advantage of this approach is that the scale of the scene can be determined by measuring the constant distance between the cameras. Monocular SLAM (next paragraph) cannot estimate this scale without additional sensor data.

To conclude this section, it is also possible to perform SLAM with just one camera (monocular SLAM or monoSLAM). Davison [21] uses an extended Kalman filter to optimize the

robot and landmark positions. Klein and Murray [22] use the computationally expensive *bundle adjustment* as background optimizer while preserving real-time tracking. Both these techniques will be described in further detail in chapter 5 and the latter will be called *Parallel Tracking and Mapping* (PTAM).

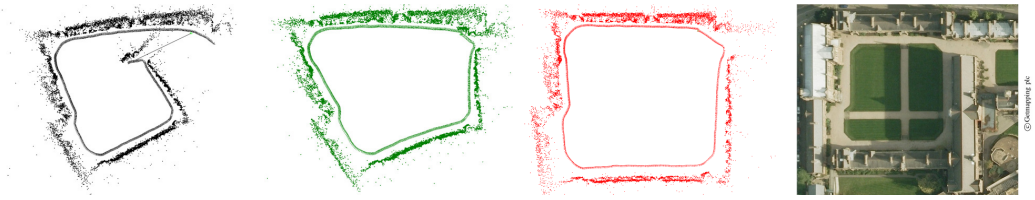


(a) before loop closure (b) after loop closure

Figure 2.4: Typical loop closure problem [23].

2.1.2 Performing SLAM

Most SLAM algorithms face the problem that the initial guess of landmark positions is noisy. This causes inconsistencies in the map. An example of such an inconsistency is the loop closure problem (Figure 2.4). In this problem, the robot comes back to a previously visited place but the position estimate has shifted from that place. The challenge is then to detect this and to properly adjust the map.



(a) before optimization (b) 6 DoF optimization (c) 7 DoF optimization (d) areal photo

Figure 2.5: Scale drift-aware monocular SLAM [24].

Typical solutions to obtain a consistent map are probabilistic. We already mentioned the extended Kalman filter and bundle adjustment. These are examples of typical algorithms to obtain consistency. An interesting alternative approach for monocular SLAM is the 7 degrees-of-freedom (7 DoF) optimization proposed by Strasdat et al. [24]. They introduce an extra DoF to determine the scale drift that typical monocular SLAM can have. One of the promising results is shown in Figure 2.5.

A second recurrent problem is scalability. Generally, SLAM algorithms become slower when the number of landmarks increases. This means that this number is limited, which can be an unacceptable constraint. Estrada et al. [25] split the map into smaller local maps and tie these

together on a higher level. This hierarchical SLAM can accurately map large environments in real time.

2.2 Quadcopter mapping

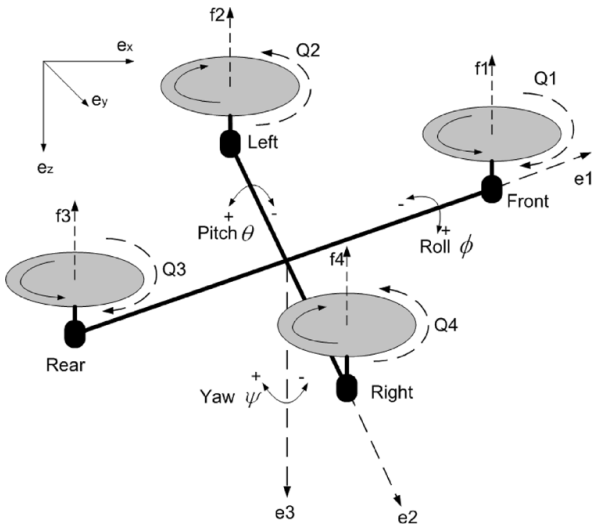
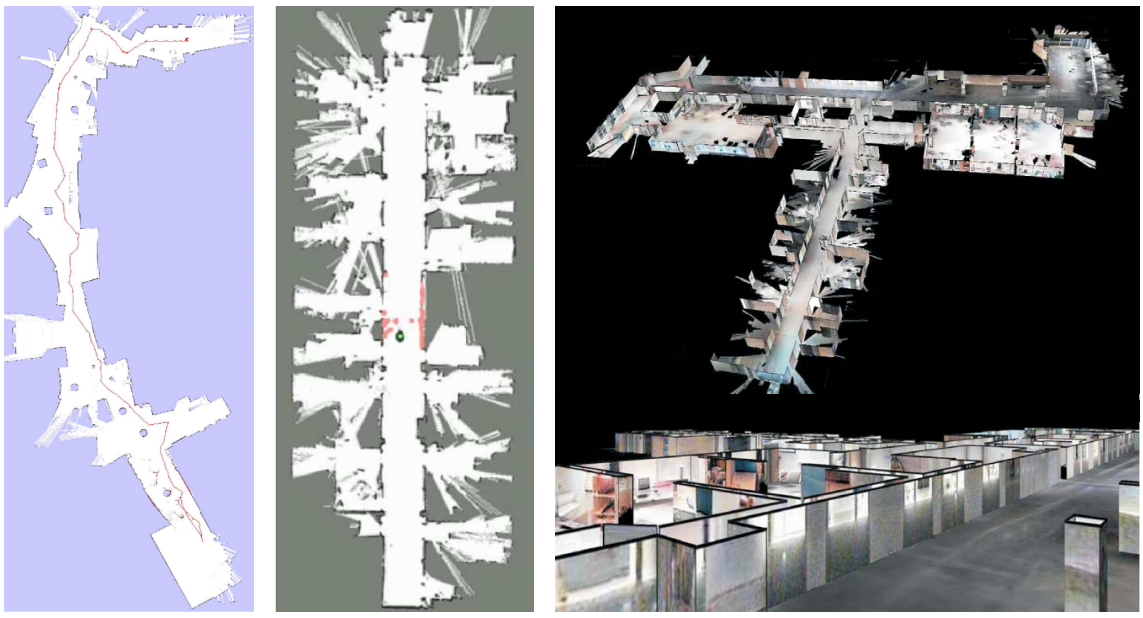


Figure 2.6: Definition of roll, pitch an yaw angles [26].

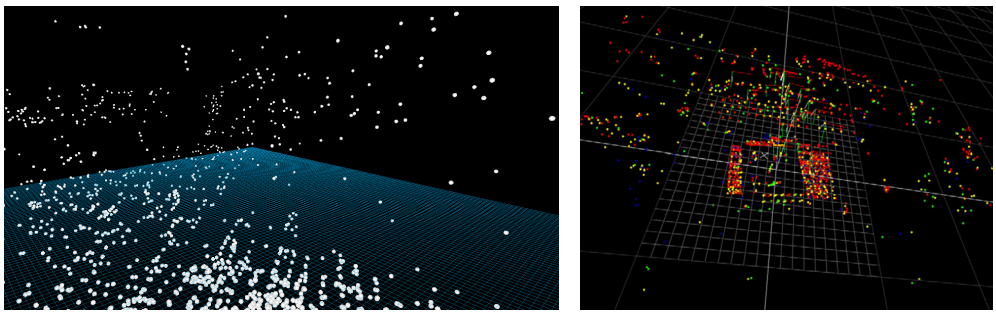


(a) (b) (c)

Figure 2.7: 2D maps generated by 3D SLAM. (a) and (b) Generated by quadcopters with laser range finder [27, 28], (c) generated by ground robot with laser scanner and panoramic camera, 3D visualization of map [5].

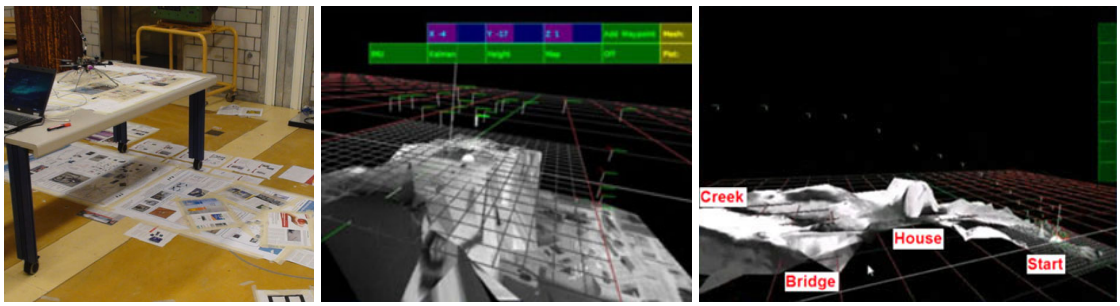
In section 2.1, we stated that flying robots have six degrees of freedom. Some works however, neglect the roll and pitch angles when mapping (see Figure 2.6 for angle definitions). This approach is only roughly valid for a quadcopter, since it can't move around without

tilting at least a little. Typically in such works, the robot is equipped with a laser range finder [27, 28]. This enables them to perform similar mapping algorithms as for ground robots. Examples of such maps can be seen in Figure 2.7(a) and (b). Biber et al. [5] showed with a ground robot that, when combined with a panoramic camera, such planar SLAM can lead to accurate and intuitive 3D maps (Figure 2.7(c)).



(a) self made (b) by Weiss et al. [13]

Figure 2.8: Examples of point clouds generated by PTAM.



(a) Photo of table (b) Map of table (c) Outdoor map

Figure 2.9: Examples of the intuitive 3D maps of Weiss et al. [13]

Previous work that allows more general quadcopter movement makes use of 6DoF SLAM, generating a 3D map. Weiss et al. [12, 13] and Jama et al. [29] showed successful implementations of PTAM (or derived algorithms) for quadcopters only equipped with one camera. A problem with these approaches for visualization is that the map consists of a sparse point cloud (Figure 2.8) and it is hard to see the underlying structure in these points. Weiss et al. partly solved this issue by applying Delaunay Triangulation to the points and mapping texture to the mesh [13]. An example of such maps can be seen in Figure 2.9.

Better maps are found when using 3D laser scanners. These yield similar maps as in Figure 2.2 and implementations on unmanned areal vehicles (UAVs) have been proven by Thrun et al. on a remote helicopter [30] and by Morris et al. on a quadcopter [31].

2.3 Autonomous quadcopter exploration

Autonomous exploration for quadcopters has been shown by Bachrach et al [27]. Since they use laser range finder-based localization with a 2D occupancy grid map, the exploration

challenges are similar to those of ground robots. They follow a classical approach of frontier-based exploration [32] and trajectory planning using dynamic programming.

For robots without laser sensors, autonomous UAV exploration based on SLAM has not yet been shown (to our knowledge). On the other hand, Bills et al. [33] looked at the problem differently and implemented autonomous flight without a map. Instead of performing localization or mapping, they classify video frames into common indoor classes like stairs and corridors. Thereafter, the UAV performs some predefined movement (like a turn in a corner) and avoids the walls using camera frame analysis. This approach enables a quadcopter to explore part of a building without real-time localization. It could be possible to create a map afterwards using offline structure-from-motion techniques[1] but navigational feedback would then be impossible.

[1]Structure-from-motion techniques are very similar to visual SLAM techniques but are usually performed offline.

Hardware platform | 3

We use the *AR.Drone* made by *Parrot* as test platform. We will first introduce this platform as a whole before going into more detail about the components.

3.1 Parrot AR.Drone

Parrot is a multinational company headquartered in Paris employing more than 650 people worldwide[1]. Their core business is designing wireless devices for mobile phones, but in 2010 they introduced the AR.Drone.

(a) (b)

Figure 3.1: The Parrot AR.Drone. Pictures provided by Parrot.

This quadcopter (Figure 3.1) was mainly designed as a toy for children, but it immediately caught the attention of electronics hobbyists and researchers. This is mainly because the drone can hover fairly stable on itself so the user can focus on navigation. The quadcopter also carries some useful and accessible hardware (see next section). These features combined with a relatively low price[2] make it a valuable research object.

3.2 Hardware components

The quadcopter contains the following hardware components, relevant for this work:

[1] This information is stated in the fourth quarter earnings report (2011) of Parrot
[2] Reservoir Lab bought the AR.Drone for 300 €

- Front camera: 640 x 480 pixels, 30 Hz
- Ground facing camera: 320 x 240 pixels, 60 Hz
- IMU (Intertial Measurement Unit): reports velocity and orientation, using a combination of accelerometers and gyroscopes
- Ultrasonic height sensor
- WiFi access point
- Computer with ARM architecture, running a Linux kernel

(a) font camera

(b) bottom of AR.Drone, the ground facing camera is the dot in the middle

Figure 3.2: The two cameras of the AR.Drone.

The most important parts for our research are the two cameras (Figure 3.2). Due to firmware restrictions, the standard setup doesn't allow us to stream both cameras simultaneously. Appendix A describes the tested remedies and concludes that it is cumbersome but possible to get two video streams from a flying AR.Drone.

The on-board Linux computer can cope with easy calculations, but is not suited for more involved calculations (like visual SLAM algorithms). This is why all algorithms described in this work are calculated on a laptop[3], unless explicitly mentioned. When running a live test, the laptop connects to the WiFi access point of the AR.Drone, gets the relevant sensor information and sends the control commands. The quadcopter-side software for this is described in appendix A. The PC-side software is either self-written or based on `python-ardrone`[4].

3.3 Wide angle lens

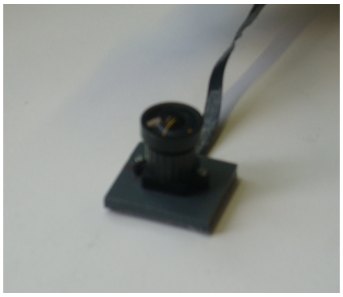
In order to improve the performance of the SLAM algorithms that we will describe later, we try to increase the viewing angle of our camera. This means that we try to change the bal-

[3] laptop specifications: 2.4 GHz processor with 4 GB RAM.

[4] `python-ardrone` is an open source client for the AR.Drone standard firmware and was written by Bastian Venthur (`https://github.com/venthur/python-ardrone`).

ance between resolution and coverage. It's clear that a larger coverage facilitates mapping. Practically, we try to attain this by replacing the camera lens with a wide angle lens.

(a) Detachable fish eye lens for mobile phones. (source: Google Store)



(b) Self-constructed stack of lenses.

Figure 3.3: Wide angle hardware.

(a) With regular lens

(b) With stacked lenses

Figure 3.4: Comparison of the image of the same scene with different lenses.

First, we tried to add a detachable fisheye lens (Figure 3.3(a)) to the front camera. This didn't work because the resulting images couldn't be focused on a scene.

In a second attempt, we stacked three lenses using a self-made placeholder, with one of the lenses a wide-angle one (Figure 3.3(b)). This construction was necessary to ensure a focused image. The result can be seen in Figure 3.4. The blurry part of the image on the left is due to a partly damaged lens. Preliminary experiments show that SLAM works surprisingly well with this augmentation. However, due to the blur and the black edges, we mostly used a regular lens for the experiments in this work.

3.4 Quadcopter control

As stated earlier, the AR.Drone is able to hover quite stable out of the box. The manufacturer achieves this by combining IMU data with flow detection on the ground facing camera. This flow detection tries to compensate for accelerometer drift.

Using the standard firmware, the user is able to steer the quadcopter to all directions and

turn the quadcopter left and right. Note that when the robot is actuated, the drift compensating flow detection is no longer active. This can lead to significant drift, e.g. while turning.

Solution architecture | 4

This chapter motivates some architectural choices that we made. Thereafter, we will discuss the entire architecture. First the functional architecture and then the software architecture.

4.1 Room based methodology

Our exploration and visualization strategy is room based. This means that we treat rooms as independent entities, connected by doors. Where a traditional approach generates one global map, ours generates a set of maps with interconnections.

This has a number of advantages:

- The total size of the map is virtually unlimited. This is completely different in large maps, where the map size has a direct impact on processing time.
- For camera-based SLAM, it's natural to start a new map in a new room. Indeed, we wouldn't want patterns in one room to be recognized in another room. In other words, when moving on to another room, the information of the previous room is obsolete.
- Loop closure is easier. The computationally heavy matter of loop closure for regular maps is no longer needed. Instead, we only have to recognize recurring rooms and make interconnections. Detecting previously visited places is significantly easier. We could for example use FAB-MAP [34] which can distinguish a visited place from a new place, only by using camera images. The results are quite impressive and implementations are freely available.
- The assumption of detecting rooms makes wall detection easier. This will be clarified in chapter 8.

Switching to this methodology also opens up a new problem: we have to detect doors. These are necessary for autonomous control and making interconnections. Detection will probably have to be done by video analysis, but this is yet to be investigated.

4.2 Visualization choices

As we suggested earlier, visualization is closely tied with the kind of SLAM algorithm that is used. The SLAM algorithm that we will choose in chapter 5 (PTAM) has a map consisting of noisy points. Chapter 2 discussed how Weiss et al. created an intuitive 3D map of these points [13]. It might have been logical to do the same thing for our visualization. However,

we argue that this would not work for our situation. In our experience, the data that results from this SLAM algorithm (PTAM) is corrupted with too much noise. A derived 3D mesh would be ugly and contain too many artifacts to be practically applicable.

Instead of a 3D map, we chose to remove the z-dimension from the surface model. This leaves us with a set of walls. Upon visualization, we return to three dimensions and map video frames onto the walls. In other words, we want to visualize the environment just like Biber et al. did with a laser scanner and a panoramic camera (Figure 2.7(c))

4.3 System architecture

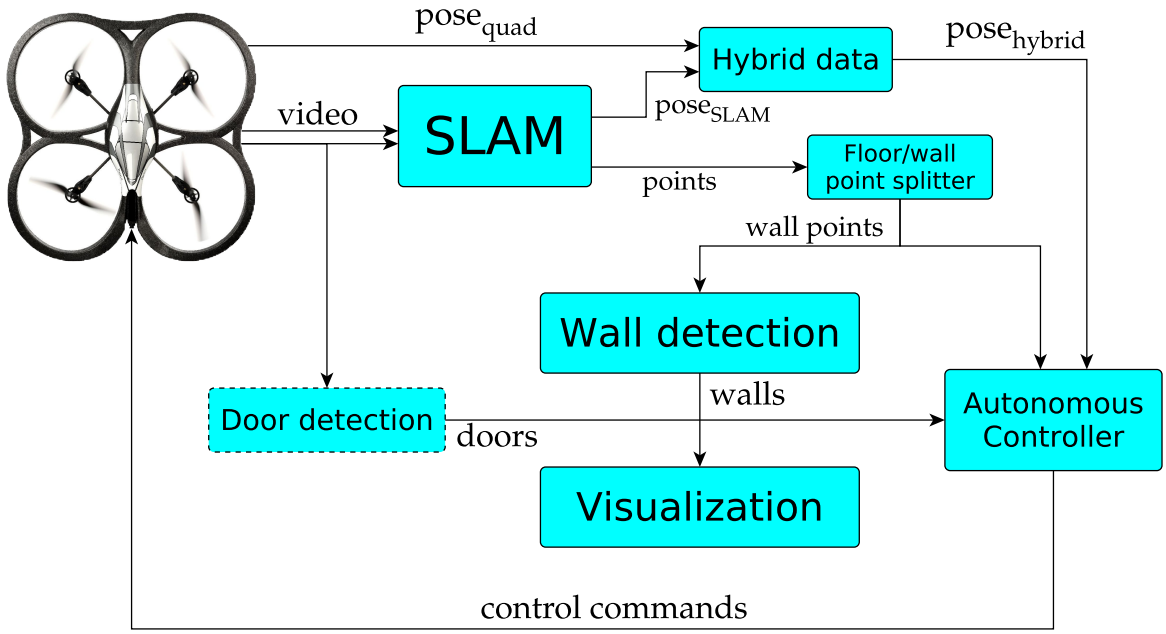


Figure 4.1: Complete system architecture. The actual system architecture does not yet contain a door detector. Note that we left out some more subtle arrows for simplicity.

Figure 4.1 shows the complete system architecture. The quadcopter generates video and IMU data where the former is used by visual SLAM. The pose estimate from this algorithm combined with the IMU estimate to generate a hybrid pose. The SLAM points (i.e. landmarks) are split into floor and wall points. These are used by the wall detector, preparing the data for visualization and the autonomous controller, using the available data for commanding the robot.

The most important blocks for this work are the SLAM block and the wall detection block. The block specifications will be further clarified in the next chapters.

4.4 Software architecture

The software behind the functional architecture consists of independently running processes. These processes are part of an event driven architecture, and communication between them is based on message passing. Passing messages is a very natural way of com-

municating information like sensor data and video. Note that in the system architecture, all arrows are actually topics on which messages can be posted.

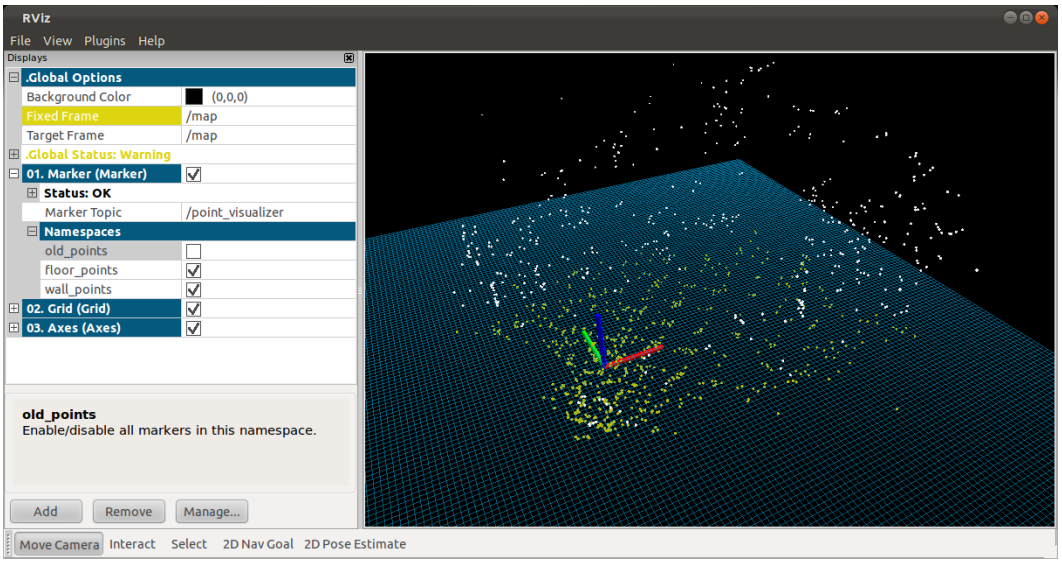


Figure 4.2: Screenshot of rviz.

We use the *Robot Operating System* (ROS) as software platform. ROS is a framework and a toolbox for the development of robot applications. The framework has many features including the aforementioned message passing. The toolbox is very extensive and we use many utilities from it. The most important ROS tool for our work is rviz, which is a visualization tool (Figure 4.2). The plus point about rviz is that it can update the pose of the quadcopter and the point positions in real time.

Visual SLAM | 5

In chapter 2, we discussed some theoretical advances in SLAM. This chapter will evaluate the practical value of existing algorithms for our purpose, i.e. visual SLAM for UAV exploration. After careful deliberation, we chose PTAM as base algorithm. We motivate this choice and discuss our attempts to improve this algorithm.

5.1 Mono vs. stereo

Performing SLAM with one camera or a stereo pair is in many ways very different. This is because a stereo pair can immediately estimate the depth of an image, even without a map, while monocular SLAM needs a high quality map for accurate depth estimation. Another large advantage of stereo SLAM is that it can estimate the metric scale of the scene using the fixed stereo separation.

On the other hand, using one camera also has its advantages. The hardware is smaller, easier to setup and cheaper. Furthermore, given a certain stereo separation, stereo SLAM accuracy decreases with increasing scene size. This is because both cameras must return a significantly different image from the same scene. It is clear that stereo SLAM in a $10\,\mathrm{m} \times 10\,\mathrm{m}$ scene will not work with less than 10 cm separation (unless the low separation is compensated by a high resolution).

We chose to follow the single camera approach because of these considerations.

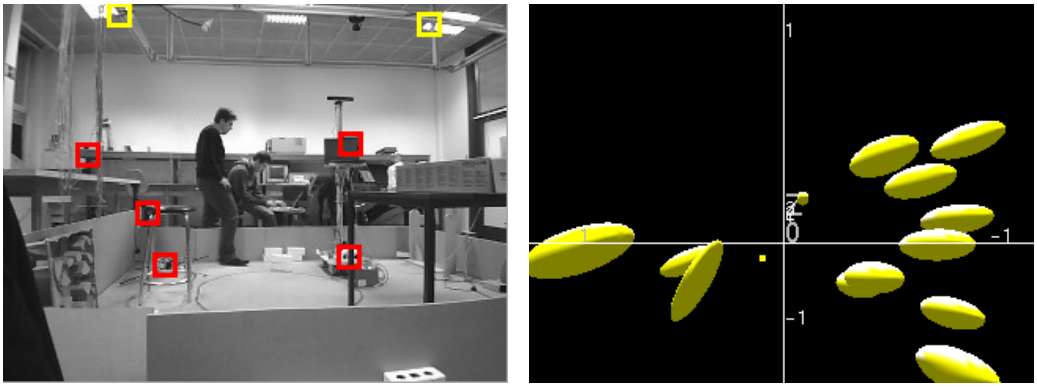
5.2 Evaluation of previous work

EKFmonoSLAM

Andrew Davison developed a Bayesian framework for single-camera SLAM based on the Extended Kalman Filter (EKF) [21]. This approach is based on traditional EKF SLAM, where a covariance matrix is updated every time step [6]. This matrix contains correlations between the landmarks and the current pose. Due to the exploitation of these correlations when optimizing, high quality landmarks are found. The drawback is that the covariance matrix grows with each landmark. This means that the SLAM algorithm scales with the third power of the number of landmarks. Consequently, the number of landmarks are seriously constrained and about 10 landmarks are typically visible in a video frame.

Davison made the source code available on his website[1]. We integrated it in our framework

[1] Davison's website: `http://www.doc.ic.ac.uk/~ajd/Scene/index.html`



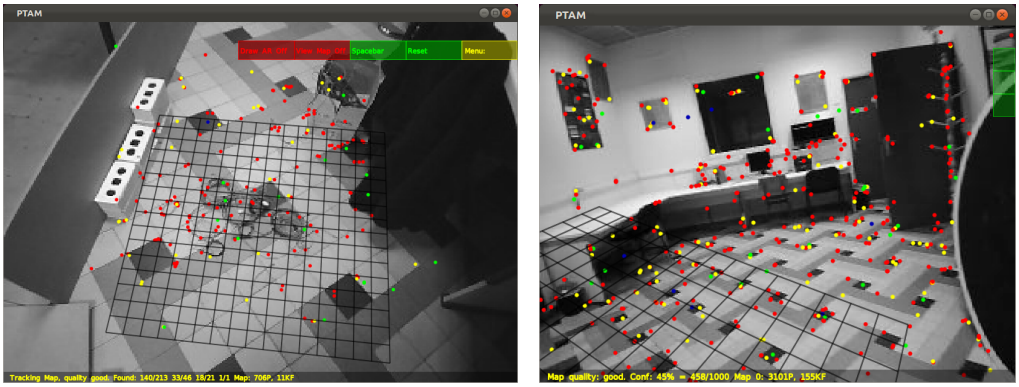
(a) Typical video frame, landmarks are indicated by the boxes.

(b) 3D map of landmarks. The covariance of the individual points are visualized.

Figure 5.1: Example of EKFmonoSLAM usage.

and Figure 5.1 shows a typical usage. Early testing immediately showed that the algorithm easily loses its focus, even with slow movement. This indicates that EKFmonoSLAM doesn't work 'out of the box' and parameter tuning is probably necessary to get its robustness to an acceptable level.

PTAM and PTAMM



(a) (b)

Figure 5.2: Example of PTAM usage. The dots are landmarks and the grid is part of the augmented reality that PTAM is capable of (i.e. virtual objects can be drawn on the video as if they were really there).

In 2007, Klein and Murray published a new algorithm called *Parallel Tracking And Mapping* (PTAM) [22]. As the name suggests, the principal idea is that the map is made in a separate thread so the mapping can run on at a slower frequency than the tracker. This is fundamentally different from EKFmonoSLAM, where the map is updated at the same frequency as the current pose.

Unlike EKFmonoSLAM, the map consists of points without probabilistic information (Figure 5.2). Tracking is done by finding the most consistent camera pose, given the current measurements. Important to know is that the map is expanded by the addition of a *keyframe*. A

keyframe is a video frame that the mapping thread uses for map point generation and optimization. The tracker decides when a new keyframe is needed. Typically a map will contain about 40 frames after a three minute session.

Because the mapper runs in a separate thread and works on a limited number of keyframes, a computationally expensive optimization algorithm can be used. PTAM uses the Levenberg-Marquardt bundle adjustment [35, Appendix 6.6]. This technique tries to optimize the landmark locations, based on maximal consistency with the keyframes. It is typically used in offline algorithms, but because of the setup, it can also be used here. Practically, this means that when a frame is added, the landmark position estimates are rough. Some seconds later, this estimate is improved when the first bundle adjustment run is complete.

The source code of PTAM is available online[2]. Castle et al. [36] later improved this code with the possibility to manage multiple maps and named it *Parallel Tracking And Multiple Mapping* (PTAMM). This project is also available as open source[3].

We integrated both packages in our environment and noted a similar performance. Unlike EKFmonoSLAM, PTAM(M) works 'out of the box' and no parameters need to be tuned. The algorithm can cope with low resolution, high frequency cameras as well as with standard VGA resolution, 30 Hz webcams.

PTAMM for VTOL Airframe (PTAMMV)

While PTAMM performs quite good, there are some drawbacks for usage in UAV's. The main problems are the map expansion and the cumbersome initialization. Jama and Schinstock partly solved these problems in their PTAMM improvement [29].

Initialization

The PTAMM initialization procedure requires the user to press the space bar and then slowly translate the camera. In the meantime, PTAMM keeps track of a set of features (Figure 5.3). When the space bar is pressed a second time, the algorithm has a set of corresponding points in two frames and creates a map with the resulting depth information.

This is cumbersome for quadcopters because gentle movements are often not possible. The result is that the initialization procedure has to be done many times before it is successful. By matching SURF features, Jama et al. allow the user to initialize the map solely by two frames without tracking in between. Our own experiments confirm that initialization is much easier but detect a small performance penalty (see section 5.3).

Map expansion

PTAMM works well when moving around an object while viewing it from different locations. However, it does not work well when new areas are viewed by rotating. The map can not be expanded this way because a stereo view of a feature is necessary to define its location.

[2]PTAM source: `http://www.robots.ox.ac.uk/~gk/PTAM/`

[3]PTAMM source: `http://www.robots.ox.ac.uk/~bob/software/index.html`

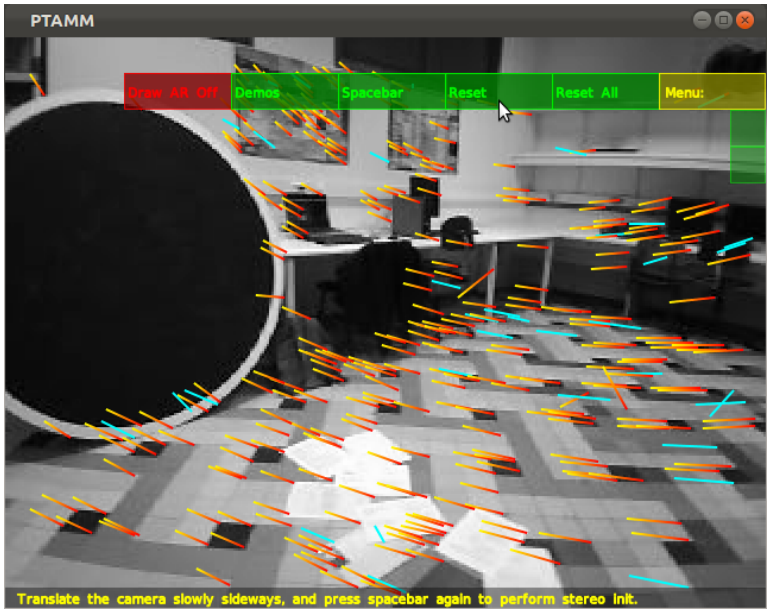


Figure 5.3: PTAMM initialization procedure. The lines represent the translation of tracked points.

Jama and Schinstock improved map expansion by changing the keyframe addition criterion. Measurements in section 5.3 show a slight performance improvement on a realistic dataset.

The source code of these improvements is not available online, but we received it via mail correspondence, thanks to Michal Jama. We will further refer to this algorithm as PTAMMV (PTAMM for Vertical takeoff and landing airframe).

DTAM

(a) Camera frame (b) 3D reconstruction: meshed depth image

Figure 5.4: Results of DTAM [37]. Source: YouTube (`youtube.com/watch?v=Df9WhgibCQA`).

In 2011, Newcombe et al. published some interesting results of their depth based visual SLAM, called *Dense Tracking And Mapping* (DTAM) [37]. As shown in Figure 5.4, scenes can be reconstructed in 3D with extremely high precision. Moreover, the algorithm is more robust than PTAM, according to the authors. Unfortunately, the source code of this framework is not available on the internet.

5.3 PTAM experiments

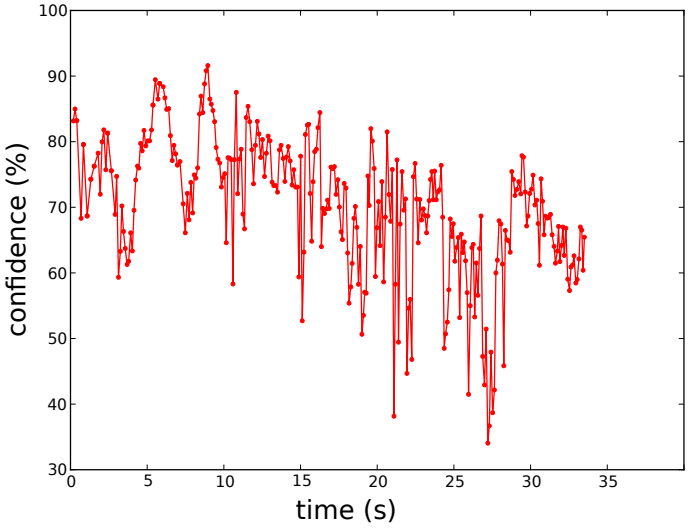


Figure 5.5: Typical confidence development during the fixed dataset. The negative peaks correspond with quadcopter maneuvers.

In order to measure the performance of PTAM and derived software, we defined a measure to assess the tracking quality. For this, the *tracking confidence* is defined as:

$$\text{confidence} = \frac{\#\text{ matched landmarks}}{\#\text{ landmarks that should be visible}} \tag{5.1}$$

where everything is defined over one frame. A similar measure is used in PTAM to measure the tracking quality. The confidence is evaluated when performing PTAM on a fixed dataset, which we recorded at Reservoir Lab by manually steering a flying AR.Drone. The confidences of the first 300 samples are averaged out and become the performance measure. A typical confidence development for the dataset can be found in Figure 5.5.

	Performance	
	Average (%)	**Stdev (%)**
PTAM	70.77	3.18
PTAMM	69.41	3.49
PTAMM with PTAMMV init.	65.40	5.81
PTAMM with PTAMMV init. and map expansion	68.83	4.35

Table 5.1: Performance comparison of different algorithms. Average confidence was computed 20 times, so robustness can be evaluated by looking at the standard deviation.

This performance measure, applied to the aforementioned algorithms and improvements, is showed in Table 5.1. Note that we measured the tracking performance 20 times in order to check how reproducible the solution is. The standard deviations are different from zero because there is always some degree of randomness, for example in the asynchronous arrival of video frames.

As expected, we see no significant difference between PTAM and PTAMM performance. PTAMMV results show a performance drop with the easier initialization and an increased performance with new map expansion.

5.4 Algorithm choice

The assessment of previous work (section 5.2) result three options for our algorithm choice:

1. Use a SLAM algorithm that is hard to get working, like EKFmonoSLAM
2. Use PTAM with PTAMM and PTAMMV improvements
3. Write an own visual SLAM implementation, e.g. based on the DTAM paper [37]

We chose option 2, because it was clearly the most feasible option available. The early experiments described above, also show that the algorithm works quite well. This option has the additional advantage that PTAM has already been proven to work with quadcopters [12, 13, 29].

5.5 PTAM scale problem

Figure 5.6: The six meter high Giant's Chair of Natsworthy (UK). Without metric or semantic information, it is hard to distinguish this chair from a normal chair. Source: `livefortheoutdoors.com`.

A consequence of using monocular SLAM is that the scale of the map compares differently to the metric scale every time a new map is created. This is because a camera is unable to grasp the metric scale of a scene. This is illustrated in Figure 5.6, where it is only because of the surroundings that a human suspects that this may be a larger chair than usual.

An ever continuously changing scale makes many further steps much harder (e.g. comparison with IMU odometry). To cope with this problem, we use the height sensor of the quadcopter. If we assume that the quadcopter flies horizontal at the time of map initialization, we can match the z-coordinate to the height measurement. Since the height measurement is quite accurate, this procedure captures a good scale estimate.

5.6 PTAM with IMU odometry

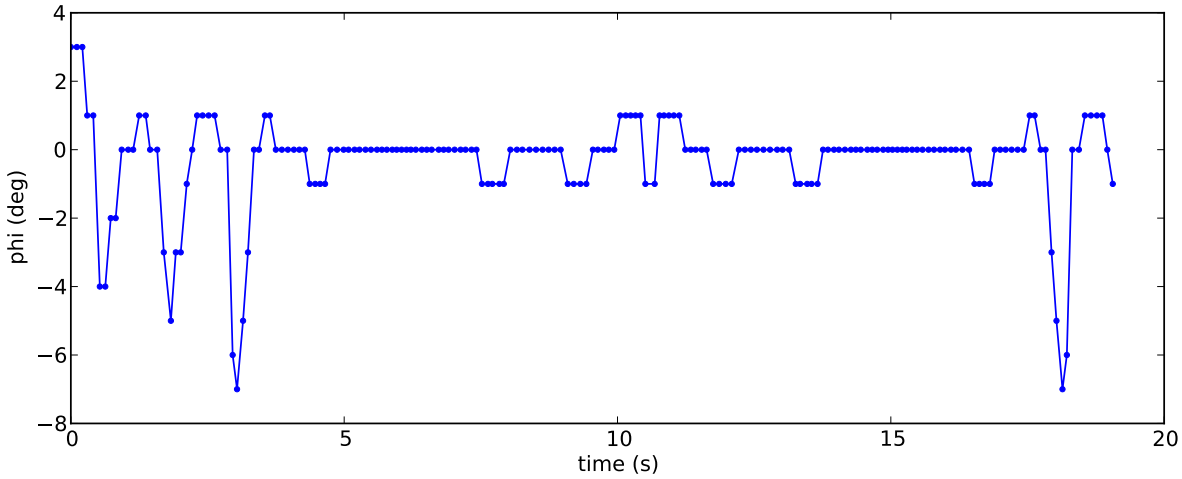


Figure 5.7: Typical sequence of ϕ as measured by the quadcopter IMU. ϕ is one of the three Euler angles.

The PTAM algorithm needs a pose estimate[4] around which it searches for an optimal displacement. The PTAM software package comes with a basic motion predictor based on constant velocity. We tried to improve this by replacing the predictor by the IMU displacement measurements. Previous work [29, 38] has proven that this is a viable improvement, also for PTAM.

In order to match coordinate systems, we made sure the PTAM xy-plane coincides with the floor (the next chapter explains how). Then, we converted the IMU displacements into PTAM displacements. Furthermore, we converted differences in angle measurements to quaternions[5], since these are used by PTAMM. The combination of these variables resulted in a motion estimation.

The experiments done according to this approach were not a success. After initialization, the algorithm instantly loses its focus. The most likely reason for this is the one-degree angle precision of the Euler angles. Since frames are streamed at 20 Hz, most angle differences are thus 0, and sporadically ± 1 (Figure 5.7). This discontinuous behaviour leads to poor predictions. Other reasons may be that the SLAM scale is not exactly equal to the metric scale. Also, the xy-plane of the helicopter is not always parallel to the floor, although this could be remedied by an extra transformation.

Because of the bad results, this approach was discontinued.

5.7 PTAM with second camera

Robustness is very important for a SLAM algorithm. In chapter 7, it will be clear that losing focus while flying is a big problem for autonomous exploration. In order to improve robustness, we tried adding an extra camera. Traditionally, two cameras are combined to form a

[4] With a *pose*, we mean a 3D position together with a 3D orientation (e.g. yaw, axis and angle).
[5] Quaternions are explained in appendix B

stereo pair, i.e. both cameras pointing in the same direction. Instead of this approach, we point the cameras in different directions, so they can look at complementary parts of the environment. The intention of this setup is that when one tracker loses focus, the other one can still provide a location.

Solà et al. [39] and Kaess and Dellaert [40,41] have successfully combined multiple cameras in a single map using custom SLAM frameworks. To our knowledge, multiple cameras have never been used together with PTAM.

Using multiple cameras uncovers a range of new cooperation possibilities. In the ideal case, all cameras detect points in a shared map and find the most consistent quadcopter displacement together. We will show that an important aspect of this solution, namely the shared map, does not work in PTAM. The alternative is using separate maps for each camera, which will be discussed in section 5.7.2.

5.7.1 Single map, multiple trackers

Using a single map which is augmented by multiple cameras is the ideal situation. Computational power is saved because only one mapper has to be run and different cameras can benefit from each others landmarks. As mentioned, this approach does not work in PTAM. This was shown experimentally, using the following setup:

- Camera 1: regular SLAM: tracker and own map
- Camera 2: passive tracker: tracker using camera 1's map without augmenting it

It turned out that the passive tracker is unable to use the map of camera 1, even when using cameras of the same type. We verified the correctness of our implementation by replaying a delayed version of camera 1 as camera 2, which resulted in good tracking performance for both cameras.

The most likely reason for this problem is that PTAM works on a low level (pixel level). The algorithm seems to be extremely sensitive to changes on this low level (even cameras of the same type are never exactly the same). Moreover, difference in brightness also seems to be a problem, because our experiments have shown that reusing a saved map on another day with the same camera does not always work.

5.7.2 Multiple maps, multiple trackers

The architecture used for multiple maps is illustrated in Figure 5.8 for 2 cameras. The bottom line is that every camera has its own SLAM algorithm, without shared information. These might even run on different computers. The trackers then produce their own poses, referred to different coordinate systems. Before we are able to combine the information, we need to transform all poses to one common reference frame. It will later be clear that the calibration of this transform has to be done after every initialization. When the transform calibration is complete, the poses can be combined, using a weighted sum based on the confidences of the trackers:

$$\text{pose} = \sum_i \frac{\text{confidence}_i}{\sum_j \text{confidence}_j} \cdot \text{pose}_i \tag{5.2}$$

The theoretical challenge of this system lies in the transform calibration, which will be explained in further detail.

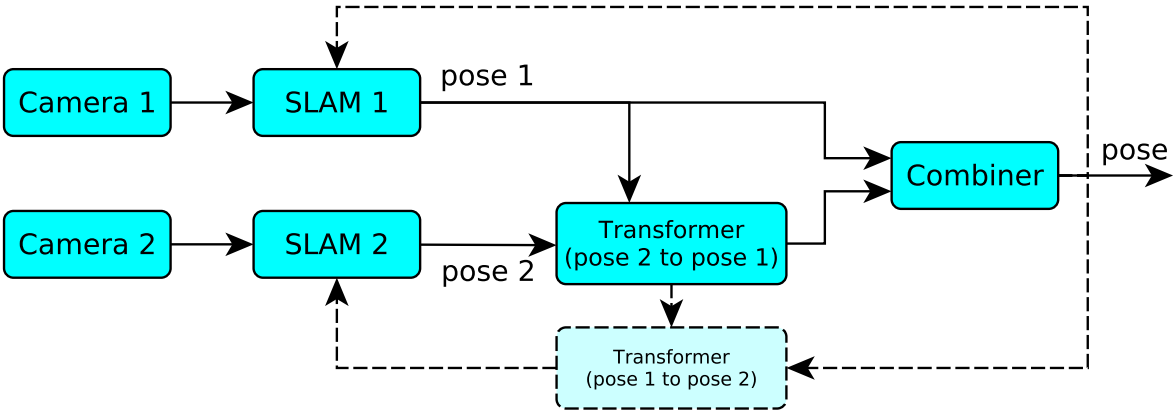


Figure 5.8: 2 camera architecture with multiple maps. The dashed lines are not yet implemented and it's an open research question whether or not the feedback loops improve performance.

Transform calibration

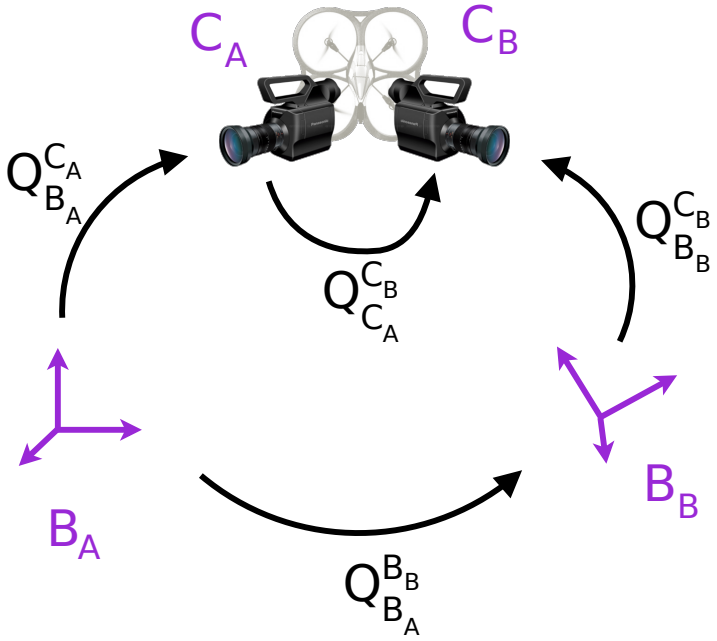


Figure 5.9: Schematic diagram of the multiple maps transformation problem. Camera C_A has base B_A and camera C_B has base B_B. All Q_y^x are 4×4 matrices which represent a translation, a rotation and a scale change.

To understand this problem, consider Figure 5.9. Each map has an own basis and the camera pose is represented in this basis. A camera pose consists of a translation $\boldsymbol{T}_B^C$ and a rotation R_B^C and can be represented by a matrix

$$\mathrm{Q}_B^C = \left[\begin{array}{c|c} \mathrm{R}_B^C & \boldsymbol{T}_B^C \\ \hline \mathbf{0} & 1 \end{array}\right] \tag{5.3}$$

Note that this is in fact a change of basis from the fixed base B to a base C that is attached to the camera. In the following paragraphs, we will keep following the convention that Q_X^Y

transforms a vector from base Y to base X. If we define for point P and base X that $P|_X$ represents the coordinates of P in base X, we can write our change of basis as

$$\left[\frac{P|_B}{1}\right] = \mathsf{Q}_B^C \cdot \left[\frac{P|_C}{1}\right] \quad ,\forall P \tag{5.4}$$

which we simplify as

$$P|_B = \mathsf{Q}_B^C \cdot P|_C \quad ,\forall P \tag{5.5}$$

In a similar manner as Q_B^C, we define the (unknown) change of basis from B_A to B_B as

$$\mathsf{Q}_{B_A}^{B_B} = \left[\begin{array}{c|c} \mathsf{R}_{B_A}^{B_B} & \boldsymbol{T}_{B_A}^{B_B} \\ \hline \mathbf{0} & s \end{array}\right] \tag{5.6}$$

Note the scale factor s, which is present because scale in monocular SLAM is never exactly equal (and B_A and B_B originate from two different PTAM maps). This transform is used in the typical way:

$$\mathsf{Q}_{B_A}^{B_B} \cdot \left[\frac{P|_{B_B}}{1}\right] = \left[\frac{s\ P|_{B_A}}{s}\right] \equiv \left[\frac{P|_{B_A}}{1}\right] \quad ,\forall P \tag{5.7}$$

The transform $\mathsf{Q}_{C_A}^{C_B}$ is defined in exactly the same way, between the two bases fixed on the cameras (and thus on the quadcopter).

Equipped with these definitions, we can reformulate the problem. We want to find $\mathsf{Q}_{B_B}^{C_B}$ when we are given $\mathsf{Q}_{B_A}^{C_A}$. For this, we can use a number of training samples $(\mathsf{Q}_{B_A}^{C_A}, \mathsf{Q}_{B_B}^{C_B})$.

The solution lies in the two changes of basis $\mathsf{Q}_{B_A}^{B_B}$ and $\mathsf{Q}_{C_A}^{C_B}$. During one PTAM run, these transformations remain fixed. If we assume we already know these matrices, the problem can be solved:

$$\begin{aligned}
& P|_{B_A} &&= \mathsf{Q}_{B_A}^{C_A} \cdot P|_{C_A} && (5.8)\\
\Rightarrow\quad & P|_{B_A} &&= \mathsf{Q}_{B_A}^{C_A} \cdot \mathsf{Q}_{C_A}^{C_B} \cdot P|_{C_B} && (5.9)\\
\Rightarrow\quad & \mathsf{Q}_{B_A}^{B_B} \cdot P|_{B_B} &&= \mathsf{Q}_{B_A}^{C_A} \cdot \mathsf{Q}_{C_A}^{C_B} \cdot P|_{C_B} && (5.10)\\
\Rightarrow\quad & P|_{B_B} &&= {\mathsf{Q}_{B_A}^{B_B}}^{-1} \cdot \mathsf{Q}_{B_A}^{C_A} \cdot \mathsf{Q}_{C_A}^{C_B} \cdot P|_{C_B} && (5.11)\\
\Rightarrow\quad & \hat{\mathsf{Q}}_{B_B}^{C_B} &&= {\mathsf{Q}_{B_A}^{B_B}}^{-1} \cdot \mathsf{Q}_{B_A}^{C_A} \cdot \mathsf{Q}_{C_A}^{C_B} && (5.12)
\end{aligned}$$

Where $\hat{\mathsf{Q}}_{B_B}^{C_B}$ is the best estimation of $\mathsf{Q}_{B_B}^{C_B}$. We do not know $\mathsf{Q}_{B_A}^{B_B}$ and $\mathsf{Q}_{C_A}^{C_B}$, but we can try to find these transformations so that $\mathsf{Q}_{B_B}^{C_B} - \hat{\mathsf{Q}}_{B_B}^{C_B}$ becomes as small as possible on the train set.

Appendix C.1 proves that

$$\hat{\mathsf{Q}}_{B_B}^{C_B} = \left[\begin{array}{c|c} {\mathsf{R}_{B_A}^{B_B}}^{-1} \cdot \mathsf{R}_{B_A}^{C_A} \cdot \mathsf{R}_{C_A}^{C_B} & \boldsymbol{T}''' \\ \hline \mathbf{0} & 1 \end{array}\right] \tag{5.13}$$

with $\boldsymbol{T}'''$ a function of rotations and translations. We see that minimizing $\mathsf{Q}_{B_B}^{C_B} - \hat{\mathsf{Q}}_{B_B}^{C_B}$ also means minimizing $\mathsf{R}_{B_B}^{C_B} - {\mathsf{R}_{B_A}^{B_B}}^{-1} \cdot \mathsf{R}_{B_A}^{C_A} \cdot \mathsf{R}_{C_A}^{C_B}$. This means that we first can calculate the ideal set of rotations $(\mathsf{R}_{B_A}^{B_B}, \mathsf{R}_{C_A}^{C_B})$ with this error function. Then, we can keep the rotations constant, and start minimizing $\boldsymbol{T}_{B_B}^{C_B} - \boldsymbol{T}'''$ with the remaining parameters: $(\boldsymbol{T}_{B_A}^{B_B}, \boldsymbol{T}_{C_A}^{C_B}, s)$.

We use *Covariance Matrix Adaptation Evolution Strategy* (CMA-ES) as minimization algorithm [42, 43]. This evolutionary optimizer can handle generic error functions and has proven itself to be a powerful algorithm [44–47]. Important to know is that CMA-ES works with a selectable number of continuous parameters. The CMA-ES user has to transform these parameters into the desired data structures (in our case, two rotation matrices / two translations and a scale factor). The user evaluates the fitness of the data structures and returns the error. With this info, CMA-ES tries to tune the parameter array so that the error becomes minimal.

Optimal rotations

Let's first consider the issue of finding the optimal rotations. The most important question that arises is how to transform a set of parameters into a rotation. As a rotation in 3D space has three degrees of freedom, one might propose three CMA-ES parameters to become the Euler angles of the rotation. This turned out to work, but not very well. This might be because the error function is periodic in every parameter (with a period of 2π). In order to improve performance, we switched to quaternions (appendix B). First, we took three CMA-ES parameters (a, b and c) and transformed them into a quaternion like:

$$\theta = \pi \tanh(c) \tag{5.14}$$

$$u_x = \tanh(a) \tag{5.15}$$

$$u_y = \tanh(b) \tag{5.16}$$

$$u_z = \begin{cases} \sqrt{1-(u_x^2+u_y^2)} & , u_x^2+u_y^2 < 1 \\ 0 & , \text{otherwise} \end{cases} \tag{5.17}$$

$$quat = \left[\cos\frac{\theta}{2},\ \frac{u_x}{|\boldsymbol{u}|}\sin\frac{\theta}{2},\ \frac{u_y}{|\boldsymbol{u}|}\sin\frac{\theta}{2},\ \frac{u_z}{|\boldsymbol{u}|}\sin\frac{\theta}{2}\right]. \tag{5.18}$$

These parameters choices didn't work at all, the resulting rotations performed nearly as bad as random ones. The best solution was found by adding an extra CMA-ES parameter and apply a one-on-one mapping between the quaternion parameters (q_0, q_1, q_2, q_3) and the CMA-ES parameters (a, b, c, d). Afterwards, the quaternion is normalized before it is used. This approach works very good, which also can be seen in Table 5.2. An additional advantage of quaternions is that concatenation of rotations can be calculated efficiently (section B.2).

Parameter mapping	CMA-ES error function	Calc. time (s)	Avg. test error
Euler	$\sum \lvert\hat{q}_{B_B}^{C_B} - q_{B_B}^{C_B}\rvert^2$	15 - 25	0.05 - 0.06
quaternion (1on1)	$\sum \lvert\hat{q}_{B_B}^{C_B} - q_{B_B}^{C_B}\rvert^2$	7 - 8	0.0251
quaternion (1on1)	$\sum \lvert\hat{\mathsf{R}}_{B_B}^{C_B} - \mathsf{R}_{B_B}^{C_B}\rvert^2$	31 - 44	0.0251 - 1.99
quaternion (1on1)	$\text{random}(\lvert\hat{q}_{B_B}^{C_B} - q_{B_B}^{C_B}\rvert^2)$	18 - 19	0.04 - 0.07

Table 5.2: Experimental results of rotation optimization. 144 pairs were used for training and the testset consisted of 20 pairs. In the CMA-ES error, $|.|^2$ denotes the sum of all squared elements. $\sum$ means that the errors of *all* samples are summed. The average test error is calculated as $\text{avg}(|\hat{q}_{B_B}^{C_B} - q_{B_B}^{C_B}|)$.

Another important choice for rotation optimization is the error function. When we want to

compare the rotations $\mathsf{R}_{B_B}^{C_B}$ and $\hat{\mathsf{R}}_{B_B}^{C_B}$, two logical choices would be $|\hat{\mathsf{R}}_{B_B}^{C_B} - \mathsf{R}_{B_B}^{C_B}|^2$ or $|\hat{q}_{B_B}^{C_B} - q_{B_B}^{C_B}|^2$ with R a rotation matrix, q a quaternion and $|.|^2$ the sum of all squared elements. Experiments showed that the latter is the best option (Table 5.2), and it is also the fastest option, since no computationally expensive conversion to matrices is needed.

A final choice that has to be made is to let the error function take all train samples into account, or just one (randomly chosen). In the latter case, the error function can be different for the same parameters. As shown in Table 5.2, working with a random sample was slower and less accurate.

We conclude that the best way of finding the optimal rotations, is to use a 1on1 quaternion mapping with $\sum |\hat{q}_{B_B}^{C_B} - q_{B_B}^{C_B}|^2$ as error function.

Optimal translations and scale factor
This part is easier, because most choices are obvious. We receive 7 CMA-ES parameters this time (instead of 6). The two translations are mapped one-on-one with the first six parameters. The last parameter p is mapped to the scale factor s like

$$s = \begin{cases} \exp(p) & , p \leqslant 0 \\ 1 + p & , p > 0 \end{cases} \tag{5.19}$$

because s should always be a positive number. As error function, we use $\sum |\boldsymbol{T}_{B_B}^{C_B} - \boldsymbol{T}'''|^2$. On the same dataset as previously used, CMA-ES always converges to the same solution (test error: $\text{avg}(|\boldsymbol{T}_{B_B}^{C_B} - \boldsymbol{T}'''|)$ = 0.819 mm) in 36 to 40 seconds.

Feedback to PTAM
With the calibration and the combiner discussed earlier, we have put together a dual camera PTAM system. The final pose could be put back into the PTAM trackers, so they can get a better pose estimate, especially when they lost focus. There is a risk that the correspondences are too noisy, and that it will not work. We have not tested this case, so it remains an open research question.

Opportunities
Because the solution to the calibration problem is quite general, it opens up some interesting applications and extensions. Firstly, we have now shown how to get the transformations between two cameras. If we want to use this with more than 2 cameras, a reference camera has to be chosen for it to work, while relationships between the other cameras are neglected. This can be extended by applying a complete graph and optimizing all transformations simultaneously.

In terms of applications, the calibration technique could be used to calibrate the position of cameras in mobile robots or camera rigs. By applying a fading memory, the calibration could update itself live, which could be handy for wearable cameras.

SLAM data processing | 6

This chapter shortly discusses the processing of PTAM data. This makes the data more valuable for further steps in the architecture. We start by showing how we combine the SLAM position with the IMU data. Next, we will explain how we try to divide the SLAM points into floor and wall points.

6.1 Position processing: Hybrid data

In this section, we want to find a good estimate of the robot position, based on the IMU from the quadcopter and the output of PTAM. We call the resulting estimate *hybrid*, because it originates from two sources.

An important consideration is that the IMU data is relative and can encounter drift, while the SLAM data is absolute. On the other hand, PTAM sometimes fails to provide a value, while the IMU is quasi 100 % reliable to provide an acceptable value. It is thus desirable that when PTAM confidence is low, the IMU takes over. To avoid drift, the position should move to the absolute SLAM position as soon as the confidence is high again.

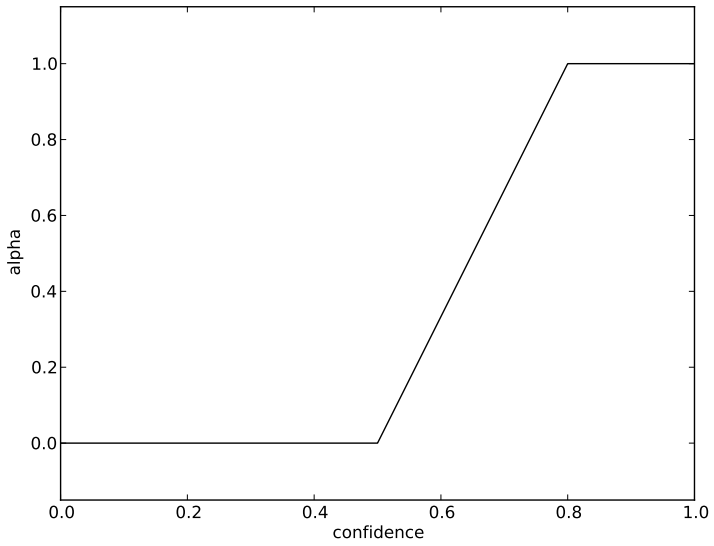


Figure 6.1: Conversion function from the confidence to α.

Following this idea, we want to calculate hybrid position $p_{h,i}$ based on the previous hybrid position $p_{h,i-1}$, the IMU displacement Δp_{IMU} and the absolute SLAM position p_{SLAM}. We

chose the following combination:

$$\Delta p_{\text{SLAM}} = p_{\text{SLAM}} - p_{h,i-1} \tag{6.1}$$

$$p_{h,i} = p_{h,i-1} + (1-\alpha)\,\Delta p_{\text{IMU}} + \alpha\,\Delta p_{\text{SLAM}} \tag{6.2}$$

where α represents how sure we are of Δp_{SLAM}, which we define as

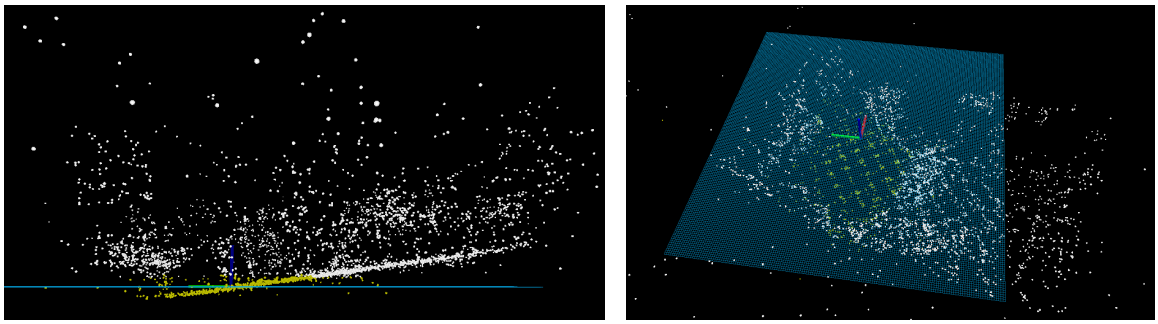
$$\alpha = \begin{cases} 0 & , c < C_{min} \\ \frac{c - C_{min}}{C_{max} - C_{min}} & , C_{min} < c < C_{max} \\ 1 & , C_{max} < c \end{cases} \tag{6.3}$$

with c the confidence and C_{min} and C_{max} thresholds. We chose 0.5 for C_{min} and 0.8 for C_{max}. Figure 6.1 shows this conversion.

Experiments show that this works fine and they confirm the desired behaviour, which we described earlier.

6.2 SLAM landmark processing: Floor point detection

We want to divide the PTAM landmark positions into floor and non-floor points. This classification will be used in the next section to classify the wall points.



(a) (b)

Figure 6.2: Side and top view of untransformed SLAM points. The grid is the SLAM xy-plane, the yellow points are classified as floor and the white points as wall.

For a human, it can be quite a challenge to detect structures in the SLAM points (e.g. Figure 6.2). A major exception to this are the floor points. The floor is typically the flattest surface of a room and often has many features. Based on these observations, our strategy for floor point detection will be classify the points near a clearly present floor plane as floor points.

If we point the camera at the floor while initializing, PTAM will make sure the xy-plane coincides with the floor. This correspondence is however only based on the first two keyframes, and is never updated again later. Consequently, when the map grows further, the floor always turns out to be slightly tilted (Figure 6.2).

To solve this issue, we will try to find the principal plane in our data. To avoid influence of wall and object points, we only look at the points which were previously classified as floor

points. We perform classical *Principal Component Analysis* (PCA) [48] on these points to get the main directions of variation. When we define point n as $\boldsymbol{x}_n$ ($n = 1..N$), this means we have to calculate

$$\mathsf{S} = \frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{x}_n - \overline{\boldsymbol{x}})(\boldsymbol{x}_n - \overline{\boldsymbol{x}})^{\mathrm{T}} \tag{6.4}$$

with

$$\overline{\boldsymbol{x}} = \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{x}_n. \tag{6.5}$$

We then calculate the two eigenvectors $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ corresponding with the two largest eigenvalues of S. Now, the floor plane is defined by point $\overline{\boldsymbol{x}}$ and normal $\boldsymbol{n} = \boldsymbol{u}_1 \times \boldsymbol{u}_2$. The next step is to project the floor to the xy-plane. This leaves a degree of freedom: the rotation around the z-axis. We choose this to be zero, so the points are as close to the old points as possible. This is done as follows:

$$\boldsymbol{u}_z = [0,\ 0,\ 1]^{\mathrm{T}} \tag{6.6}$$

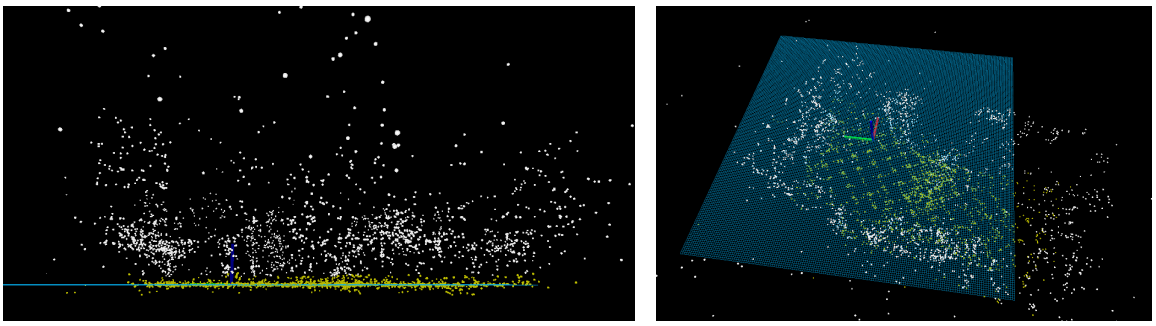
$$\boldsymbol{r} = \operatorname{sign}(n_z)\ \boldsymbol{n} \times \boldsymbol{u}_z \tag{6.7}$$

$$\theta = \arccos(\boldsymbol{n} \cdot \boldsymbol{u}_z) \tag{6.8}$$

$$\text{rotation} = \texttt{axis\_angle}(\text{axis} = \boldsymbol{r}, \text{angle} = \theta) \tag{6.9}$$

where the sign function is needed to assure that $\operatorname{sign}(n_z) \cdot \boldsymbol{n}$ is facing up. $\texttt{axis\_angle}(\boldsymbol{r}, \theta)$ returns a rotation of θ around $\boldsymbol{r}$. After the rotation is performed, the mean is shifted to the origin. The transformed points are then classified as

$$\texttt{floor\_points} = \left\{ p \;\middle|\; |p_z| < 30\,\text{cm} \right\} \tag{6.10}$$



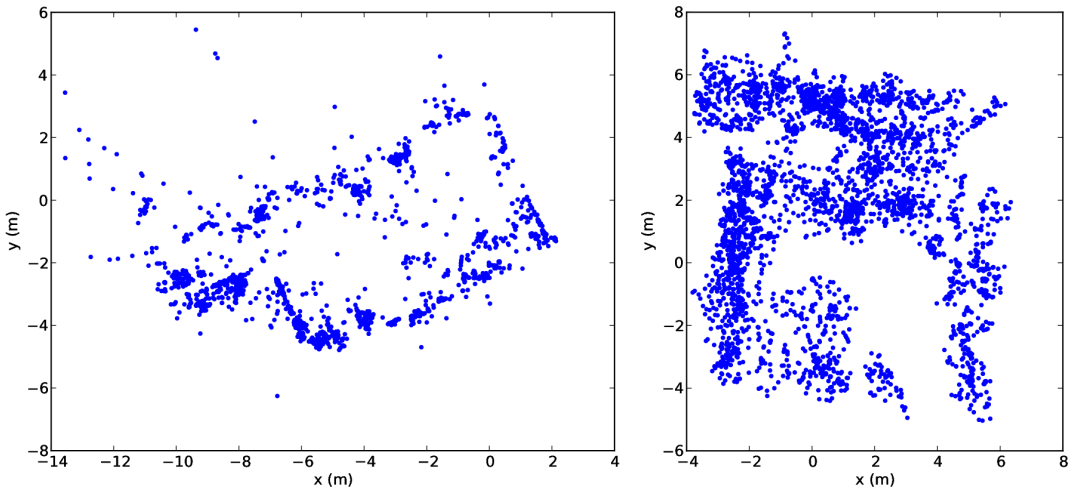
(a) (b)

Figure 6.3: Side and top view of transformed SLAM points. The same conventions apply as in Figure 6.2.

This operation is performed at each points update and converges to a stable solution in one or two runs. The result can be seen in Figure 6.3.

6.3 SLAM landmark processing: Wall point detection

The classifier of the previous section can be used for wall point detection. We want to know which points originate from walls because the autonomous controller and the wall detector depend on these points.



(a) dataset recorded at the reception of the ELIS department (b) dataset recorded in a rectangular room

Figure 6.4: Examples of landmark point clouds without floor points.

A first try could be to classify all non-floor points as wall points. Indeed, many landmarks will be on walls (or near walls) and when inspecting the point cloud of Figure 6.4(a), we clearly see where the walls probably are. However, when a room has furniture in the middle or when the ceiling also gets mapped, one gets a more cluttered point cloud like in Figure 6.4(b).

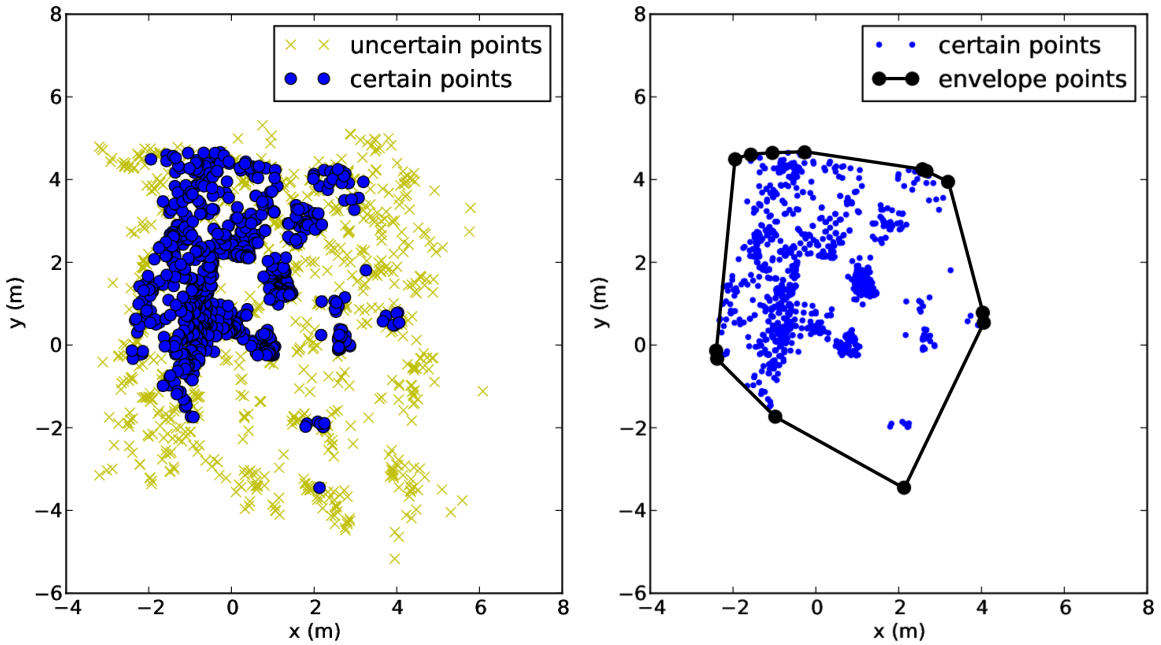
It is clear that only a limited number of rooms can be mapped with this easy classification. Therefore, we try to remedy this issue by using the floor points. These give us information about where the floor is. Obviously, walls will not be surrounded by floor points, and our strategy will use this observation to filter out wall points that lie amidst floor points.

First, we try to find the floor points of which we are sure that they are genuine floor points. We also try to make sure that these are not too near the walls to avoid filtering actual wall points. We define a certain floor point as a floor point when it has at least 20 neighbours in a radius of 0.5 m. This approach favors points on the inside amidst many other floor points. An example application of this definition is shown in Figure 6.5(a).

We now classify wall points as *all non-floor points of which the cluster does not lie completely within the envelope of the certain floor points*. An envelope is the largest convex polygon that can be constructed with a set of points as corners that encompasses all points (see Figure 6.5(b)). A point belongs to a cluster if the distance between this point and the nearest point in the cluster is smaller than 15 cm. We use clusters to avoid wall corners in concave rooms to be chopped off. This filtering procedure is illustrated in Figure 6.6.

Note that this filtering is not perfect. It will have difficulties with clearly concave rooms like L-shaped rooms. To be able to cope with such situations, we probably will need a different approach. However, we showed in this section that making use of the floor points can significantly improve the wall point classification.

The algorithms of the last two sections enable us to quite robustly detect floor points in a point cloud. The remaining points are filtered, using the information that the floor points



(a) (b)

Figure 6.5: (a) Typical result of the determination procedure for finding the floor points of which we are certain. The same dataset was used as in Figure 6.4(b). Note that these are not the same points as Figure 6.4(b) plots the non-floor points. (b) envelope of certain points.

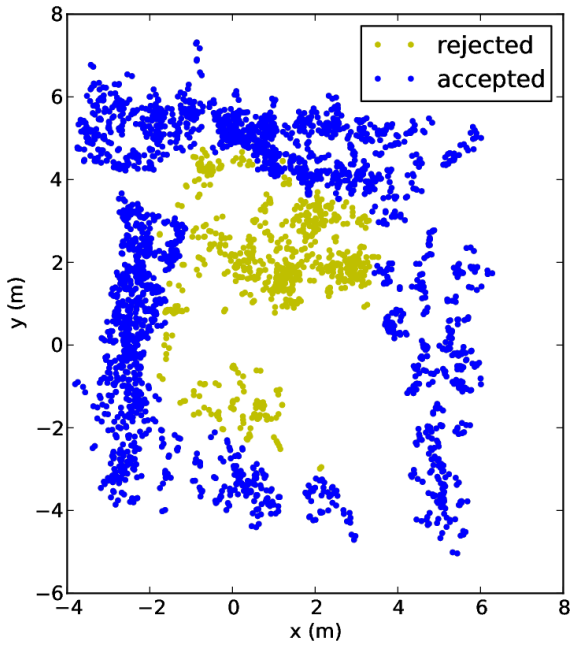


Figure 6.6: Typical wall point classification. The input is the set of non-floor points, the accepted points are classified as wall points.

give us. The resulting wall points typically have the shape of the room. However, it would be false to assume that these all originate from actual walls. Often, these points originate instead from e.g. shelves, closets or computers. With the limited information available and the relatively large uncertainty on their positions, it is infeasible to detect which points are

from walls and which are from other objects. Hence, the algorithms using these points will have to keep in mind a high level of distortion.

Autonomous controller | 7

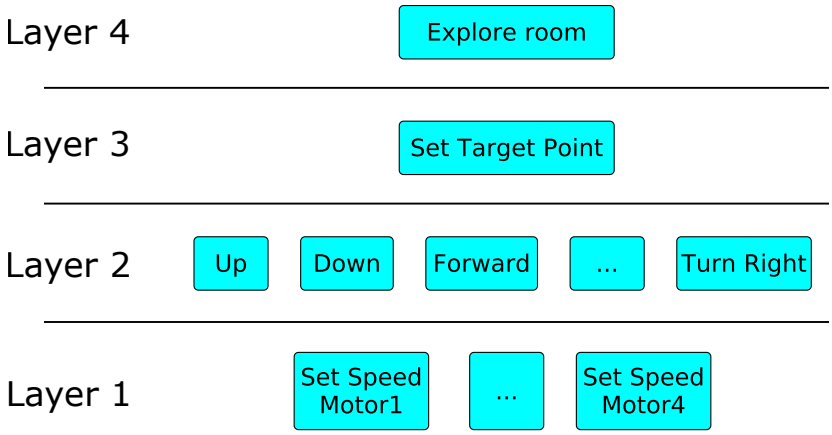


Figure 7.1: Command hierarchy for quadcopter control.

This chapter discusses how the quadcopter can be controlled so that a room is explored. Since we use the AR.Drone, we can send easy-to-understand commands like *fly left* and it will perform them with acceptable accuracy. As shown in Figure 7.1, these layer 2 commands hide the complexity of the underlying layer 1. To simplify the exploring task, we introduced another abstraction layer (layer 3). This layer has just one command: *Fly to target point*, which allows the client to define a pose in the xy-plane to which the quadcopter will fly. Section 7.1 explains our implementation.

Section 7.2 discusses the top layer. This part has been solved theoretically, but it has only partially been implemented. The reason for this is that, while testing the controller, PTAM always lost its focus at some point. The problem is that quadcopter movements are quite abrupt and it might happen that suddenly half of the points disappear. Moreover, it's very hard for an autonomous controller to know when PTAM is on the verge of losing its focus. Thirdly, the PTAM map is only expanded when adding a keyframe. This leads to a stepwise map growth and it's hard to predict or induce an expansion.

These problems could probably be resolved if the PTAM algorithm would be more robust. Therefore, a large part of this work was committed to improve PTAM (as explained in chapter 5). Unfortunately, the improvements were not sufficient and we will only state our theoretical considerations in section 7.2.

7.1 Target point controller

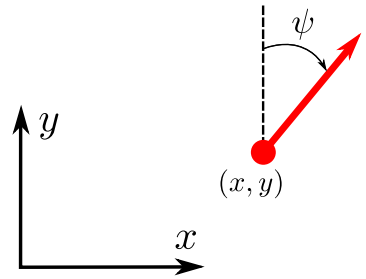


Figure 7.2: Definition of a pose in the xy-plane.

The target point controller is a piece of software that implements layer 2. Its purpose is to fly to a pose in the xy-plane. Note that this pose has three dimensions (x, y, ψ), which are shown in Figure 7.2. The target can be set graphically, using rviz[1] or by any other ROS node.

When a target is set, the controller sends quadcopter commands with a flying speed proportional to the coordinate difference, keeping in mind a maximum speed. In other words, we implemented a P-controller.

We chose the parameters so that the quadcopter moves as slow as possible to its target. However, for a maximum speed which is too low, the drift becomes larger than the actuated speed and the controller stops working.

7.2 Exploring controller

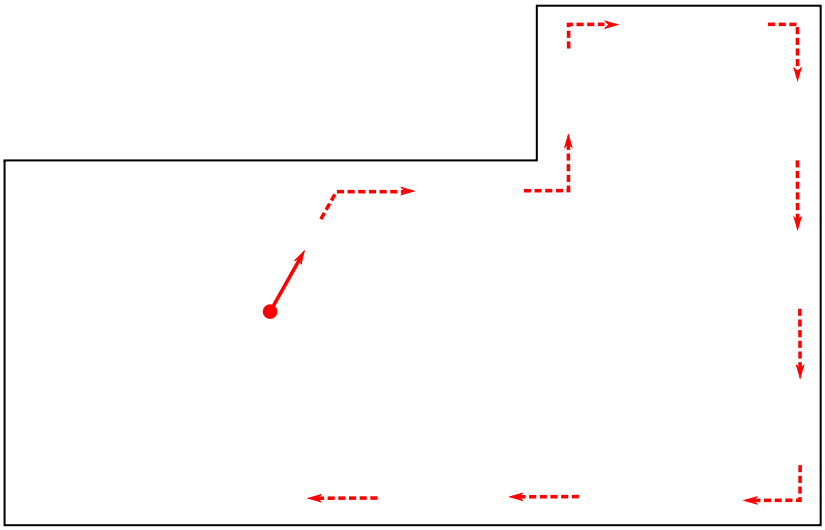


Figure 7.3: Desired exploring strategy.

The goal of the exploring controller is to make sure PTAM maps all relevant parts of a room. A way of accomplishing this is by the strategy shown in Figure 7.3. First, the drone flies

[1] rviz is a visualization toolkit of ROS (see chapter 4).

forward until it reaches a wall. Then it follows that wall until the cycle is complete[2].

The main technical challenge here is to know when there is a wall in front of the quadcopter. The only frontal input device we have, is the camera. We chose PTAM points that are classified as wall to become our indicators. Other video analysis tools like the spatial layout estimators[3] could be a possible alternative.

It can clearly be seen on Figure 7.3 that an exploring vehicle needs to look at the wall and the right at the same time to detect gaps and corners. Therefore, we propose the camera to be pointed oblique so it makes a 45° angle with the wall.

As stated earlier, these are theoretical considerations and it has not yet been proven to work on a real setup.

[2]Detecting a completed cycle is equivalent to detecting a loop closure in SLAM. We thrust that the SLAM algorithm has an adequate implementation for this. In our case, PTAM is able to close loops that are moderately small (like regular sized rooms).

[3]Some interesting spatial layout estimators use for example volumetric reasoning [49], appearance models [50] or depth-ordered grouping [51].

Wall detection | 8

This chapter will discuss the problem of wall detection. This means that we want to fit vertical surfaces to the PTAM wall points[1]. The final result should be a consistent concatenation of walls so the ground plan can clearly be recognized. Since most walls are vertical, we assume that the z-axis has no influence on the wall placement. The problem thus reduces to finding the most likely set of 2D line segments in the 2D projection of the PTAM points[2] (Figure 8.1).

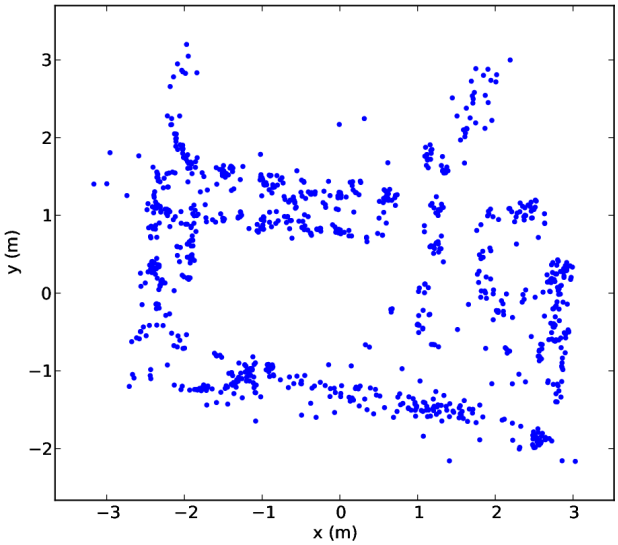


Figure 8.1: Typical input for wall detection problem. A 2D projection of PTAM points generated from a rectangular room. This data was recorded at a PC room in the ELIS department.

We will first discuss previous work in the field before we explain our solution.

8.1 Line fitting

Related research predominantly focuses on line fitting and does not consider finding segment boundaries. Segment boundaries are essential for our application as we want to discover closed room geometries. This is illustrated in Figure 8.2, where the problem of finding the wall edges becomes non-trivial.

[1] Floor/wall point classification was discussed in chapter 6.

[2] Note that, even when the walls are vertical, relevant information is lost when projecting the points. It might for example be possible to filter out points originating from desk surfaces because of their spatial correlation in 3D. We chose to neglect this information because of the noise on the data.

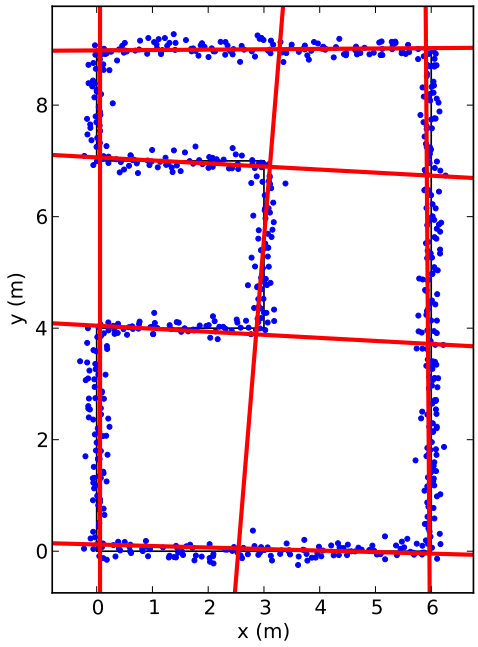
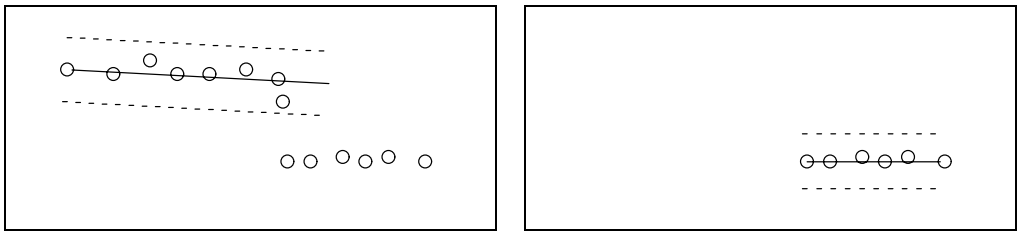


Figure 8.2: Example of room where line boundaries are important. A set of lines is fitted to the wall points, but finding the corresponding segments is a non-trivial problem.

The line fitting problem has been well researched in the case of many outliers, because the standard least squares method fails in these situations. One of the best known algorithms is RANSAC, introduced in 1981 [53] and later extended to fit more than one line [52, 54–56]. A typical extension is to apply RANSAC sequentially, like in Figure 8.3. These approaches have the advantage of simplicity, but more complex methods have been developed to improve performance.

Another well known technique is to perform line detection in the Hough space. The Hough transform (which converts points to Hough space) was proposed in 1962 by P.V.C. Hough and generalized by Duda and Hart [59, 60]. The idea is to let every point vote for a range of cells in parameter space. In the case of line fitting, parameter space has two dimensions and is generally represented by θ and ρ (see Figure 8.4 for definitions). This means that each point adds a sinusoid in parameter space. Figure 8.5 shows a typical transform. If we can localize the bright spots in parameter space, we have found the line parameters.

In 2005, Bandera et al. applied mean shift clustering in the Hough space [61]. The mean



(a) Fit first line and remove inliers

(b) Fit next line on outliers

Figure 8.3: Using the sequential RANSAC approach to find multiple lines [52].

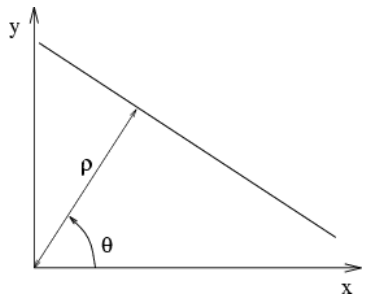
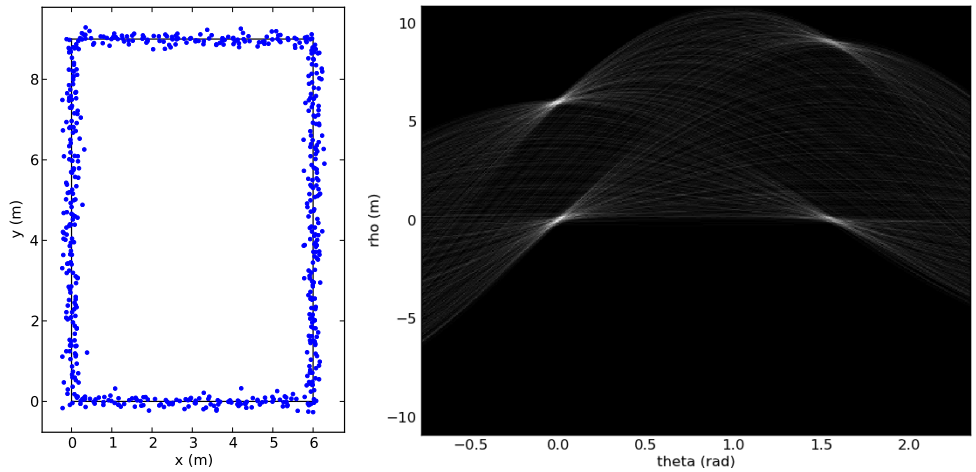
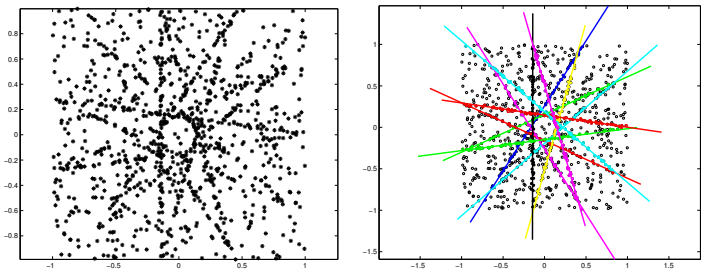


Figure 8.4: Definition of the Hough parameters [57]. The points on the line are defined by $x\cos\theta + y\sin\theta = \rho$



(a) Points in rectangular shape. (b) Hough transform of points with 460×360 grid. Lighter areas have more votes.

Figure 8.5: Illustration of the Hough transform.

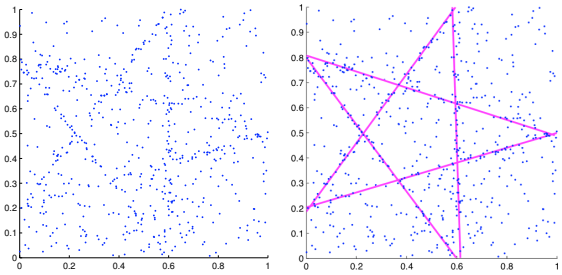


(a) Data (b) Fitted lines

Figure 8.6: Demonstration of J-Linkage [58].

shift algorithm shifts a kernel function according to its mean and is useful for clustering in density functions [62, 63]. This combination of procedures has become a successful line fitting technique.

Furthermore, the J-Linkage method by Toldo and Fusiello has become a popular technique [58]. Their solution is based on random sampling and conceptual data representation. Its performance is illustrated in Figure 8.6.



(a) Data (b) Fitted lines

Figure 8.7: Demonstration of the statistical learning approach [64].

More recent advances suggest that research is far from finished. Chin et al. [64] followed a statistical learning approach for robust fitting and attained impressive results on heavily corrupted data. Their results are demonstrated in Figure 8.7. Similar results apply to AK-SWH, which estimates the scale of inlier errors [65], and PEaRL, following an energy based approach [66]. It should be noted that all models are optimized on a large number of outliers and a relatively small inlier error.

8.2 Expectation Maximization for Gaussian Mixtures

The aforementioned algorithms are optimized for data with many outliers and our results (Section 8.6) will show that they stop working when a lot of (Gaussian white) noise is present in the data. Another approach is thus desired if we want to detect walls autonomously.

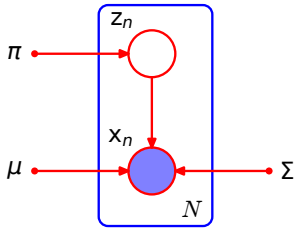
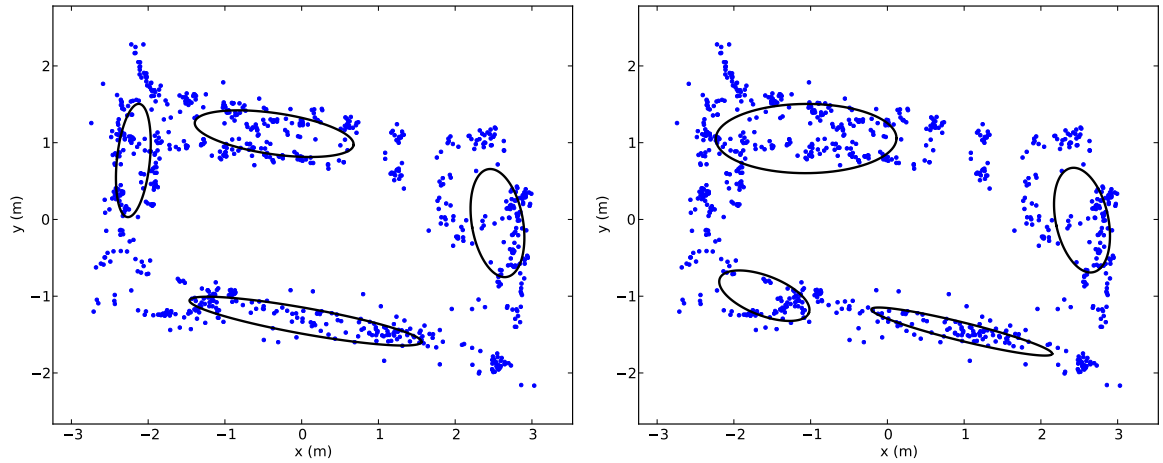


Figure 8.8: Graphical model of a GMM for a set of N data points $\{\boldsymbol{x}_n\}$, with corresponding latent variables $\{z_n\}$. π_k defines $p(z_{nk} = 1)$ and $(\boldsymbol{\mu}_k, \Sigma_k)$ define $p(\boldsymbol{x}_n | z_{nk} = 1)$ independently of n [67].

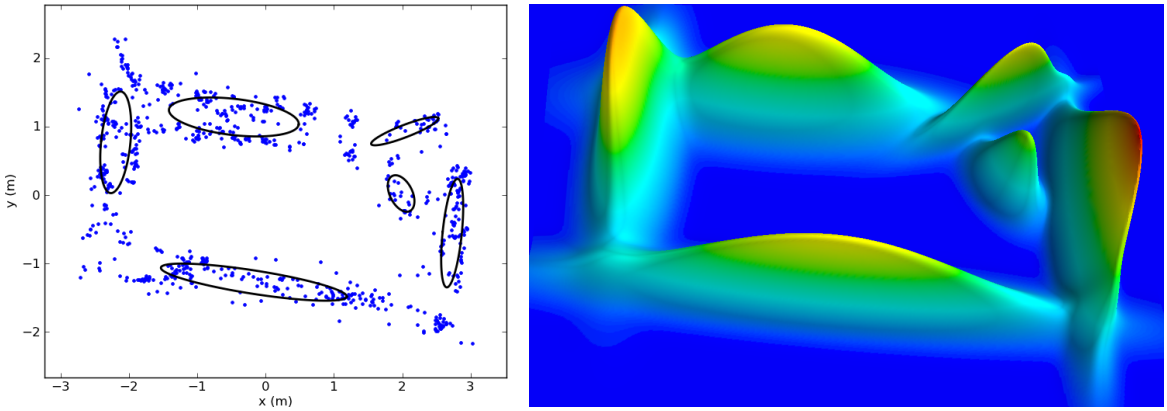
We found that good results are obtained when a standard Gaussian Mixture Model (GMM) optimized with the Expectation Maximization algorithm is applied to this problem. The EM algorithm is a popular technique for finding maximum likelihood solutions for probabilistic models having latent variables [67–69]. The model that we use, is a mixture of Gaussians, which is depicted in Figure 8.8. The essence is that when the latent variable z_{nk} of a point is known to be 1, the coordinates of this point are normally distributed around mean $\boldsymbol{\mu}_k$ with covariance Σ_k.

Before we can try this on a real-world example (with four real walls), we have to provide the number of walls K. When we set K to four, we get an acceptable solution (Figure 8.9(a)). Unfortunately, this is not always the outcome, and sometimes the algorithm returns a worse



(a) (b)

Figure 8.9: EM for Gaussian Mixtures applied to the PC-room test case. Typical outcomes when $K = 4$. The ellipses symbolize the Gaussians.



(a) 2D representation. The ellipses symbolize the Gaussians.

(b) 3D representation of $p(\boldsymbol{x})$ for the solution in (a) (with $\boldsymbol{x} = (x, y)$)

Figure 8.10: EM for Gaussian Mixtures applied to the PC-room test case. Typical outcome when $K = 6$.

solution like in Figure 8.9(b). When we investigate this closer, it turns out that the initial conditions are the most important success factors. To cope with this varying correctness, we will need a measure to define when the outcome is good.

Another problem is that it is hard to determine the appropriate value for K. If we would only look at the probability of the solution, given K, the probability will always increase with K because of extra degrees of freedom. For example, The solution with six walls in Figure 8.10(a) has a higher probability than the solution in Figure 8.9(a). Possible solutions are to try different K-values and compare them or use non-parametric methods such as Dirichlet Process clustering. We will apply the former solution as it can be implemented efficiently, but we will need a more precise estimation of the likelihood.

Further drawbacks are that Gaussians are poor models for wall-point samples since they have no intrinsic notion of edges, and points near the center of a wall have a significantly

higher probability. This becomes evident when looking at the model for the bottom wall in Figure 8.10(b). One would expect a long line with maximum probability and a steep probability descent at the edges.

8.3 Expectation Maximization for Wall Mixtures

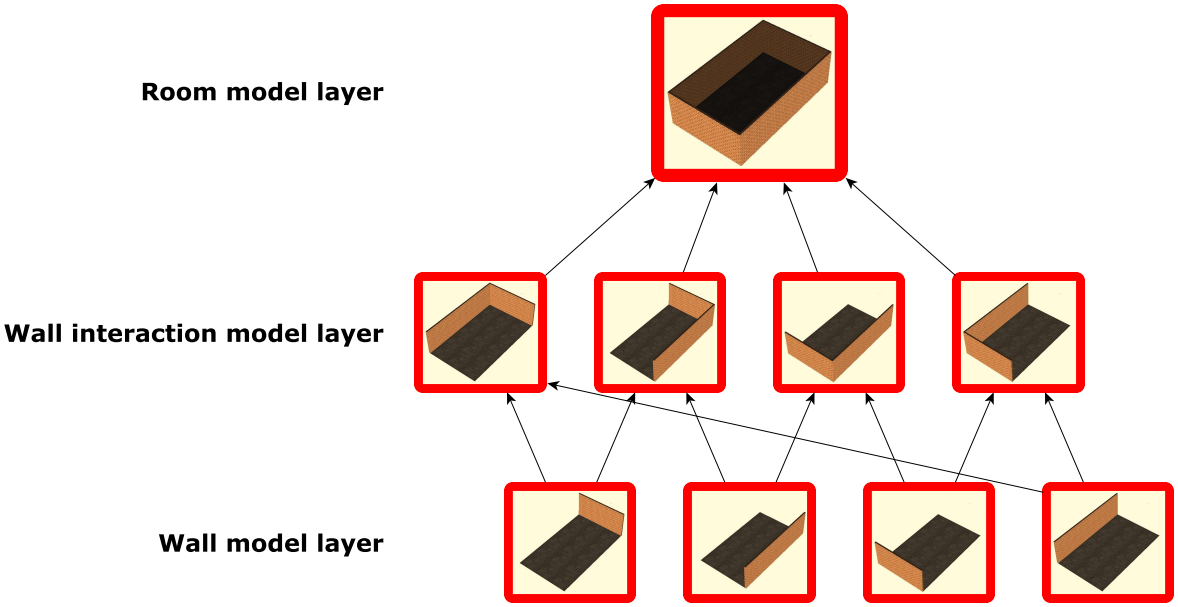


Figure 8.11: Overview of the layered model architecture. Arrows denote propagation of probability. The probabilities associated with one layer are used to calculate the probability of the layer above.

We adapted EM for Gaussian Mixtures in order to solve the problems of section 8.2. This was done by creating a more complex and realistic model for wall-samples. However, looking at every wall individually is not enough. In real life, placement of one wall has a large impact on the placement of other walls. Therefore, we developed a model that incorporates these considerations. Lastly, we added a third model for an entire room, which takes all walls into account. These three models form a layered model architecture which is depicted in Figure 8.11.

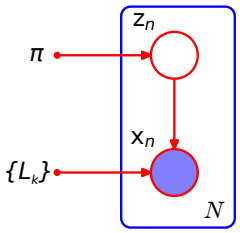


Figure 8.12: Graphical model of a wall mixture model. The same definitions for π, z_n and $\boldsymbol{x}_n$ apply as in Figure 8.8. L_k represents all parameters of wall k.

Figure 8.12 shows the updated graphical model of our wall mixtures, in which the wall

parameters are bundled into $\{L_k\}$. We will now explain each layer and how they are used to improve wall and room detection. We call our approach *EM for Wall Mixtures*.

8.3.1 Wall-sample model

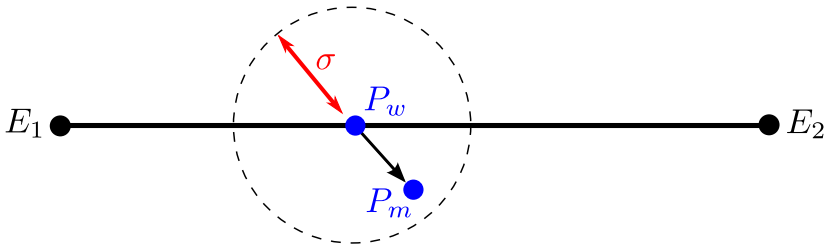


Figure 8.13: Model for sampling from a wall. P_w is a randomly picked point on the segment. P_m is sampled from a Gaussian around P_w with standard deviation σ.

Our wall-sample model is illustrated in Figure 8.13. We model a wall as a line segment with two edge points E_1 and E_2. When we detect a point P_m from this wall, we first define the originating point P_w on the wall. P_w symbolizes the actual landmark on the wall. To model the measurement errors, we add Gaussian white noise with standard deviation σ to P_w, which gives us the measurement P_m. This can be expressed as

$$p(P_m|P_w) = \mathcal{N}(P_m|P_w, \sigma^2 \mathsf{I}_2) \tag{8.1}$$

with I_2 the 2×2 identity matrix and $\mathcal{N}(P_m|...)$ the bivariate normal distribution. We assume that the landmarks P_w are distributed uniformly over the wall surface. The probability density function (pdf) for a data point $\mathbf{x} = (x, y)$ to lie on wall k is $Q(\mathbf{x}|L_k) := p(\mathbf{x}|z_k = 1, L_k)$, with L_k the wall parameters and $Q(\mathbf{x}|L_k)$ defined in the next section.

8.3.2 EM adaptation for wall mixtures

Since we replaced the Gaussian in the GMM by this wall-sample model, we should also adapt the EM algorithm. The EM algorithm is summarized below [67] (applying the same definitions as in Figures 8.8 and 8.12):

1. Initialize the walls $\{L_k\}$ randomly and set $\pi_1 = ... = \pi_K = 1/K$
2. **E-step**: Evaluate the responsibilities $\{\gamma(z_{nk})\}$ using the current parameter values

$$\gamma(z_{nk}) := p(z_{nk} = 1|\boldsymbol{x}_n, \{L_k\}) \tag{8.2}$$

$$= \frac{\pi_k p(\boldsymbol{x}_n|z_{nk} = 1, L_k)}{\sum_{k'} \pi_{k'} p(\boldsymbol{x}_n|z_{nk'} = 1, L_{k'})} \tag{8.3}$$

$$= \frac{\pi_k Q(\boldsymbol{x}_n|L_k)}{\sum_{k'} \pi_{k'} Q(\boldsymbol{x}_n|L_{k'})} \tag{8.4}$$

3. **M-step**: Re-estimate the parameters using the current responsibilities

$$N_k = \sum_n \gamma(z_{nk}) \tag{8.5}$$

$$\pi_k := p(z_k = 1) = \frac{N_k}{N} \tag{8.6}$$

$$\{L_k\} = \arg\max_{\{L_k\}} p(\{L_k\}|\boldsymbol{X}) \tag{8.7}$$

with $\boldsymbol{X} = \{\boldsymbol{x}_n\}$.

4. Return to step 2 unless the parameters have converged or the maximum number of iterations has been reached.

Two important questions remain: what is $Q(\boldsymbol{x}_n|L_k)$ and how do we solve equation (8.7)? First, we will show that the second question can be reduced to the first.

$\{L_k\}$ is maximized for all walls simultaneously. As stated earlier, this coupling is also physically present (closed geometry). However, it gives rise to practical problems for maximization, because these equations are hard to solve. To simplify this problem, we ignore the coupling and maximize $p(L_k|\boldsymbol{X})$ for each wall separately. We thus get (with $\{L_{k'\neq k}\} := \{L_{k'} : k' \neq k\}$)

$$\begin{aligned} L_k &= \arg\max_{L_k} p(L_k|\boldsymbol{X}, \{L_{k'\neq k}\}) && (8.8)\\ &= \arg\max_{L_k} p(L_k)p(\boldsymbol{X}|\{L_{k'}\}) && (8.9)\\ &= \arg\max_{L_k} p(L_k)\prod_n p(\boldsymbol{x}_n|\{L_{k'}\}) && (8.10)\\ &= \arg\max_{L_k} p(L_k)\prod_n \sum_{k'} \pi_{k'} Q(\boldsymbol{x}_n|L_{k'}) && (8.11)\\ &= \arg\max_{L_k} \left[\ln p(L_k) + \sum_n \ln\left(\sum_{k'} \pi_{k'} Q(\boldsymbol{x}_n|L_{k'})\right)\right] && (8.12) \end{aligned}$$

where we can remove $p(L_k)$ from the equation because we have no prior on the parameters of a single wall. A local optimimum of $p(\boldsymbol{X}|L_k)$ is found where all partial derivatives of $p(\boldsymbol{X}|L_k)$ w.r.t. L_k become zero:

$$\begin{aligned} \boldsymbol{0} = \nabla_{L_k} p(\boldsymbol{X}|L_k) &= \sum_n \frac{\pi_k \ \nabla_{L_k} Q(\boldsymbol{x}_n|L_k)}{\sum_{k'} \pi_{k'} \ Q(\boldsymbol{x}_n|L_{k'})} && (8.13)\\ &= \sum_n \gamma(z_{nk}) \ \nabla_{L_k} \ln Q(\boldsymbol{x}_n|L_k) && (8.14) \end{aligned}$$

When we later on get an expression for Q, we can finalize this calculation.

Parameter optimization

We will now define which parameters of L_k we will optimize. One possibility is to choose $\{E_1, E_2, \sigma\}$. A better choice is $\{M, \alpha, e, \sigma\}$ (see Figure 8.14 for parameter definitions), because the values for some of these parameters can be approximated using the parameters of GMM [67]. M is equal to the center of the Gaussian fitted on this cluster and similarly, α is chosen so that the wall is collinear with the main axis of the Gaussian. σ is then set to the variance perpendicular to the wall. This process is illustrated in Figure 8.15. These approximations are computationally less demanding than exact parameter optimization[3], at the cost of some loss of precision.

The remaining parameter e has no clear correspondence with GMM and we need to define the pdf Q to optimize equation (8.14) w.r.t e.

[3] Exact parameter optimization gives rise to a system of coupled non-linear equations because equation (8.14) has to be solved for all elements of L_k simultaneously.

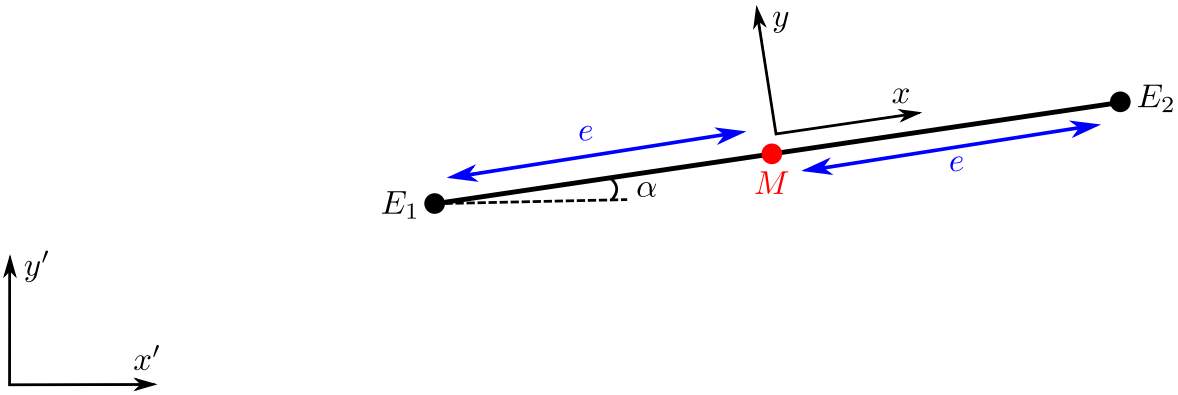


Figure 8.14: Definition of the parameters of a wall. (x', y') defines the general coordinate system and (x, y) defines a coordinate system attached to the wall with M as origin.

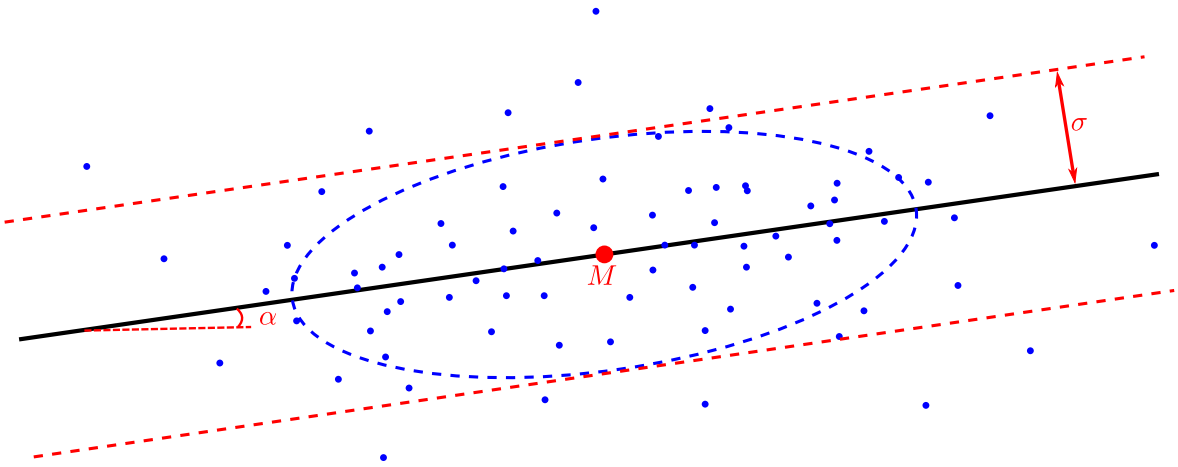


Figure 8.15: Illustration of the wall parameters calculation. The blue dots are landmark measurements. The ellipse is a representation of Σ_k, which could have been derived from the measurements. α and σ are derived from Σ_k.

Ideal pdf

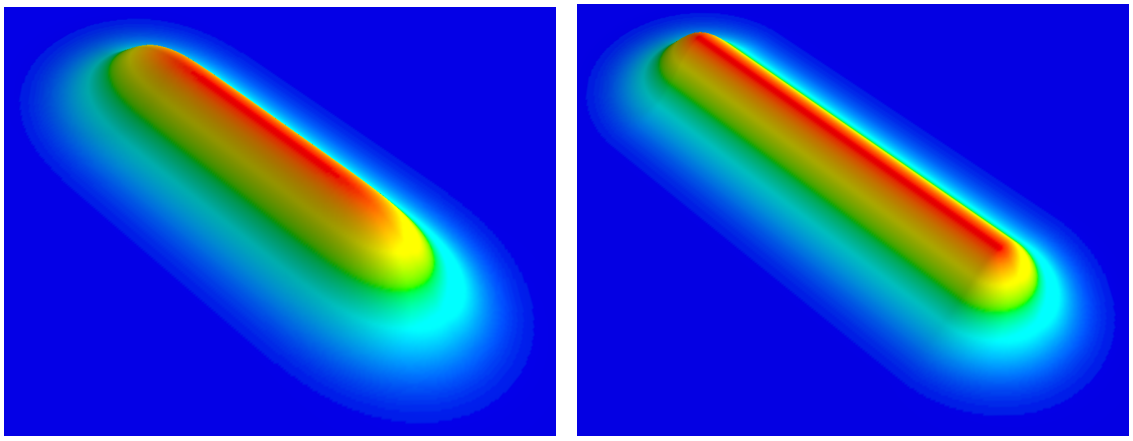
The ideal pdf ($Q(\boldsymbol{x}|L_k)$) can immediately be derived from the wall-sample model of section 8.3.1 without further assumptions or approximations. This comes down to a convolution of a Gaussian along a line segment. The calculations are done in appendix C.2, which concludes with

$$Q(\boldsymbol{x}|L) = \frac{\operatorname{erf}\left(\frac{e+x}{\sqrt{2}\sigma}\right) + \operatorname{erf}\left(\frac{e-x}{\sqrt{2}\sigma}\right)}{4\,e}\,\mathcal{N}(y \mid 0, \sigma^2). \tag{8.15}$$

This function is plotted in Figure 8.16(a).

Except for the improvements of the other model layers, only the need for a computation procedure for the optimal e remains to be able to perform the EM algorithm. For this, we need to solve equation (8.14) with our expression for Q. Appendix C.3 shows that this leads to the following equation:

$$\frac{N_k}{e} = 2\sum_n \gamma(z_{nk})\,\frac{\mathcal{N}(x_n|\text{-}e, \sigma^2) + \mathcal{N}(x_n|e, \sigma^2)}{\operatorname{erf}\left(\frac{e+x_n}{\sqrt{2}\sigma}\right) + \operatorname{erf}\left(\frac{e-x_n}{\sqrt{2}\sigma}\right)} \tag{8.16}$$



(a) the ideal pdf (b) the simplified pdf

Figure 8.16: A 3D plot comparison between the ideal and simplified pdf for $e = 5\sigma$.

which needs to be solved numerically. Alternatively, optimizing equation (8.14) w.r.t e can be done using gradient ascent in a straightforward manner. This has to be done for every line and EM iteration and experiments show that this becomes the bottleneck of our algorithm, even when the number of gradient steps is limited. Therefore, we approximated the ideal pdf with a simplified one. We will show that this enables us to calculate a good value for e in a single iteration.

Simplified pdf

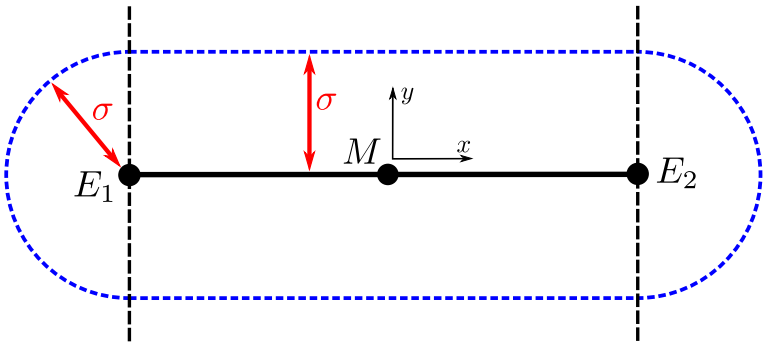


Figure 8.17: The simplified pdf. The three areas are divided by the black dashed line and the pdf is symbolized by the blue dashed line.

The approximated pdf is illustrated in Figure 8.17. Instead of working out a convolution, we assume that the distance to the wall is normally distributed. When deriving a mathematical expression for this, it's easiest to divide 2D space into three areas. In the middle region, $|y|$ is the distance to the line. In the other regions, it's the distance to the nearest point. These considerations lead to the following expression:

$$Q(\boldsymbol{x}|L) = a \begin{cases} \mathcal{N}(y|0,\sigma^2) & ,|x| \leqslant e \\ \mathcal{N}\left(\sqrt{(|x|-e)^2+y^2}\,\middle|\,0,\sigma^2\right) & ,|x| > e \end{cases} \tag{8.17}$$

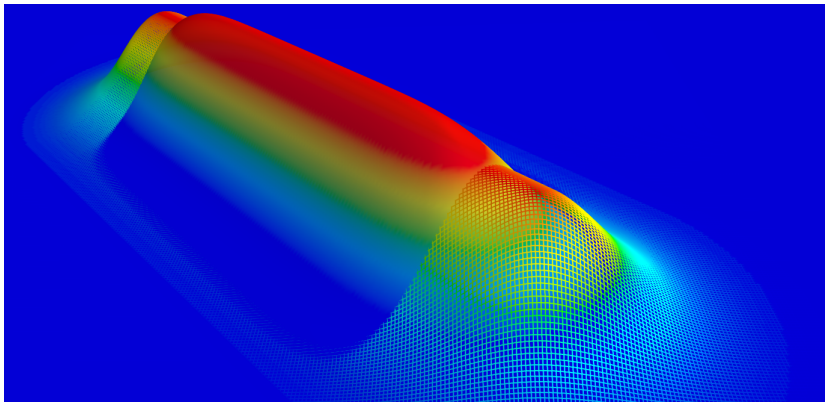
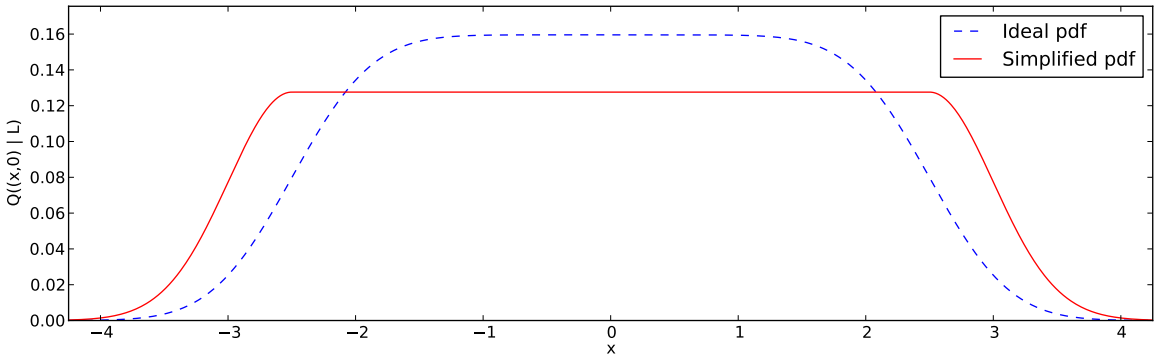
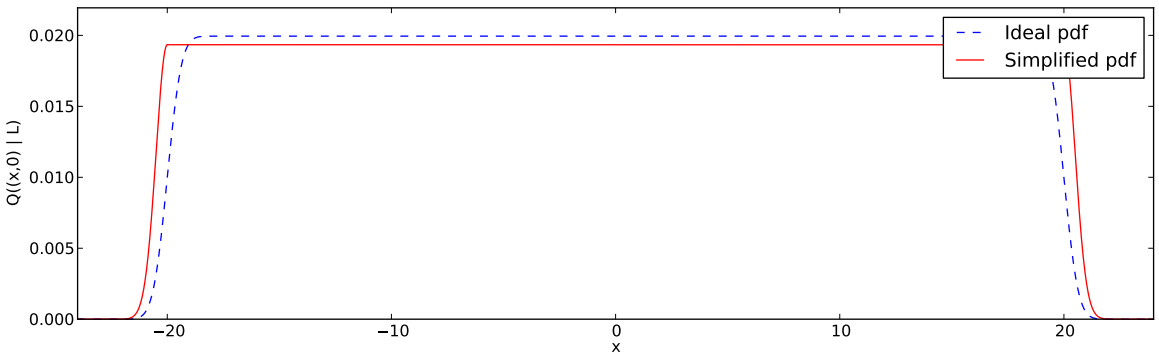


Figure 8.18: 3D plot of the ideal and simplified pdf together for $e = 5\sigma$. The wireframe surface originates from the simplified pdf.



(a) $e = 5\,\sigma$



(b) $e = 40\,\sigma$

Figure 8.19: Intersection plots of the ideal and simplified pdf together for different choices of e. All intersections are made at $y = 0$.

where a is a normalisation factor. Appendix C.4 derives that a is

$$a = \frac{1}{\sigma\sqrt{2\pi} + 2\,e}. \tag{8.18}$$

This pdf is shown in Figure 8.16(b).

Figures 8.16, 8.18 and 8.19 compare the two probability densities. We see that the simplified pdf overestimates the probability at the edges. The difference however becomes smaller as

the wall length increases with respect to σ. In the next section, we will show that this slight deviation has little influence.

Again, we can try to find an expression for the ideal e by inserting the simplified pdf into equation (8.14). Appendix C.5 shows an iterative algorithm that can solve the problem. The procedure selects the points beyond the edges (given the previous e) and uses them to calculate a new value for e. This aspect very much resembles that of *support vector regression*[4]. Furthermore, when we know those points, we can calculate e directly, using a single equation (derived in appendix C.5). In principle, we should still iterate this equation because e influences this set of points. Results show however that we obtain a good estimate already after the first iteration. This approach is significantly faster than gradient ascent on the ideal pdf while experiments show the same performance.

8.3.3 Wall interaction model

We now discuss how the prior information about wall interactions was incorporated into the model. Our prior is twofold:

1. We assume that wall edges are connected, i.e.

$$p(\{A_k\}) = \prod_{(E_i,E_j)\in\{A_k\}} \mathcal{N}(E_i - E_j \mid \mathbf{0},\, \sigma_e^2 \mathsf{I}_2), \tag{8.19}$$

   with $\{E_i\}$ the collection of all wall edges, $\{A_k\} := \{(E_i, E_j) : j = \arg\min_j |E_i - E_j|\}$ (edge pairs where the second is the closest edge to the first) and σ_e a measure for the deviation noise between the edges.

2. Adjacent walls tend to be connected in fixed angles. Here we used a simple 90° prior using a Gaussian on the angle between two walls:

$$p(\{c_i\}) \;=\; \prod_{c_i} \mathcal{N}\left(\theta_{c_i} \;\middle|\; \frac{\pi}{2}, \sigma_\theta^2\right), \tag{8.20}$$

   where $\{c_i\}$ are all wall connections and $\sigma_\theta = 5°$. Equivalently, a mixture of Gaussians can be used to allow for different angles. It is for example plausible that junctions of 90° and 135° often occur, but never an angle in between.

Part of this prior information is applied inside the EM algorithm. Besides this EM update, there is a second important application for the wall interaction model as we can use our model to compute the likelihood of the solution and thus compare solutions with different numbers of walls.

Interaction model inside EM

The prior on the edge locations gives the highest probability to connected edges. Since the M-step should maximize likelihood, we extend this step to enforce a connection of nearby edges[5] by extending the walls (Figure 8.20). This procedure changes the wall parameters as little as possible.

[4] Support vector regression is a regression technique which bases its estimation only on a subset of the training data. It was proposed by Vapnik et al. in 1997 [70,71].

[5] We defined nearby as within 2 m, but experiments show that this paramaeter has little influence.

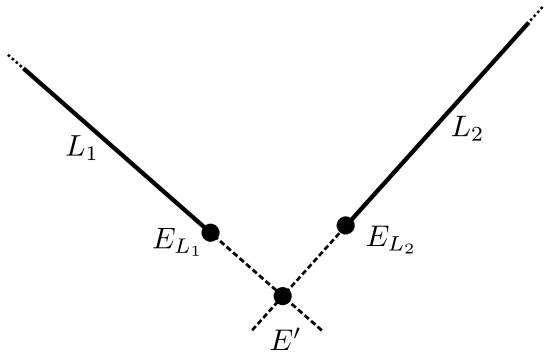


Figure 8.20: The connection procedure of two walls.

We do not incorporate the second prior into the EM algorithm here, because we want to evaluate its influence separately. Section 8.4 discusses an extension that incorporates this prior inside EM.

Probability assessment

As stated earlier, there is no prior on the parameters of separate walls (equation (8.12)). Instead, prior information on wall interaction is used. The prior can be used to evaluate the likelihood of the parameters given the data:

$$p(\{L_k\}|\boldsymbol{X}) \sim p(\boldsymbol{X}|\{L_k\})\ p(\{L_k\}) \tag{8.21}$$
$$= p(\boldsymbol{X}|\{L_k\})\ p(\{A_k\})\ p(\{c_i\}) \tag{8.22}$$

where $p(\boldsymbol{X}|\{L_k\})$ is easily evaluated (see section 8.3.2) and we can in practice ignore $p(\{A_k\})$ as it will roughly be the same for all edges because of the adapted M-step.

8.3.4 Room model

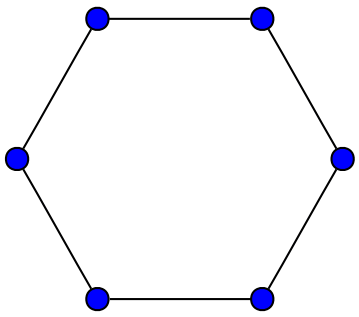


Figure 8.21: Example of a cycle graph of length 6 (source: Wikipedia).

Our room model has one constraint: all walls should form a cycle graph. A cycle graph is a graph where all vertices are connected in a closed chain (Figure 8.21). The algorithm to check this is very easy. Since all walls have at most two neighbours, we choose one line at random and follow a path until we reach the same line again. If this process cannot be completed or if not all walls were seen, this is not a cycle graph.

When computing two solutions, we will always prefer a cycle graph over a non-cycle graph.

8.3.5 Room detection

We have discussed all models that we use and we explained how we can calculate a set of walls and evaluate their likelihood. The following describes how we merge this into a functional algorithm.

Shape	# walls in shape	# points	K	time (s)
Figure 8.5(a)	4	600	4	0.6
Figure 8.5(a)	4	600	8	11.5
Figure 8.2	8	720	4	3.3
Figure 8.2	8	720	8	15.9

Table 8.1: Calculation times of one EM solution (average out of five experiments). Calculations were done with python/numpy on a single 3 GHz core. It is expected that these times could be greatly reduced when implemented in C or C++.

Firstly, it should be noted that the quality of the EM solutions can vary a lot. For complex rooms with 8 walls, it can happen that only one out of eight solutions is good. Our remedy for this is to execute the EM algorithm multiple times (e.g. 20 times) and keep the best solution. This is a very expensive remedy in terms of execution time. However, the total execution time is still acceptable because the EM calculations are light and convergence is fast (see Table 8.1 for calculation times of one solution). Moreover, multiple executions of the same algorithm are done on different processor cores simultaneously.

Secondly, if we stick with our 90° assumption, the number of walls K must be even. The possible values for K are explored in ascending order. When the found solution is worse two times in a row (for the two next K values), the algorithm is aborted and the best found solution is returned. This way, we leave room for improvements with larger K values, keeping in mind that every step takes more time (see Table 8.1).

The combined algorithms described above are thus capable of detecting an arbitrary number of walls in a point cloud. An illustration is shown in Figure 8.22, which also shows some specific information about the EM-algorithm and the wall mixtures.

8.4 EM for Wall Mixtures with angle prior

This section describes an extension to the previous approach. We will try to incorporate the prior on the connection angles into the EM algorithm. This prior was defined in equation (8.20):

$$p(\{c_i\}) \;=\; \prod_{c_i} \mathcal{N}\left(\theta_{c_i} \,\middle|\, \frac{\pi}{2}, \sigma_\theta^2\right) \tag{8.23}$$

Since the wall parameter α is the only parameter which has an impact on the connection angles, we will change the M-step assignment of α. Recall that its value was previously

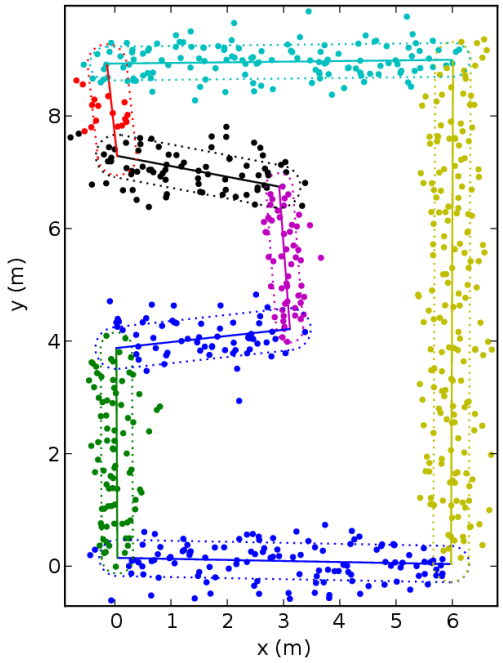


Figure 8.22: Typical outcome of the wall mixture positions for the complex room test case with $\sigma = 300$ (see section 8.6 for definitions). The dotted lines symbolize the wall mixture shapes and the point colors indicate which mixture has the highest responsibility for a given point.

assigned by the main axis of the fitted Gaussian. Now, we will start from equation (8.7):

$$\{L_k\} = \arg\max_{\{L_k\}} p(\{L_k\}|\boldsymbol{X}), \tag{8.24}$$

which we immediately simplify to

$$\{\alpha_k\} = \arg\max_{\{\alpha_k\}} p(\{\alpha_k\}|\boldsymbol{X}), \tag{8.25}$$

where we only minimize the equation for α because it would again take too much time to optimize all parameters simultaneously. We will only take the angle prior into account at this point. This gives:

$$\{\alpha_k\} = \arg\max_{\{\alpha_k\}} p(\{c_i\}) \cdot p(\boldsymbol{X}|\{\alpha_k\}) \tag{8.26}$$

$$= \arg\max_{\{\alpha_k\}} p(\{c_i\}) \cdot \prod_n \sum_k \pi_k Q(\boldsymbol{x}_n|L_k), \tag{8.27}$$

still using the simplified pdf for Q. Appendix C.6 tries to solve this, but shows that again, the optimal α's can not be computed directly. Therefore, we chose to apply gradient ascent with the gradient from appendix C.6:

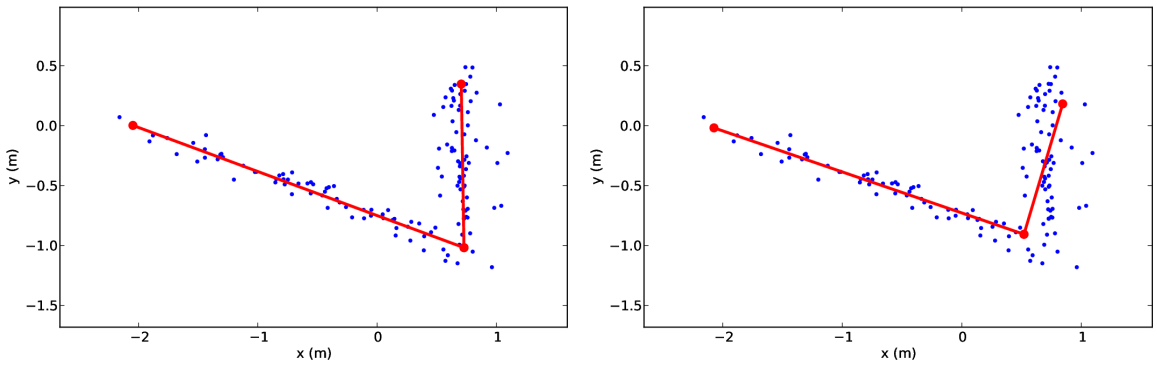
$$\begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_K \end{bmatrix} \leftarrow \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_K \end{bmatrix} + \beta \, \nabla_{\{\alpha_k\}} \ln p(\{L_k\}|\boldsymbol{X}) \tag{8.28}$$

which theoretically solves the problem. We limit the number of gradient ascent iterations so it doesn't slow down the algorithm too much. Note that the other parameters are calculated in the same way as the previous section, except for σ, which is reformulated to

$$\sigma_k^2 = \frac{1}{N_k} \sum_n \gamma(z_{nk})\, y^2 \tag{8.29}$$

where the coordinate system (x, y) is still attached to the wall.

There are some problems with the aforementioned algorithm. The main issue is that the adapted EM algorithm has trouble converging because of the gradient ascent jumping from one local minimum to another. This was partly solved by recalculating the other wall parameters during gradient ascent (but not the responsibilities), which prevents an ever changing optimum. A second issue is the slow evolution towards a local optimum. This conflicts with the first problem because when we allow faster evolution, the chances of infinitely jumping over the maximum also increases.



(a) Wall mixtures

(b) Wall mixtures with angle prior

Figure 8.23: Comparison of the wall mixtures algorithms with and without the angle prior extension. The example is a simple self-generated example, where the connection is deliberately chosen smaller than 90° to compare the behaviour.

Figure 8.23 compares this approach with the previous one on a simple example, and clearly shows an improvement. Section 8.6, which describes more detailed results, will also show a slight performance increase. This comes at the cost of longer processing time (a single step takes more time and due to slower convergence, more steps are needed) and two additional significant meta-parameters (β and the number of gradient ascent steps).

8.5 Preprocessing filtering

8.5.1 Outlier filtering

Our wall mixture model assumes Gaussian noise on the measurements. This implicitly also assumes that there are no outliers. Unfortunately, there are a lot of outliers, as can be seen in Figure 8.24. We try to solve this issue by filtering the points before they are processed by the room detector.

Our filter algorithm is based on clustering. A point belongs to a cluster if the distance between this point and the nearest point in the cluster is smaller than a threshold T. In prac-

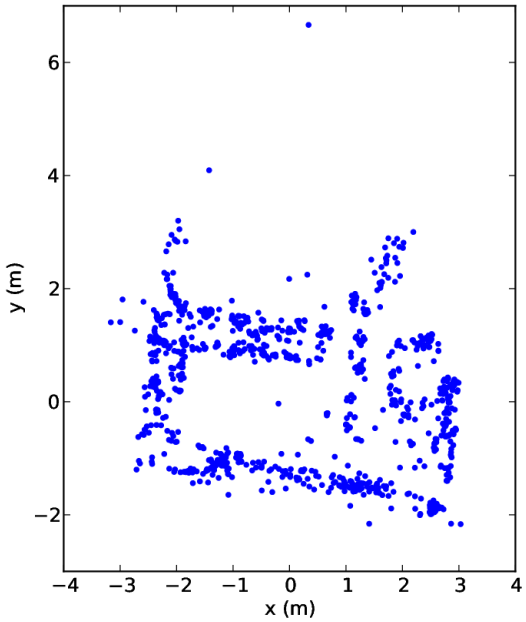
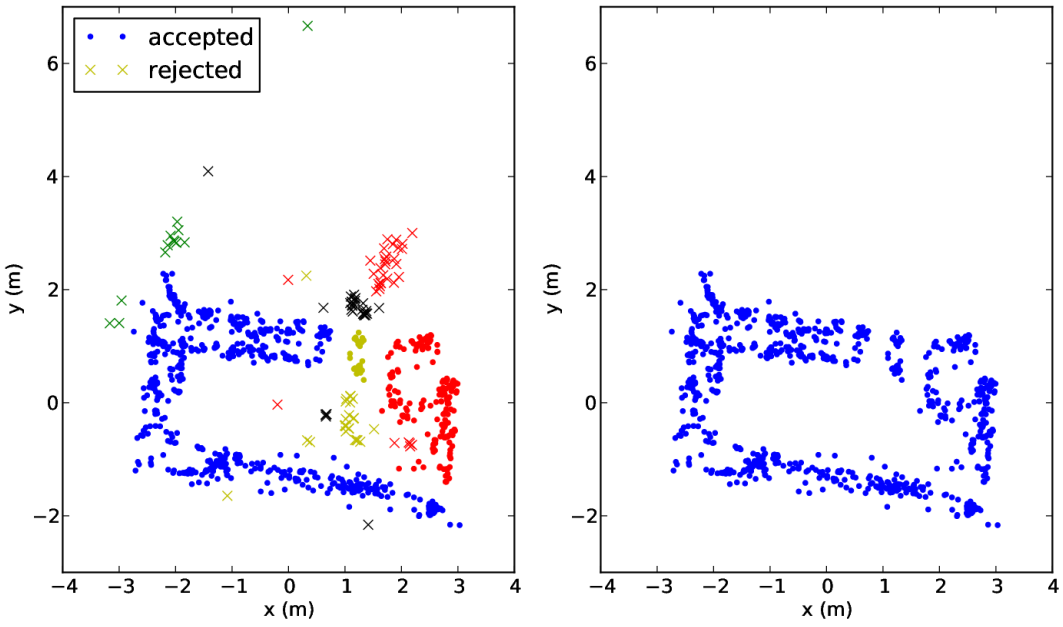


Figure 8.24: Typical points input



(a) Clusters (b) Filtered points

Figure 8.25: Illustration of outlier filtering.

tice, this ensures that all clusters are divided by pointless channels with minimum width T. This clustering can be done quite efficiently. With the same setup as in Table 8.1 and a python/numpy script, it takes about half a second to cluster 1000 points.

Now for the actual filtering, we simply remove all clusters with less than N points. This ensures that small groups of correlated outliers are taken out. For our parameter choices $T = 0.4\,\mathrm{m}$ and $N = 30$, the result on the test case is shown in Figure 8.25.

8.5.2 Density ceiling

When running experiments, we noticed that PTAM sometimes finds a large number of points on small spots (e.g. plants). This gives areas of high wall-point concentrations. Furthermore, high density areas tend to deteriorate wall detection because too much weight is given to a single spot.

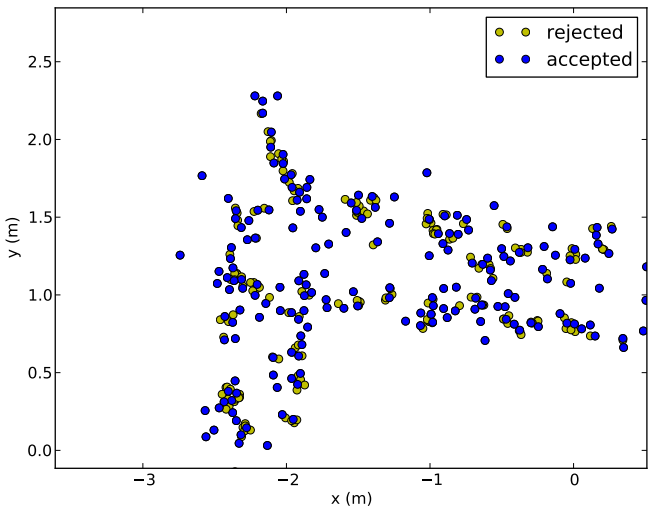
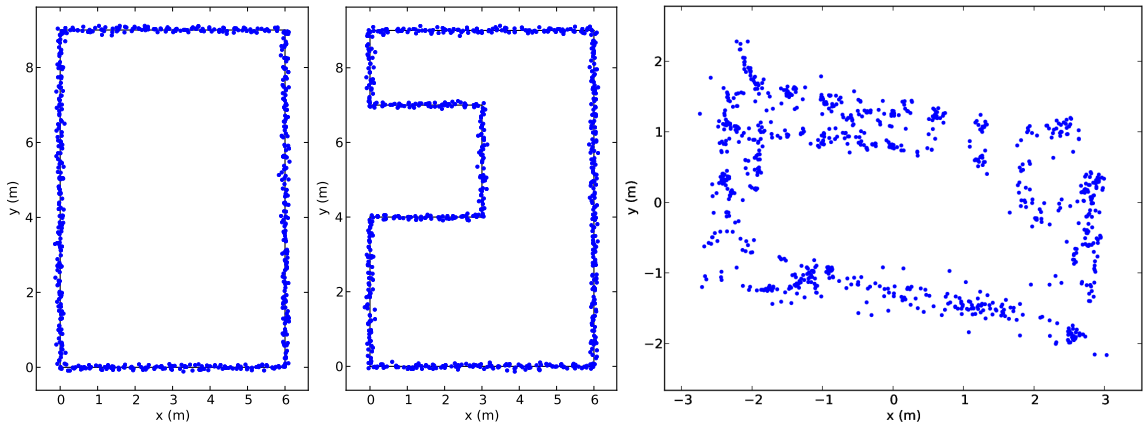


Figure 8.26: Illustration of the density ceiling algorithm.

We cope with this by adding an additional input filter that enforces a maximum density. The algorithm is quite simple: iterate over all points and only accept a point if it has no accepted neighbour within 5 cm. A typical result is shown in Figure 8.26. Experiments show a significant increase in performance after density ceiling on many datasets.

8.6 Results



(a) Rectangular room (b) Complex room (c) PC-room mapped by PTAM

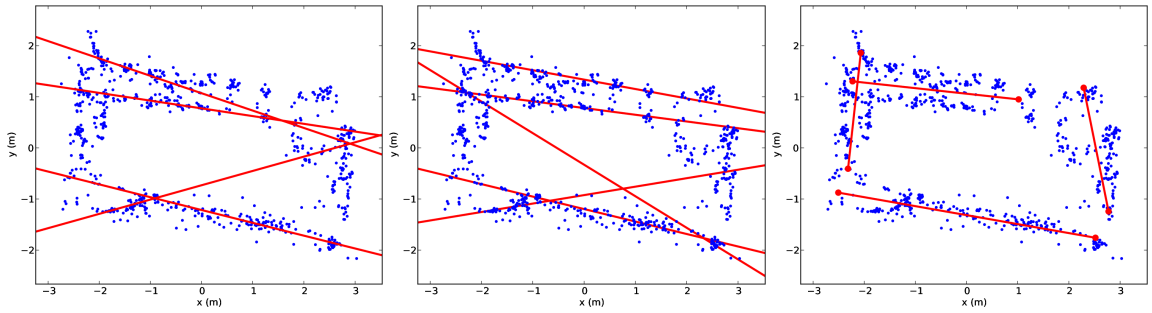
Figure 8.27: The three test cases for room detection. (a) en (b) have added noise with $\sigma = 50$ mm.

To verify our approaches, we compared them to two other algorithms, using three test cases (Figure 8.27). The first two cases were computer-generated, using a similar model as our

wall-sample model. The most important property of the generation procedure is the standard deviation of the noise added to a point on the line. We call this σ, which is consistent with the σ used in the wall mixtures.

The first algorithm to which we compare, uses the Hough transform. We apply the standard algorithm for transforming points to a Hough space with 460 values for θ and 360 for ρ. To find the line parameters, we search for K local maxima, where we provide K beforehand to simplify the task. The second algorithm is the statistical learning approach by Chin et al. [64]. This is a state-of-the-art algorithm for line fitting with many outliers (as discussed in section 8.1). The authors published their source code which we used without any modifications. Unlike the previous method, this one automatically finds the number of lines (just like ours).

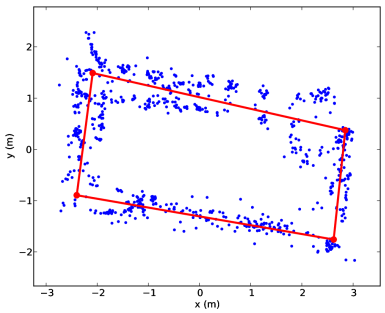
Figures 8.30 - 8.33 compare the algorithms using the computer generated test cases. Each row describes a different wall-generating algorithm and compares the generated walls for multiple room complexities and noise levels. Besides our final algorithms, we included EM for Wall Mixtures without interaction prior and room model so we can fairly compare with rows **A** and **B**, which do not use prior information. We see that our wall mixtures perform better than the others, even without prior. The positive influence of the interaction and room model layers is clear when comparing row **C** and **D**. Comparing row **D** and **E**, we see that de differences are mostly small.



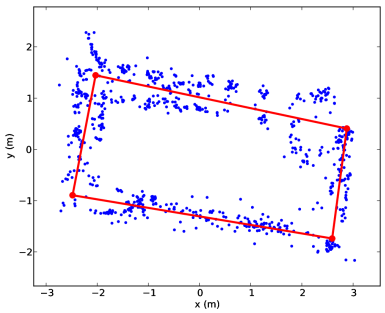
A Hough transform

B Statistical learning

C Wall mixtures without interaction/room model



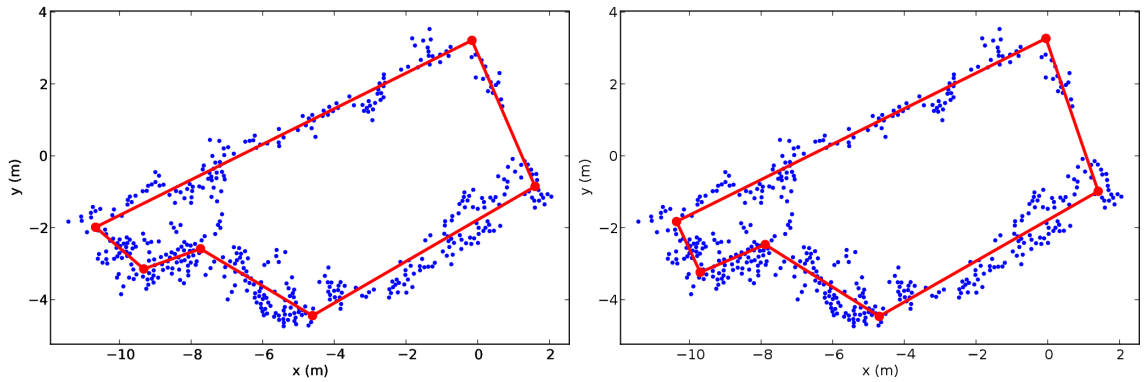
D Wall mixtures



E Wall mixtures with angle prior

Figure 8.28: Comparison of the algorithms for the PC-room test case.

Figure 8.28 shows that the only our wall mixture approach can solve this realistic problem. In Figure 8.29, we test our approach on a more challenging room geometry and see that the detection is only roughly correct. These last two real life examples also show a rather small



D Wall mixtures

E Wall mixtures with angle prior

Figure 8.29: Wall mixtures applied to a dataset recorded at the reception of the ELIS departement. This dataset was recorded by PTAM in the same way as the PC-room.

improvement of the angle extension.

A movie on the attached CD demonstrates the process of fitting walls, starting from the video stream[6].

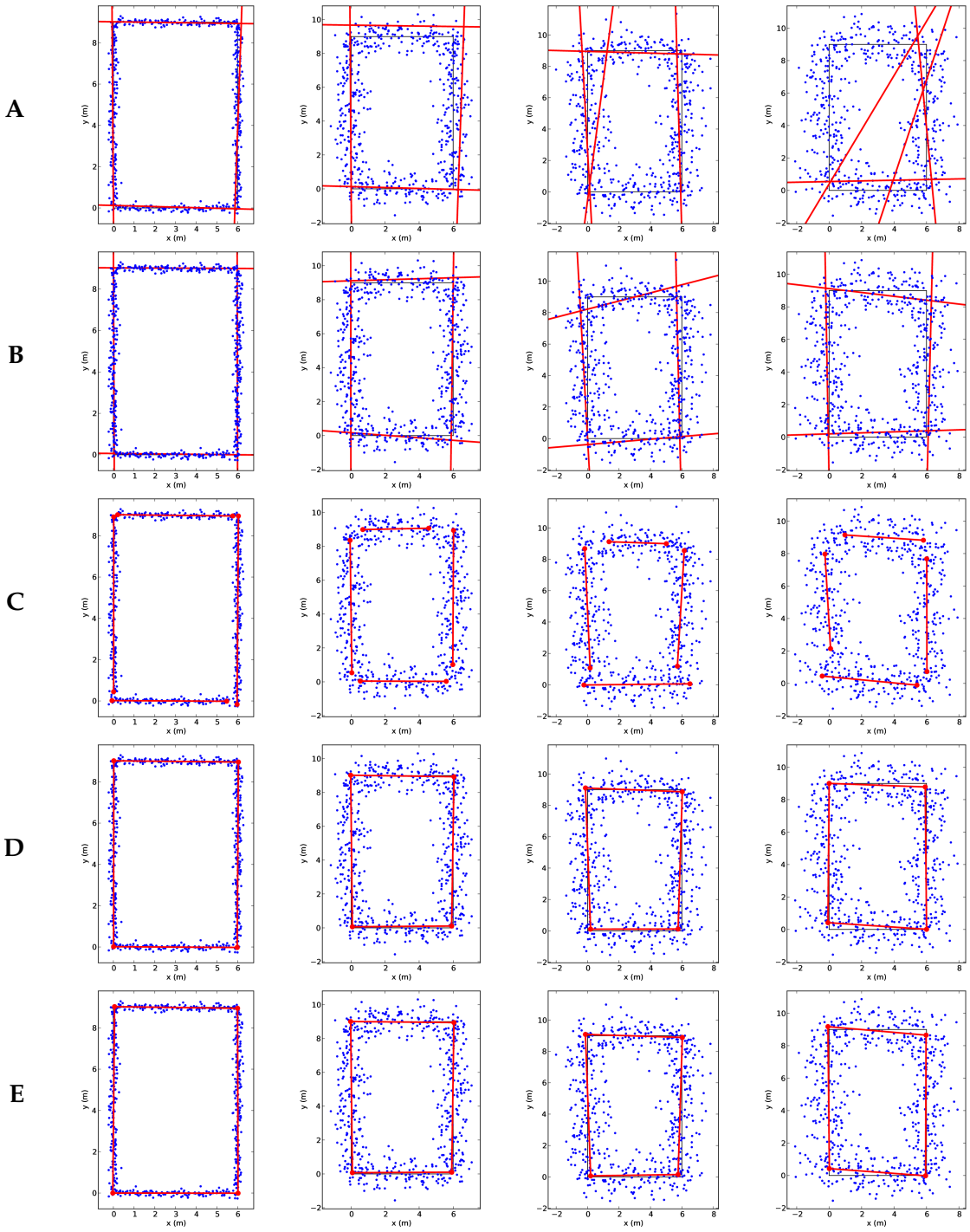
8.7 Conclusion

We described a robust probabilistic algorithm for room fitting based on a noisy point cloud. We attained efficient room fitting by using wall mixtures combined with the EM algorithm and developed a room model which incorporates prior knowledge (e.g. common angles between walls).

We compared other relevant approaches to our layered room model and conclude that even our bottom layer model outperforms the other techniques. Furthermore, the robustness of our method was assessed using multiple realistic datasets.

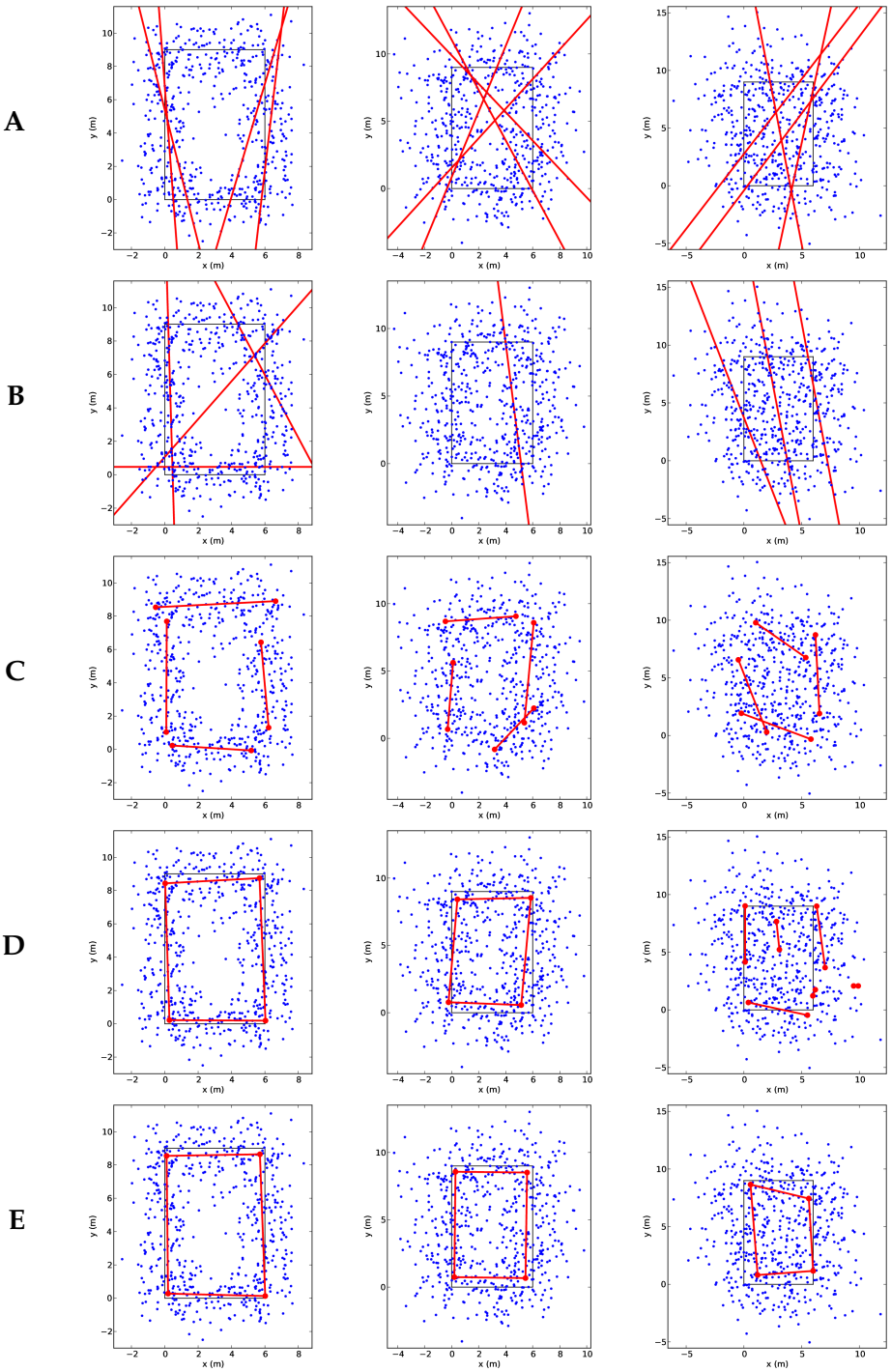
Because of its probabilistic nature, the model can be extended to include various types of prior knowledge about room geometries.

[6]This movie can also be found on YouTube: `https://www.youtube.com/watch?v=E35xbo3r8rA`



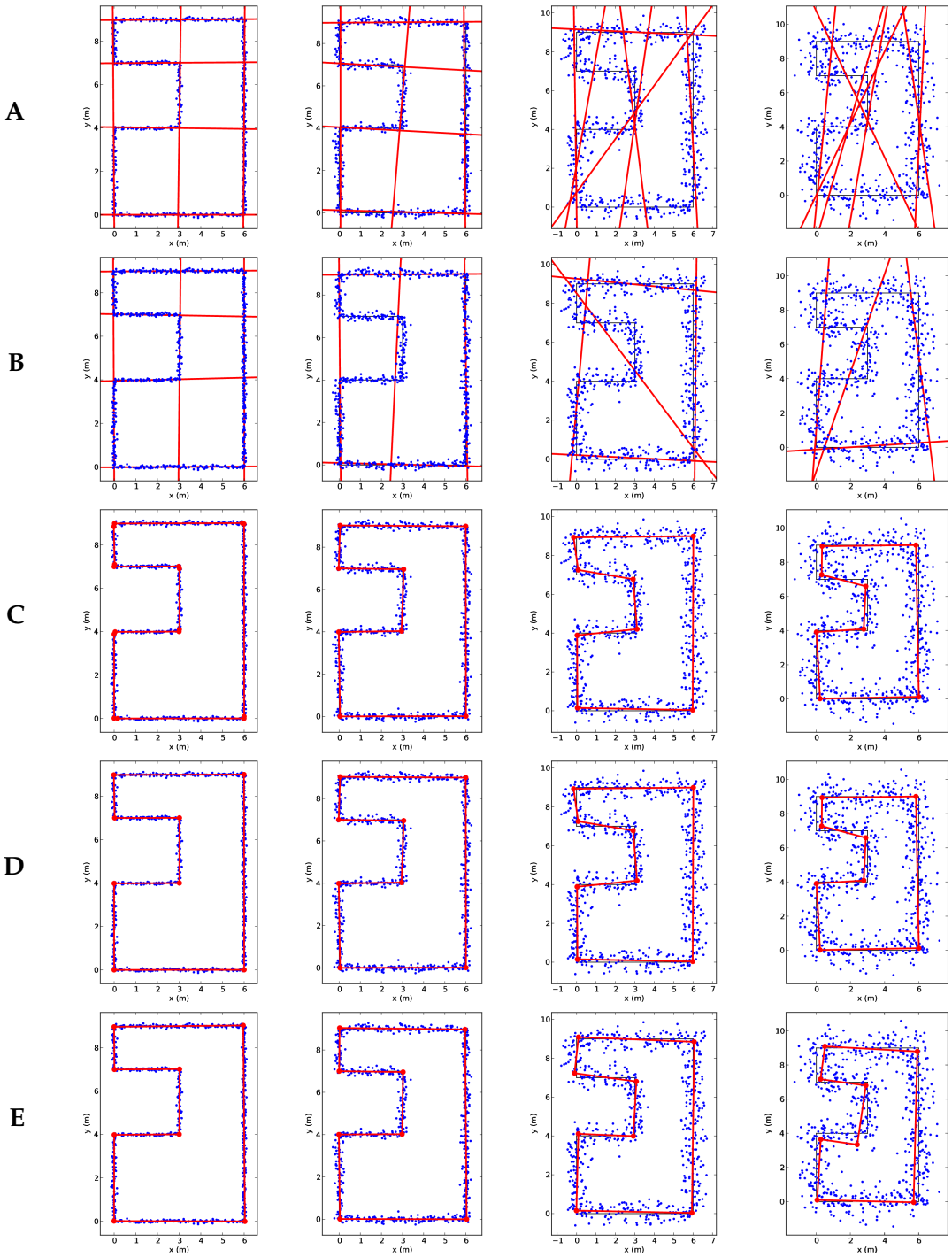
(a) $\sigma = 100\,\text{mm}$ (b) $\sigma = 500\,\text{mm}$ (c) $\sigma = 700\,\text{mm}$ (d) $\sigma = 800\,\text{mm}$

Figure 8.30: Comparison of the algorithms for the rectangular room test case, part 1. **A**: Hough transform, **B**: Statistical learning, **C**: Wall mixture (without interaction prior and room model), **D**: Wall mixtures, **E**: Wall mixtures with angle prior.



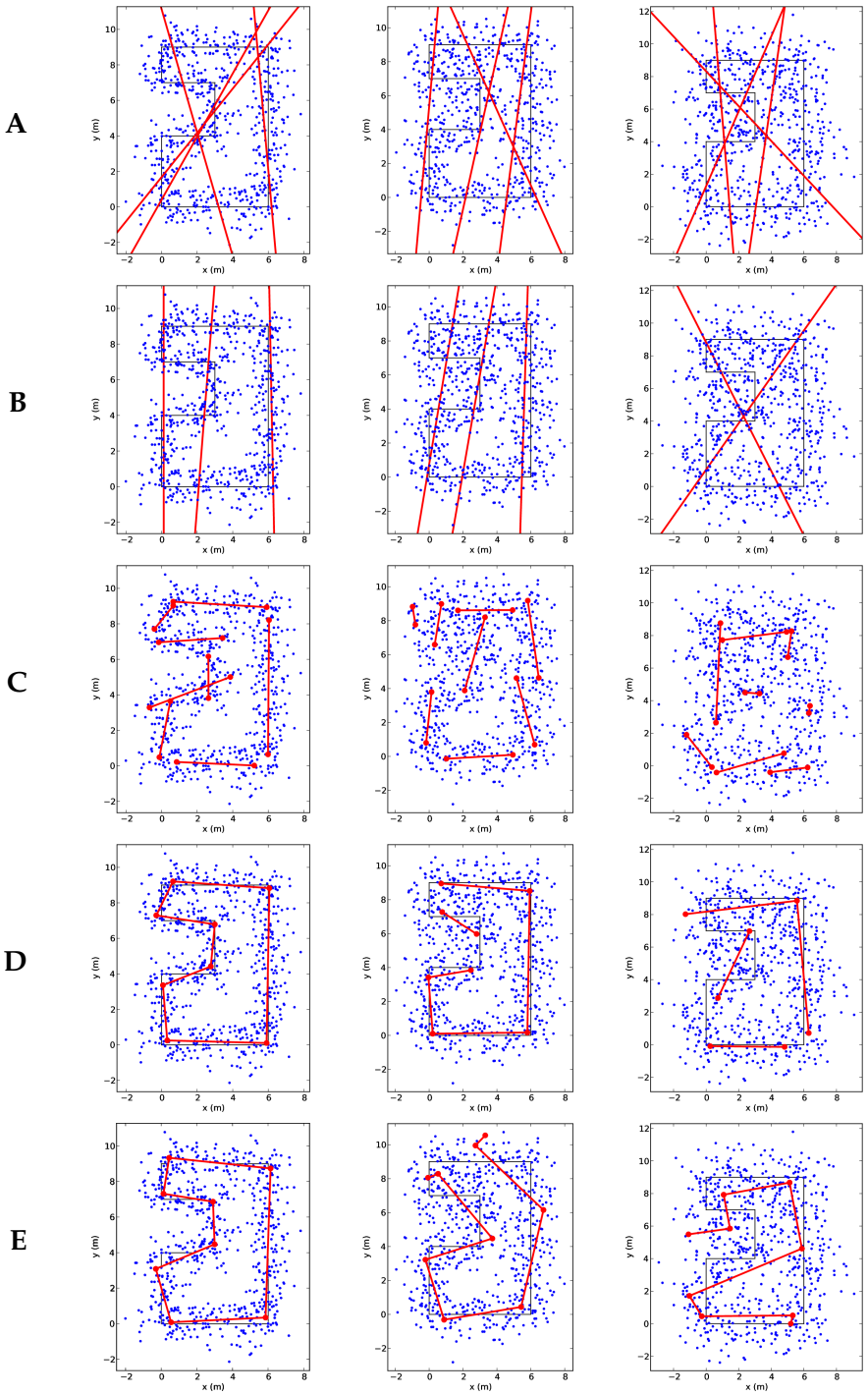
(a) $\sigma = 1000\,\text{mm}$ (b) $\sigma = 1500\,\text{mm}$ (c) $\sigma = 2000\,\text{mm}$

Figure 8.31: Comparison of the algorithms for the rectangular room test case, part 2. **A**: Hough transform, **B**: Statistical learning, **C**: Wall mixture (without interaction prior and room model), **D**: Wall mixtures, **E**: Wall mixtures with angle prior.



(a) $\sigma = 50$ mm (b) $\sigma = 100$ mm (c) $\sigma = 300$ mm (d) $\sigma = 500$ mm

Figure 8.32: Comparison of the algorithms for the complex room test case, part 1. **A**: Hough transform, **B**: Statistical learning, **C**: Wall mixture (without interaction prior and room model), **D**: Wall mixtures, **E**: Wall mixtures with angle prior.



(a) $\sigma = 700\,\text{mm}$ (b) $\sigma = 800\,\text{mm}$ (c) $\sigma = 1000\,\text{mm}$

Figure 8.33: Comparison of the algorithms for the complex room test case, part 2. **A**: Hough transform, **B**: Statistical learning, **C**: Wall mixture (without interaction prior and room model), **D**: Wall mixtures, **E**: Wall mixtures with angle prior.

Visualization | 9

Figure 9.1: Example of textured walls, created by a ground robot with a laser scanner and a panoramic camera [5]. [Copy of Figure 1.3]

This chapter explains how we construct a map which intuitively shows the information that an autonomous quadcopter has gathered. We are inspired by the visualization by Biber et al. [5] and we basically want the same map as theirs (Figure 9.1). The big difference is that we don't have a laser scanner and a panoramic camera. Instead, the only input we have is a simple webcam.

In the following, we first specify the actual input data. Then we briefly mention the 3D software environment. Thereafter, we discuss the hardest part, i.e. the texture generation. We will finally explain what needs to be done to stitch multiple rooms together.

9.1 Input data

The input data that we use for wall visualization is:

- the PTAM points
- about 10-50 video frames, including correspondences between pixels and map points (one set for every room)
- the walls from the room detector (one set for every room)
- the door locations (one couple for every room connection)

The video frames can correspond with the keyframes or can be periodically or manually sampled. The most important thing is that the images fully cover the walls and the floor.

The door locations have to be created manually due to the lack of a door detection algorithm. This is done by pressing a button in PTAM when inside a door gap. In chapter 4, we explained that we work with a room-based methodology. This means that we re-initialise the PTAM map every time we enter a new room. Consequently, a door indication should be done twice, once in both maps.

It should be noted that the PTAM points are very noisy. The point correspondences within a frame often contradict each other significantly. Moreover, projecting two frames to the same spot often results in quite different textures.

9.2 Software

Initially, we wanted to use rviz for visualization, since we already used it for debugging. Unfortunately, rviz doesn't support textures, so we were forced to reconsider.

We finally chose *pygame* combined with *pyggel*. Pygame is a python library for fast development of unconventional[1] user interfaces. Pyggel is a simple 3D game engine for python that mainly acts as an OpenGL abstraction layer.

In this environment, we implemented a first person and a third person view. Everything is kept as intuitive as possible so the user can focus on the visualised data. We call our software *wallviz*.

9.3 Texture generation

Getting plain walls into wallviz is relatively easy. Getting the textures is a more challenging issue. The general idea is to project frames onto the wall, using the correspondences between frame pixels and map points. Once this works, we project all frames onto the walls and the floor. We will assume convex rooms for simplicity, but we will come back on this later.

9.3.1 Frame projection

Figure 9.2 shows our situation. We are given the position of the camera, a number of pixel coordinates and a range of corresponding map points, that not necessarily lie on a wall. The number of points N is typically between 40 and 100.

First, we need to project the map points onto the wall we want to fill. We use the camera position P_C as projection center (see Figure 9.2). We check that the point falls between the wall boundaries and convert it to a coordinate on the texture image. An important factor we derive from the projection is λ, which denotes the position of P^m on $[P_C, P^w]$:

$$P^m = (1 - \lambda)P_C + \lambda P^w \tag{9.1}$$

A very accurate point will have $\lambda = 1$ and the relative projection error can be defined as $|\lambda - 1|$.

[1] with unconventional, we mean a user interface which doesn't consist of text, buttons, scrollbars etc.

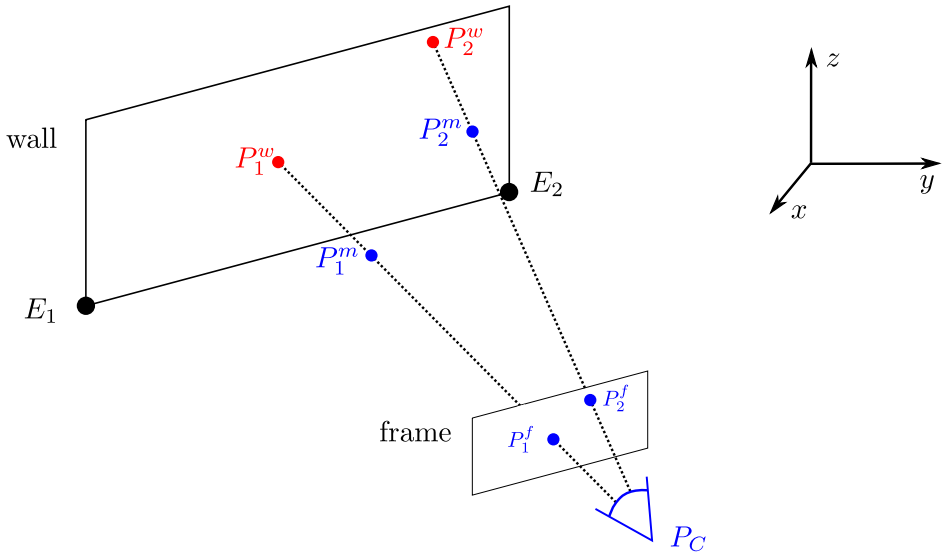


Figure 9.2: The situation before projection. The blue points (P_C, P_n^f, P_n^m) originate from PTAM, the red are to be calculated. P_C is the position of the camera, $\{P_n^f\}$ are pixel coordinates in the frame, $\{P_n^m\}$ are corresponding map points and $\{P_n^w\}$ are the map points projected onto the wall.

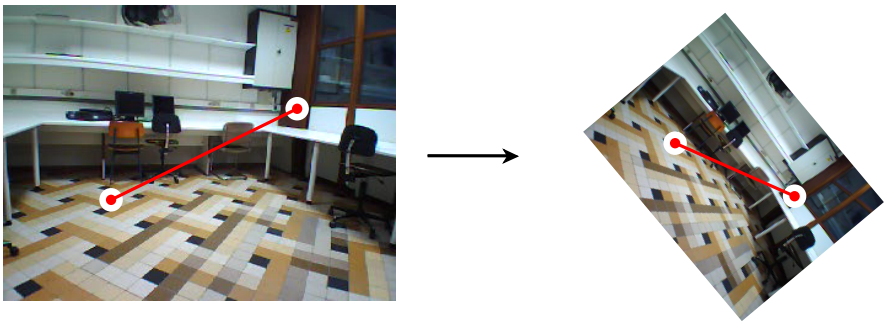


Figure 9.3: Example of a 4 DoF transformation, based on two corresponding points.

If we want to transform a frame to the wall texture image, some choices have to be made. The first choice is the number of degrees of freedom that our transformation should have. If we only allow scaling and rotation, we get a 4 DoF transformation which conserves angles. This transformation is illustrated in Figure 9.3 by considering two corresponding points. The bottom line is that scaling is the same in all directions.

Figure 9.4 illustrates the alternative with different scaling in different directions: a 6 DoF transform, for example possible with three points. To model a real projection on a surface, this is the only correct possibility. However, because the points are so noisy, the 4 DoF transformation has the advantage that the task is easier. Because we prioritize perpendicularly shot frames[2] (see next section), the 4 DoF transformation can actually perform better in some cases. There is an important exception to this rule, i.e. the floor texture. In images of the floor, the camera is rarely directed completely downwards. Figure 9.5 shows the difference

[2]note that perpendicularly shot frames do not stretch out and so a 4 DoF transformation can correctly model this.

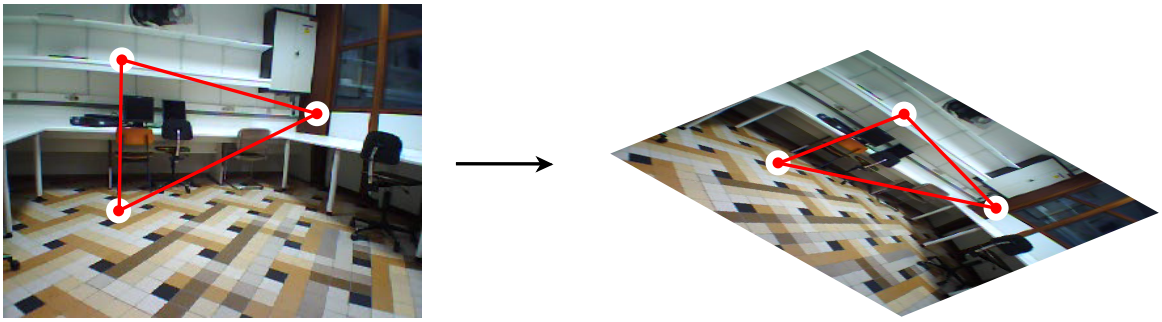


Figure 9.4: Example of a 6 DoF transformation, based on three corresponding points.

(a) 4 DoF projection (b) 6 DoF projection

Figure 9.5: Projection of a frame to the floor. The grey area is the area to be filled.

between the two for an example frame. This clear advantage is why we chose the 6 DoF approach.

The second major choice we have to make, is how we will find the transformation. For this purpose, we defined an error measure. We want to minimize the squared sum of the projection errors. The projection errors are the deviations of projected example points (i.e. the given set of corresponding points). It will be shown that the method naturally flows out of this definition.

Appendix C.7 shows that the 6 DoF transform can be described as

$$\begin{bmatrix} P^w \\ \hline 1 \end{bmatrix} = \left[\begin{array}{cc|c} a & c & e \\ b & d & f \\ \hline 0 & 0 & 1 \end{array}\right] \cdot \begin{bmatrix} P^f \\ \hline 1 \end{bmatrix}. \tag{9.2}$$

Now, we reformulate our problem: find the parameters $(a, \ldots, f)$ of a linear transformation so that the sum of the squared errors of a train set is minimized (with train set, we mean the given point correspondences). This is exactly the definition of linear regression, which can be solved elegantly and efficiently. With the reasoning and definitions of appendix C.8, this

gives

$$\left[\begin{array}{cc} a & b \\ c & d \\ \hline e & f \end{array}\right] = \mathsf{W} = \left(\mathsf{X}^{\mathrm{T}}\,\mathsf{X}\right)^{-1}\mathsf{X}^{\mathrm{T}}\,\mathsf{T} \tag{9.3}$$

which solves our problem. However, this solution does not yet incorporate how certain we are of a point. Earlier, we defined an error measure ($|\lambda - 1|$) for point projections. We incorporate this by using weighted linear regression [72]:

$$\left[\begin{array}{cc} a & b \\ c & d \\ \hline e & f \end{array}\right] = \mathsf{W} = \left(\mathsf{X}^{\mathrm{T}}\,\mathsf{R}\,\mathsf{X}\right)^{-1}\mathsf{X}^{\mathrm{T}}\,\mathsf{R}\,\mathsf{T} \tag{9.4}$$

where R is a diagonal matrix with

$$R_{nn} = \tanh\frac{a}{|\lambda_n - 1|}. \tag{9.5}$$

This corresponds with the error function

$$\mathrm{SSE}_{\mathrm{train}} = \sum_n R_{nn}\left|\mathsf{W}^{\mathrm{T}}\,\boldsymbol{x}_n - \boldsymbol{t}_n\right|^2 \tag{9.6}$$

so it's clear that points near the wall get a higher weight, which is desirable because they are more likely to be correct. We chose $a = 0.1$ in our implementation. A typical result of this algorithm is Figure 9.5(b).

9.3.2 Combining frames

The procedure of the previous section can be repeated for every frame so at the end every visible part is filled in. Evidently, many parts of the walls and the floor will have multiple frames to choose from. Therefore, we define a priority rule, so the best frame is always selected.

The priority rule is based on a cost metric. This metric is a weighted sum of two cost indicators, being:

- **Projection cost**:

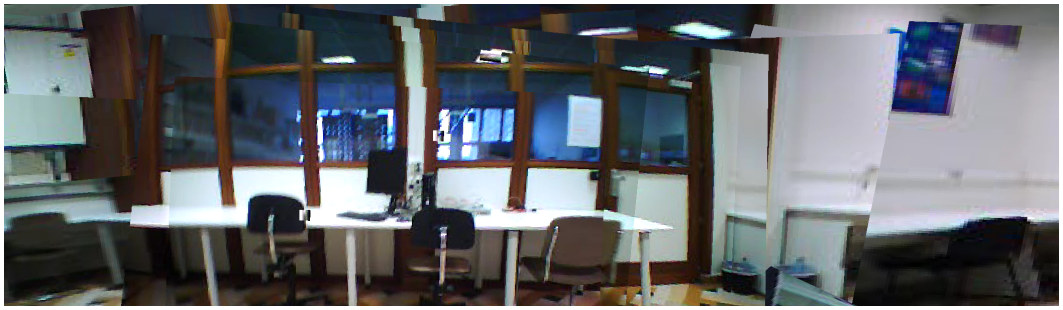
$$\frac{\mathrm{SSE}_{\mathrm{train}}}{\mathrm{PPM}^2 \cdot \sum_n R_{nn}} \tag{9.7}$$

  with $\mathrm{SSE}_{\mathrm{train}}$ defined in equation (9.6) and PPM = "pixels per meter" on the wall texture image. The denominator makes sure this number only depends on the projection quality.

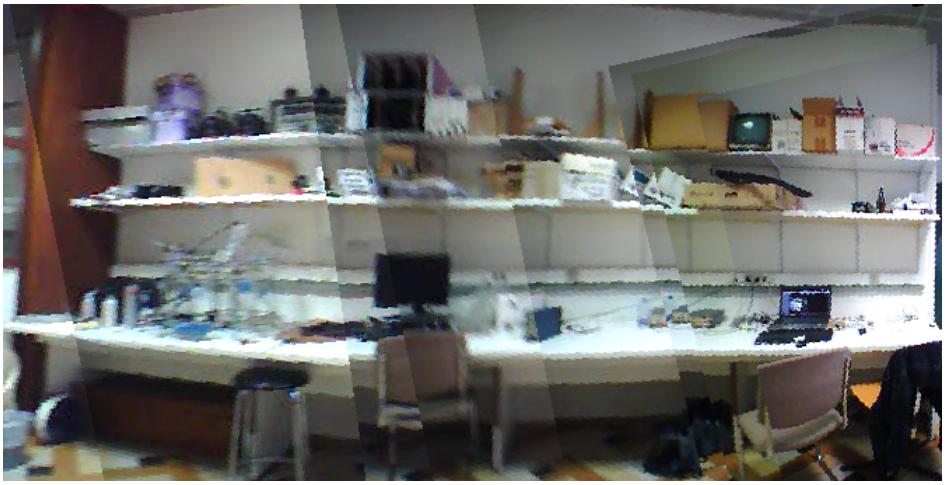
- **Angle cost**:

$$\frac{\measuredangle(\boldsymbol{n},\ P^w_{\mathrm{avg}} - P_C)}{\pi} \tag{9.8}$$

  with $\measuredangle(.,.)$ the angle between two vectors, $\boldsymbol{n}$ the normal vector pointing out of the wall and $P^w_{\mathrm{avg}} = \overline{P^w_n}$ the average wall point.



(a) Wall



(b) Wall

(c) Floor

Figure 9.6: Typical textures that are generated by our algorithm.

The weights were manually optimized, resulting in

$$\text{frame cost} = 5000 \cdot \text{projection cost} + 32 \cdot \text{angle cost}. \tag{9.9}$$

Figure 9.6 shows some typical results of the above approach. The projections are not perfect, and one might see some artefacts here and there, which are probably caused by the noise in the PTAM points. However, the overall impression of the surface is correct and rooms will easily be recognized with these textures. Figure 9.7 shows the final result for one room, rendered in wallviz.

Figure 9.7: Visualization of a room in wallviz. This is a 3D representation of the solution found in Figure 8.28**D** with wall mixtures for the PC room dataset.

9.3.3 Extension for concave rooms

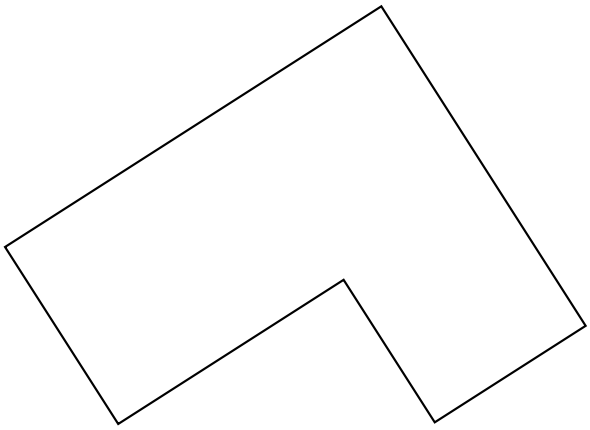


Figure 9.8: Example of a concave room.

Most rooms with four walls are convex, but as rooms get more walls, they are mostly concave (e.g. Figure 9.8). This leads to two problems:

1. It becomes non-trivial to distinguish floor surface from non-floor surface. In the convex case, we required an inside point to be on the in-room side of every wall[3]. This doesn't work for concave rooms.

[3]the in-room side of a wall was determined by looking at which side a known inside point lies. A known inside point could be the mean of the wall edges.

2. Walls stand in the way of presumed projection paths. If this would not be remedied, textures will be distorted by false correspondences.

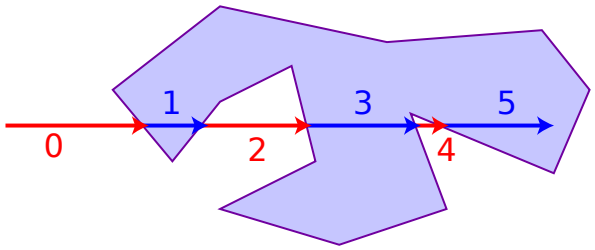


Figure 9.9: Ray casting algorithm (source: Wikipedia).

The first problem is known as the point-in-polygon problem [73, 74]. One of the solutions is the ray casting algorithm. As depicted on Figure 9.9, the algorithm counts the number of edge encounters an incident ray has. A point is on the inside if this number is odd. Practically, we use the python implementation of *Rosetta Code*[4].

This implementation contains a function `ray_intersects_segment()`, which immediately solves the second problem. When a ray from the camera to the wall is intersected by another wall, we ignore the correspondence.

Figure 9.10 shows an example of a map of a concave room.

9.4 Allowing multiple rooms

To stitch rooms together, we are given the door positions. We project these positions to their nearest walls. This leads us to the situation depicted in Figure 9.11. We want to move P_2' to P_1' in a way that makes the walls fit properly. This is a change of basis of which the calculation is treated in appendix C.9.

When we try this on real data, we get the result shown in Figure 9.12. This data originates from actual connected rooms at the ELIS department. The 3D model represents these relatively accurate, and can compete with the work of Biber et al. [5]. A movie on the attached CD demonstrates a quick generation of a similar map[5].

[4] `http://rosettacode.org/wiki/Ray-casting_algorithm`
[5] This movie can also be found on YouTube: `https://www.youtube.com/watch?v=E35xbo3r8rA`

Figure 9.10: Visualization of a concave room in wallviz. This is a 3D representation of the solution found in Figure 8.29**D** with wall mixtures for the ELIS reception dataset.

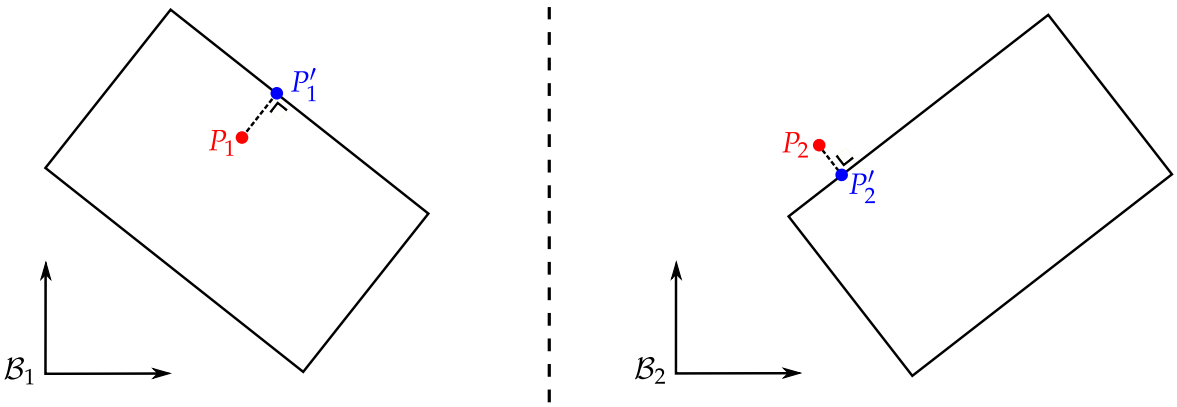


Figure 9.11: Multi-room problem input. The door positions $\{P_i\}$ are projected on the nearest spot on a wall.

Figure 9.12: Visualization of three rooms in wallviz. This dataset was recorded at the PC-room referred to earlier and two other adjacent rooms.

Conclusion | 10

Despite not reaching the ultimate goal, we did construct a novel architecture for indoor autonomous quadcopter exploration, with a focus on a room-based methodology.

Moreover, by implementing the largest part of the architecture, we were able to uncover the remaining barriers for reaching the final goal. We found that the bottleneck of our system is the SLAM algorithm. To be more precise, it's the tracking robustness and the noisiness of the feature locations that form the problem. Recall that in all functional blocks that rely on SLAM features, we had to cope with relatively large deviations. If we want to reach the final goal, the SLAM algorithm should either be replaced or improved.

Apart from these general observations, we achieved some promising results in the individual blocks. The **wall detection** method turns out to perform better than the (to our knowledge) available algorithms. The **multi-camera calibration** procedure works well and can be applied to other problems as well. Finally, our **3D visualization** program creates an intuitive 3D map, despite the noisy input.

Recommendations for future work

If we continue with what we stated earlier, eliminating the bottleneck (SLAM) is essential for further progress. We suggest to replace that PTAM by an (own) implementation of DTAM (section 5.2). Alternatively, one could try to combine PTAM with an image analysis algorithm like a spatial layout estimator. PTAM noise and outliers could be reduced by using prior knowledge from these algorithms.

If the previous improvements would succeed, wall detection could be generalized. Now, the assumptions for wall detection are quite stringent (90° angles, no curved walls, no stairs, ...). If the input becomes less noisy, these requirements can be relaxed without loss of robustness.

Obviously, the missing parts in our architecture should be implemented as well. The main missing parts are door detection and a more intelligent autonomous controller. Also, loop closure should be implemented, e.g. using FAB-MAP [34].

An enhancement for the hardware platform could be to add some distance sensors. These still allow for a cheap, small and light platform, but can provide valuable information for SLAM and autonomous control.

Since our work was devoted to exploration with a low-cost robot platform, it is now possible to extend the architecture to a swarm of robots. In the simplest case, the agents could explore different rooms independently and combine their maps by laying interconnections. A more intense cooperation has even more potential. Suppose for example that every quadcopter in a room knows how far it's away from another actor (e.g. by the round trip time of communication), then they could optimize their collective location together. This is just one of the many possibilities of swarm robotics and it indicates that the simplicity of our platform might actually become the biggest asset.

Embedded software and communication | A

As stated in chapter 3, the AR.Drone contains a Linux computer (an ARM machine). When connected to the wireless access point of the quadcopter, it is possible to connect to the robot via *FTP* and *Telnet*. FTP is obviously used for file transfer from and to the robot, while the latter is used for remote access. When starting a Telnet session, you get root access to the AR.Drone via a command line interface with basic Linux commands.

Note that the Linux kernel of the AR.Drone, together with some kernel modules, is available as open source. The firmware, which handles most communication and the quadcopter control, is not publicly available.

A.1 Official firmware

The closed source firmware is an application that runs in the embedded Linux environment. It provides all quadcopter-specific functionality such as:

- Streaming video
- Stabilizing the quadcopter
- Moving the quadcopter

A.2 Run custom programs

It is possible to compile and run programs on the embedded computer. Since the embedded PC is an ARM machine, all programs need to be cross compiled on a regular PC. This is done by the *Sourcery G++ Lite Toolchain for ARM GNU/Linux*[1]. Then, the compiled file can be transferred via FTP to the quadcopter and be started remotely via Telnet.

A.3 Extra video stream

An important restriction of the official firmware is that it creates only one video stream, while two cameras are available. For research purposes, we wanted two or more video streams from the quadcopter. Two paths were investigated:

- Bypassing firmware restrictions as to stream both existing cameras anyway
- Adding a third camera and streaming that one

[1] The toolchain can be downloaded from `https://sourcery.mentor.com/GNUToolchain/release858`.

A.3.1 Stream both cameras

The most difficult aspect of this problem is the fact that a video device normally can only be read by one program at a time. The official firmware constantly reads both cameras, even when their images are not streamed.

It is possible to stop the firmware, which avoids conflicts, but then the quadcopter also can't fly anymore. This is however still a valuable option for testing and gathering datasets (the quadcopter then has to be moved around by hand). We first consider the problem of streaming video without access conflicts, then we will discuss the device access problem.

Custom videostreamer

Based on work of Hugo Perquin[2], we created software that can stream both cameras simultaneously at maximal frame rates[3]. To achieve this, the largest stream (`video0`) has to be subsampled (factor 2) and jpeg compressed (quality of 90%).

Judging the quality of an individual video stream, our streamer performs better than the official firmware because it doesn't have occasional block artefacts while frame rate and resolution are the same[4]. Moreover, our program provides two streams without loss of performance.

Device access

The common approach if two programs want to access the same video device, is to create an extra virtual read-write device and let the custom streamer write the video stream to it (Figure A.1). The virtual device was created by v4l2loopback[5]. This is a kernel module which we cross-compiled for ARM and loaded on the embedded machine. However, when writing to the virtual device, the kernel stops working (kernel panic). Due to debugging difficulties, this approach was abandoned.

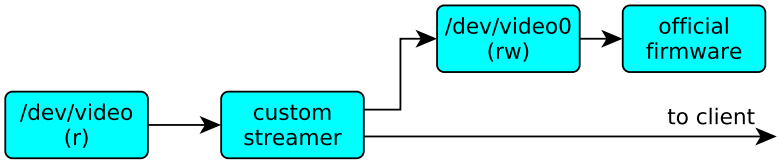


Figure A.1: System for duplicating a video device. (r) means read-only, (rw) means read-write.

A.3.2 Extra camera

The AR.Drone has a USB port, which can be activated by cross-compiling the kernel module `dwc_otg`, loading it and executing some gpio[6] commands (see Procedure 1).

[2]See Perquin's blog: `http://blog.perquin.com/blog/ardrone-motor-controller/`
[3]Frame rates are 30 Hz and 60 Hz for `video0` and `video1` respectively.
[4]The official firmware also subsamples `video0` by a factor 2.
[5]v4l2loopback is open source and can be downloaded at `http://code.google.com/p/v4l2loopback/`.
[6]gpio: general purpose input/output. Controls digital pins of the ARM chip.

Procedure 1 Enable USB support on AR.Drone.

```
gpio 127 -d i
insmod dwc_otg.ko
gpio 127 -d i
```

Now we have a working USB port, it should be possible to add a webcam to the quadcopter. Another kernel module named `uvcvideo` detects attached USB webcams and converts them to video devices. However, the resulting video device is not selectable. At the time of writing, this is still an open problem.

A last option is to create a second video stream without using the AR.Drone hardware, but instead by providing a separate wireless connection. For example, we used the PCB of a second (broken) AR.Drone to stream its cameras to the client. This approach is cumbersome and not very elegant, but it did turn out to work.

Quaternions | B

We use quaternions to represent 3D rotations. This chapter explains some properties of this representation, based on [75] and [76].

B.1 Quaternions

A quaternion is an element of $\mathbb{R}^4$ and is an extension of complex numbers. Instead of just $\mathbf{i}$, we have three different numbers $\mathbf{i}$, $\mathbf{j}$ and $\mathbf{k}$ with

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = 1 \tag{B.1}$$

which leads to the following representation of a quaternion:

$$\boldsymbol{q} = [q_0, q_1, q_2, q_3] = q_0 + \mathbf{i}q_1 + \mathbf{j}q_2 + \mathbf{k}q_3. \tag{B.2}$$

We define the complex conjugate and norm of $\boldsymbol{q}$ as

$$\boldsymbol{q}^* = q_0 - \mathbf{i}q_1 - \mathbf{j}q_2 - \mathbf{k}q_3 \tag{B.3}$$

$$|\boldsymbol{q}| = \sqrt{\boldsymbol{q}^*\boldsymbol{q}} = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} \tag{B.4}$$

B.2 Rotations as quaternions

Now, we can represent a 3D rotation as a quaternion. Without loss of generality, assume a rotation θ around an arbitrary unit vector $\boldsymbol{u}$. We write the corresponding quaternion as

$$\boldsymbol{q} = \left[\cos\frac{\theta}{2},\ u_x \sin\frac{\theta}{2},\ u_y \sin\frac{\theta}{2},\ u_z \sin\frac{\theta}{2}\right]. \tag{B.5}$$

If we represent a point $\boldsymbol{p}$ as

$$\boldsymbol{P} = \left[0,\ p_x,\ p_y,\ p_z\right] \tag{B.6}$$

it can be proven that

$$\boldsymbol{P}_{\text{transformed}} = \boldsymbol{q}\,\boldsymbol{P}\,\boldsymbol{q}^{-1}. \tag{B.7}$$

This results in the very interesting property that the concatenation of rotation 1 (first) and rotation 2 (last) results in the quaternion product:

$$\boldsymbol{q}_{2 \text{ after } 1} = \boldsymbol{q}_2\,\boldsymbol{q}_1 \tag{B.8}$$

which can be calculated faster than matrix products.

Calculations | C

C.1 Multimap SLAM: transform calibration

This chapter discusses the more complex calculations of section 5.7.2. We follow the same definitions as in this section, and try to get an expression for $\hat{\mathsf{Q}}_{B_B}^{C_B}$. We start with

$$\mathsf{Q}_{B_A}^{C_A} \cdot \mathsf{Q}_{C_A}^{C_B} = \left[\begin{array}{c|c} \mathsf{R}_{B_A}^{C_A} & \boldsymbol{T}_{B_A}^{C_A} \\ \hline \mathbf{0} & 1 \end{array}\right] \cdot \left[\begin{array}{c|c} \mathsf{R}_{C_A}^{C_B} & \boldsymbol{T}_{C_A}^{C_B} \\ \hline \mathbf{0} & s \end{array}\right] = \left[\begin{array}{c|c} \mathsf{R}_{B_A}^{C_A} \cdot \mathsf{R}_{C_A}^{C_B} & \boldsymbol{T}' \\ \hline \mathbf{0} & s \end{array}\right] \tag{C.1}$$

with

$$\boldsymbol{T}' = \mathsf{R}_{B_A}^{C_A} \cdot \boldsymbol{T}_{C_A}^{C_B} + s\, \boldsymbol{T}_{B_A}^{C_A}. \tag{C.2}$$

Further, we can deduce that ${\mathsf{Q}_{B_A}^{B_B}}^{-1}$ should be of the form

$$\left[\begin{array}{c|c} {\mathsf{R}_{B_A}^{B_B}}^{-1} & \boldsymbol{T}'' \\ \hline \mathbf{0} & s^{-1} \end{array}\right] \tag{C.3}$$

because

$$\mathsf{Q}_{B_A}^{B_B} \cdot {\mathsf{Q}_{B_A}^{B_B}}^{-1} = \left[\begin{array}{c|c} \mathsf{R}_{B_A}^{B_B} & \boldsymbol{T}_{B_A}^{B_B} \\ \hline \mathbf{0} & s \end{array}\right] \cdot \left[\begin{array}{c|c} {\mathsf{R}_{B_A}^{B_B}}^{-1} & \boldsymbol{T}'' \\ \hline \mathbf{0} & s^{-1} \end{array}\right] = \left[\begin{array}{c|c} I & \mathbf{0} \\ \hline \mathbf{0} & 1 \end{array}\right]. \tag{C.4}$$

Equation (C.4) further constrains $\boldsymbol{T}''$:

$$\mathbf{0} = \mathsf{R}_{B_A}^{B_B} \cdot \boldsymbol{T}'' + s^{-1}\, \boldsymbol{T}_{B_A}^{B_B} \tag{C.5}$$

$$\boldsymbol{T}'' = -s^{-1}\, {\mathsf{R}_{B_A}^{B_B}}^{-1} \cdot \boldsymbol{T}_{B_A}^{B_B} \tag{C.6}$$

We are now able to write $\hat{\mathsf{Q}}_{B_B}^{C_B}$ in function of known rotations and translations:

$$\hat{\mathsf{Q}}_{B_B}^{C_B} = {\mathsf{Q}_{B_A}^{B_B}}^{-1} \cdot \mathsf{Q}_{B_A}^{C_A} \cdot \mathsf{Q}_{C_A}^{C_B} = \left[\begin{array}{c|c} {\mathsf{R}_{B_A}^{B_B}}^{-1} & \boldsymbol{T}'' \\ \hline \mathbf{0} & s^{-1} \end{array}\right] \cdot \left[\begin{array}{c|c} \mathsf{R}_{B_A}^{C_A} \cdot \mathsf{R}_{C_A}^{C_B} & \boldsymbol{T}' \\ \hline \mathbf{0} & s \end{array}\right] \tag{C.7}$$

$$= \left[\begin{array}{c|c} {\mathsf{R}_{B_A}^{B_B}}^{-1} \cdot \mathsf{R}_{B_A}^{C_A} \cdot \mathsf{R}_{C_A}^{C_B} & \boldsymbol{T}''' \\ \hline \mathbf{0} & 1 \end{array}\right] \tag{C.8}$$

with

$$\boldsymbol{T}''' = {\mathsf{R}_{B_A}^{B_B}}^{-1} \cdot \boldsymbol{T}' + s\, \boldsymbol{T}''. \tag{C.9}$$

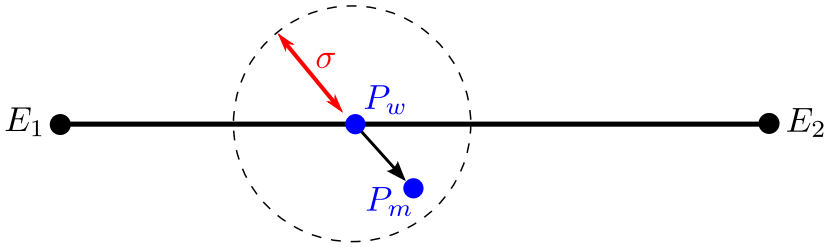


Figure C.1: Model for sampling from a wall. [Copy of Figure 8.13]

C.2 Wall detection: ideal pdf around wall

Figure C.1 recapitulates the situation of section 8.3.1. We want to find the pdf

$$Q(\boldsymbol{x}|L_k) = p(\boldsymbol{x}|z_k = 1, L_k) \tag{C.10}$$

corresponding with the proposed wall-sample model.

The model assumes a latent point P_w randomly chosen on the wall. The measured point P_m is a noisy version of this point:

$$p(P_m|P_w) = \mathcal{N}(P_m|P_w, \sigma^2 \mathsf{I}_2) \tag{C.11}$$

with I_2 the 2×2 identity matrix and $\mathcal{N}(P_m|...)$ the bivariate normal distribution, which is defined as follows for the case of independent x and y:

$$\mathcal{N}\left(\begin{bmatrix} x \\ y \end{bmatrix} \middle| \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}, \sigma^2 \mathsf{I}_2\right) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(x-x_0)^2 + (y-y_0)^2}{2\,\sigma^2}\right) \tag{C.12}$$

This models Gaussian noise that is equal in all directions. The bivariate normal distribution is not to be confused with the univariate normal distribution

$$\mathcal{N}(x \mid x_0, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-x_0)^2}{2\,\sigma^2}\right) \tag{C.13}$$

with

$$\mathcal{N}\left(\begin{bmatrix} x \\ y \end{bmatrix} \middle| \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}, \sigma^2 \mathsf{I}_2\right) = \mathcal{N}(x \mid x_0, \sigma^2)\, \mathcal{N}(y \mid y_0, \sigma^2) \tag{C.14}$$

Using the definitions of Figure C.2, we can calculate Q (with $\boldsymbol{x} = (x, y)$):

$$\begin{aligned} Q(\boldsymbol{x}|L_k) &= p[\boldsymbol{x}|z_k = 1, L_k] && \text{(C.15)} \\ &= p[P_m = (x, y)] && \text{(C.16)} \\ &= \int_{-e}^{e} p[P_{wx} = t]\; p[P_m = (x, y)|P_{wx} = t]\; \mathrm{d}t && \text{(C.17)} \\ &= \frac{1}{2e} \int_{-e}^{e} \mathcal{N}\left(\begin{bmatrix} x \\ y \end{bmatrix} \middle| \begin{bmatrix} t \\ 0 \end{bmatrix}, \sigma^2 \mathsf{I}_2\right) \mathrm{d}t && \text{(C.18)} \end{aligned}$$

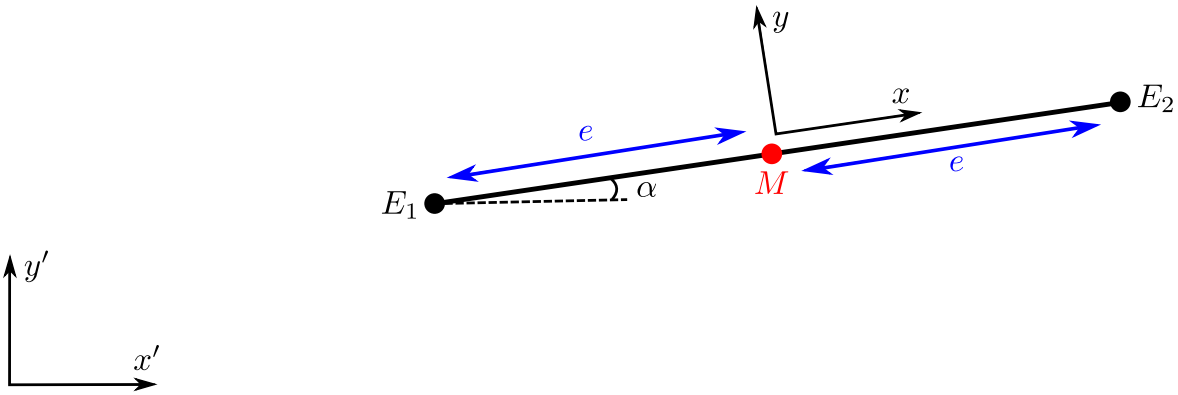


Figure C.2: Definition of parameters of a wall. (x', y') defines the general coordinate system and (x, y) defines a coordinate system attached to the wall with M as origin. [Copy of Figure 8.14]

where $p[P_{wx} = t] = 1/2e$ because P_w is uniformly distributed on the wall segment. We see that this is acually a convolution of the bivariate normal distribution along the wall. When working this out further, one gets

$$Q(\boldsymbol{x}|L) = \frac{1}{2e} \int_{-e}^{e} \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(x-t)^2 + y^2}{2\,\sigma^2}\right) \mathrm{d}t \tag{C.19}$$

$$= \frac{1}{2e}\, \mathcal{N}(y \mid 0, \sigma^2) \int_{-e}^{e} \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(x-t)^2}{2\,\sigma^2}\right) \mathrm{d}t. \tag{C.20}$$

We put this equation into *Maple* (mathematical software by Maplesoft) and found

$$Q(\boldsymbol{x}|L) = \frac{\operatorname{erf}\left(\frac{e+x}{\sqrt{2}\,\sigma}\right) + \operatorname{erf}\left(\frac{e-x}{\sqrt{2}\,\sigma}\right)}{4\,e}\, \mathcal{N}(y \mid 0, \sigma^2). \tag{C.21}$$

C.3 Wall detection: finding e with the ideal pdf

Following section 8.3.2, we can calculate the optimal value of e by inserting the expression for Q of equation (C.21) into equation (8.14). Let us start by restating equation (8.14):

$$0 = \frac{\partial}{\partial e} \ln p(\boldsymbol{X}|L_k) = \sum_n \gamma(z_{nk})\, \frac{\partial}{\partial e} \ln Q(\boldsymbol{x}_n|L_k) \tag{C.22}$$

We will first try to find an expression for $\frac{\partial}{\partial e} \ln Q(\boldsymbol{x}_n|L_k)$ with $\boldsymbol{x}_n = (x_n, y_n)$:

$$\frac{\partial}{\partial e} \ln Q(\boldsymbol{x}_n|L_k) = \frac{4\,e}{\operatorname{erf}\left(\frac{e+x_n}{\sqrt{2}\,\sigma}\right) + \operatorname{erf}\left(\frac{e-x_n}{\sqrt{2}\,\sigma}\right)} \cdot \frac{\partial}{\partial e} \left(\frac{\operatorname{erf}\left(\frac{e+x_n}{\sqrt{2}\,\sigma}\right) + \operatorname{erf}\left(\frac{e-x_n}{\sqrt{2}\,\sigma}\right)}{4\,e} \right) \tag{C.23}$$

$$= 2\, \frac{\mathcal{N}(x_n|\text{-}e, \sigma^2) + \mathcal{N}(x_n|e, \sigma^2)}{\operatorname{erf}\left(\frac{e+x_n}{\sqrt{2}\,\sigma}\right) + \operatorname{erf}\left(\frac{e-x_n}{\sqrt{2}\,\sigma}\right)} - \frac{1}{e} \tag{C.24}$$

again, with help from Maple. Plugging equation (C.24) back into equation (C.22) gives

$$\frac{N_k}{e} = 2 \sum_n \gamma(z_{nk})\, \frac{\mathcal{N}(x_n|\text{-}e, \sigma^2) + \mathcal{N}(x_n|e, \sigma^2)}{\operatorname{erf}\left(\frac{e+x_n}{\sqrt{2}\,\sigma}\right) + \operatorname{erf}\left(\frac{e-x_n}{\sqrt{2}\,\sigma}\right)} \tag{C.25}$$

which has to be solved numerically.

C.4 Wall detection: simplified pdf

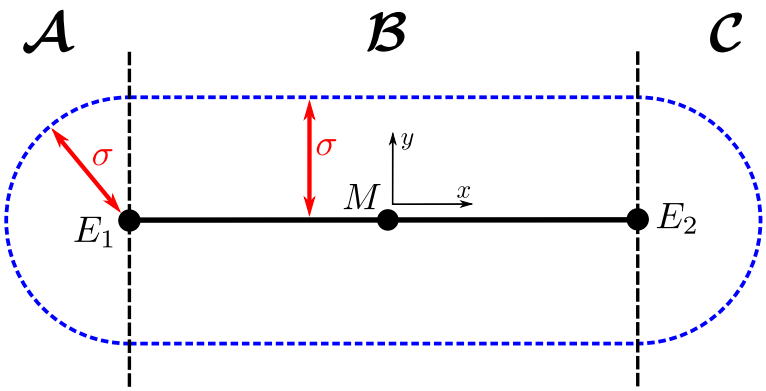


Figure C.3: Schematic diagram of the simplified pdf. The three areas $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ are divided by the black dashed line and the pdf is symbolized by the blue dashed line.

In section 8.3.2, we derived the following expression for Q:

$$Q(\boldsymbol{x}|L) = a \begin{cases} \mathcal{N}(y|0,\sigma^2) & ,|x| \leqslant e \\ \mathcal{N}\left(\sqrt{(|x|-e)^2+y^2}\,\middle|\,0,\sigma^2\right) & ,|x| > e \end{cases}. \tag{C.26}$$

With definitions (C.12) and (C.13), we see that the second expression can be rewritten as a bivariate normal distribution:

$$\mathcal{N}\left(\sqrt{(|x|-e)^2+y^2}\,\middle|\,0,\sigma^2\right) = \sigma\sqrt{2\pi}\ \mathcal{N}\left(\begin{bmatrix} |x|-e \\ y \end{bmatrix}\middle|\,\mathbf{0},\sigma^2\,\mathrm{I}_2\right) \tag{C.27}$$

In order to find a, we need to integrate $Q(\boldsymbol{x}|L)$ for all $\boldsymbol{x}$ and require this to be one. The integration over region $\mathcal{A}$ and $\mathcal{C}$ together can be seen as an integration over a single bivariate normal:

$$Q(\forall \boldsymbol{x} \in \mathcal{A}\cup\mathcal{C}|L) = \iint_{(x,y)\in\mathcal{A}\cup\mathcal{C}} a\,\sigma\sqrt{2\pi}\ \mathcal{N}\left(\begin{bmatrix} |x|-e \\ y \end{bmatrix}\middle|\,\mathbf{0},\sigma^2\,\mathrm{I}_2\right)\,\mathrm{d}x\,\mathrm{d}y \tag{C.28}$$

$$= \int_{x=-\infty}^{x=\infty}\int_{y=-\infty}^{y=\infty} a\,\sigma\sqrt{2\pi}\ \mathcal{N}\left(\begin{bmatrix} x \\ y \end{bmatrix}\middle|\,\mathbf{0},\sigma^2\,\mathrm{I}_2\right)\,\mathrm{d}x\,\mathrm{d}y \tag{C.29}$$

$$= a\,\sigma\sqrt{2\pi}. \tag{C.30}$$

The integration over area $\mathcal{B}$ is also straight forward:

$$Q(\forall \boldsymbol{x} \in \mathcal{B}|L) = \int_{x=-e}^{x=e}\int_{y=-\infty}^{y=\infty} a\,\mathcal{N}(y|0,\sigma^2)\,\mathrm{d}x\,\mathrm{d}y \tag{C.31}$$

$$= \int_{x=-e}^{x=e} a\,\mathrm{d}x \tag{C.32}$$

$$= 2\,e\,a. \tag{C.33}$$

Combining equation (C.30) and (C.33) gives

$$1 = Q(\forall \boldsymbol{x} \in \mathcal{A}\cup\mathcal{B}\cup\mathcal{C}|L) = a\,\sigma\sqrt{2\pi} + 2\,e\,a \tag{C.34}$$

$$\Rightarrow \quad a = \frac{1}{\sigma\sqrt{2\pi}+2\,e}. \tag{C.35}$$

C.5 Wall detection: finding e with the simplifed pdf

Just like in section C.3, we start by calculating $\frac{\partial}{\partial e} \ln Q(\boldsymbol{x}_n|L_k)$ with

$$Q(\boldsymbol{x}_n|L_k) = a(e) \begin{cases} \mathcal{N}(y_n|0,\sigma^2) & ,|x_n| \leqslant e \\ \mathcal{N}\left(\sqrt{(|x_n|-e)^2+y_n^2}\,\middle|\,0,\sigma^2\right) & ,|x_n| > e \end{cases} \tag{C.36}$$

$$= a(e) \begin{cases} A & ,|x_n| \leqslant e \\ B(e) & ,|x_n| > e \end{cases} \tag{C.37}$$

where $a(e)$ denotes that a is a function of e (see equation (C.35)) and A and $B(e)$ were introduced for notational brevity. This gives

$$\frac{\partial}{\partial e} \ln Q(\boldsymbol{x}_n|L_k) = \frac{\frac{\partial}{\partial e} Q(\boldsymbol{x}_n|L_k)}{Q(\boldsymbol{x}_n|L_k)} \tag{C.38}$$

with

$$\frac{\partial}{\partial e} Q(\boldsymbol{x}_n|L_k) = \frac{\partial}{\partial e} a(e) \begin{cases} A & ,|x_n| \leqslant e \\ B(e) & ,|x_n| > e \end{cases} \quad + \quad a(e) \begin{cases} 0 & ,|x_n| \leqslant e \\ \frac{\partial}{\partial e} B(e) & ,|x_n| > e \end{cases}. \tag{C.39}$$

and

$$\begin{cases} \dfrac{\partial}{\partial e} a(e) = \dfrac{-2}{(\sigma\sqrt{2\pi} + 2\,e)^2} \\ \dfrac{\partial}{\partial e} B(e) = \dfrac{|x_n| - e}{\sigma^2}\, B(e) \end{cases}. \tag{C.40}$$

Hence

$$\frac{\partial}{\partial e} \ln Q(\boldsymbol{x}_n|L_k) = \frac{\frac{\partial}{\partial e} a(e)}{a(e)} \quad + \quad \begin{cases} 0 & ,|x_n| \leqslant e \\ \frac{\partial}{\partial e} B(e)/B(e) & ,|x_n| > e \end{cases} \tag{C.41}$$

$$= -\frac{2}{\sigma\sqrt{2\pi} + 2\,e} \quad + \quad \mathcal{P}\left(\frac{|x_n| - e}{\sigma^2}\right) \tag{C.42}$$

with $\mathcal{P}()$ defined as:

$$\mathcal{P}(x) = \begin{cases} 0 & ,x \leqslant 0 \\ x & ,x > 0 \end{cases}. \tag{C.43}$$

This expression for $\frac{\partial}{\partial e} \ln Q(\boldsymbol{x}_n|L_k)$ can be plugged into equation (8.14):

$$0 = \frac{\partial}{\partial e} \ln p(\boldsymbol{X}|L_k) = \sum_n \gamma(z_{nk})\, \frac{\partial}{\partial e} \ln Q(\boldsymbol{x}_n|L_k) \tag{C.44}$$

$$= -\frac{2\,N_k}{\sigma\sqrt{2\pi} + 2\,e} \quad + \quad \sum_n \gamma(z_{nk})\, \mathcal{P}\left(\frac{|x_n| - e}{\sigma^2}\right) \tag{C.45}$$

$$\Rightarrow \quad \frac{2\,N_k}{\sigma\sqrt{2\pi} + 2\,e} \quad = \quad \sum_n \gamma(z_{nk})\, \mathcal{P}\left(\frac{|x_n| - e}{\sigma^2}\right) \tag{C.46}$$

This leads us to the same problem as with the ideal pdf, i.e. that we can't solve the ideal e-equation without iterating. If we would assume that iterations are allowed, then a possible solution would be the following procedure:

1. choose a random value for e
2. collect all points with $|x_n| > e$:

$$\mathcal{S}_E = \left\{ \boldsymbol{x}_n \;\middle|\; |x_n| > e \right\} \tag{C.47}$$

3. solve the equation

$$\frac{2\ N_k}{\sigma\sqrt{2\pi} + 2\ e} \;=\; \sum_{\boldsymbol{x}_n in \mathcal{S}_E} \gamma(z_{nk})\ \frac{|x_n| - e}{\sigma^2} \tag{C.48}$$

   for e.

4. until convergence: return to step 2.

The interesting thing is now that we can omit step 1 and 4, which means only iterating once every EM step, and this without a significant change in behaviour. This is because the EM algorithm works with probability improving steps, but individual steps don't have to maximize the probability.

To conclude this section, we will work out equation (C.48). For this purpose, we define

$$\boldsymbol{\mu}_k^E = \sum_{\boldsymbol{x}_n in \mathcal{S}_E} \gamma(z_{nk})\ |x_n| \tag{C.49}$$

$$N_k^E = \sum_{\boldsymbol{x}_n in \mathcal{S}_E} \gamma(z_{nk}) \tag{C.50}$$

$$\tag{C.51}$$

which simplifies equation (C.48):

$$\frac{2\ N_k}{\sigma\sqrt{2\pi} + 2\ e} \;=\; \frac{\boldsymbol{\mu}_k^E}{\sigma^2} \;-\; \frac{N_k^E}{\sigma^2}\ e \tag{C.52}$$

$$\Rightarrow \quad \frac{N_k^E}{\sigma^2}\ e^2 \;+\; \left(\sqrt{\frac{\pi}{2}} \frac{N_k^E}{\sigma} - \frac{\boldsymbol{\mu}_k^E}{\sigma^2} \right)\ e \;+\; \left(N_k - \sqrt{\frac{\pi}{2}} \frac{\boldsymbol{\mu}_k^E}{\sigma} \right) \;=\; 0 \tag{C.53}$$

This is a simple quadratic equation and has two solutions. To discover which solution is the desired one, we have to look at the simplified version of $\frac{\partial}{\partial e} \ln p(\boldsymbol{X}|L_k)$ which we equate to zero:

$$\left[\frac{\partial}{\partial e} \ln p(\boldsymbol{X}|L_k) \right]_{\text{simplified}} = \; - \frac{2\ N_k}{\sigma\sqrt{2\pi} + 2\ e} \quad + \sum_{\boldsymbol{x}_n in \mathcal{S}_E} \gamma(z_{nk})\ \mathcal{P}\left(\frac{|x_n| - e}{\sigma^2} \right) \tag{C.54}$$

$$= \; - \frac{2\ N_k}{\sigma\sqrt{2\pi} + 2\ e} \;+\; \frac{\boldsymbol{\mu}_k^E}{\sigma^2} \;-\; \frac{N_k^E}{\sigma^2}\ e. \tag{C.55}$$

This function is plotted in Figure C.4. By examining this figure and equation (C.55), we see that this function goes to -∞ at $e = -\sqrt{\frac{\pi}{2}}\sigma$ and at $e \to \infty$. This means that the function will always be positive between the two zeros. This, in turn, means that the solution with the highest e will always be the most probable one because the probability slope between them is always positive.

We conclude that we choose the largest solution of the quadratic equation, which gives us an expression for the optimal e.

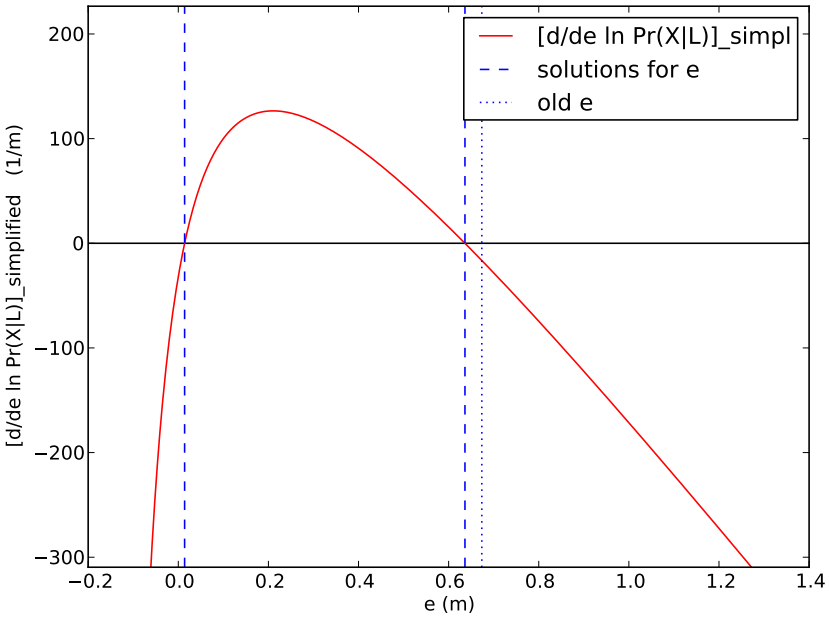


Figure C.4: Plot of simplified version of $\frac{\partial}{\partial e} \ln p(\boldsymbol{X}|L_k)$ in a typical stage of the EM algorithm. The solutions of the quadratic equation are shown as well as the solution for e of the previous iteration.

C.6 Wall detection: calculating gradient for angles

Equation (8.27) defines the optimal $\{\alpha_k\}$:

$$\{\alpha_k\} = \arg\max_{\{\alpha_k\}} p(\{\alpha_k\}|\boldsymbol{X}) = \arg\max_{\{\alpha_k\}} p(\{c_i\}) \cdot \prod_n \sum_k \pi_k Q(\boldsymbol{x}_n|L_k). \tag{C.56}$$

In an attempt to find an expression for the ideal alpha, we derive $\ln p(\{\alpha_k\}|\boldsymbol{X})$ w.r.t. α_k:

$$\frac{\partial}{\partial \alpha_k} \ln p(\{\alpha_k\}|\boldsymbol{X}) = \frac{\partial}{\partial \alpha_k} \ln p(\{c_i\}) + \sum_n \gamma(z_{nk}) \frac{\partial}{\partial \alpha_k} \ln Q(\boldsymbol{x}_n|L_k) \tag{C.57}$$

We start by calculating the first term:

$$\frac{\partial}{\partial \alpha_k} \ln p(\{c_i\}) = \frac{\partial}{\partial \alpha_k} \sum_{c_i} \ln \mathcal{N}\left(\theta_{c_i} \,\middle|\, \frac{\pi}{2}, \sigma_\theta^2\right) \tag{C.58}$$

$$= \frac{\partial}{\partial \alpha_k} \sum_{c_i} -\frac{(\theta_{c_i} - \frac{\pi}{2})^2}{2\,\sigma_\theta^2} \tag{C.59}$$

$$= \frac{-1}{\sigma_\theta^2} \sum_{c_i} (\theta_{c_i} - \frac{\pi}{2}) \frac{\partial \theta_{c_i}}{\partial \alpha_k} \tag{C.60}$$

where θ_{c_i} is linearly dependent on α_k and $\partial \theta_{c_i} / \partial \alpha_k$ is either -1 or 1. Both are easily calculated.

$\partial \ln Q(\boldsymbol{x}_n|L_k)/\partial \alpha_k$ in the second term can be simplified with the chain rule:

$$\frac{\partial}{\partial \alpha_k} \ln Q(\boldsymbol{x}_n|L_k) = \frac{\partial}{\partial \alpha_k} \ln Q(\boldsymbol{x}(\boldsymbol{x}'_n)|L_k) \tag{C.61}$$

(with $\boldsymbol{x}$ and $\boldsymbol{x}'$ defined with respect to the attached and fixed coordinate systems respectively, see Figure C.2)

$$= \nabla_{\boldsymbol{x}} \ln Q(\boldsymbol{x}|L_k) \cdot \frac{\partial}{\partial \alpha_k} \boldsymbol{x}(\boldsymbol{x}'_n) \tag{C.62}$$

with

$$\boldsymbol{x}(\boldsymbol{x}') = \begin{bmatrix} \cos\alpha_k & \sin\alpha_k \\ -\sin\alpha_k & \cos\alpha_k \end{bmatrix} \cdot (\boldsymbol{x}' - M) \tag{C.63}$$

and thus

$$\frac{\partial}{\partial \alpha_k} \boldsymbol{x}(\boldsymbol{x}') = \begin{bmatrix} -\sin\alpha_k & \cos\alpha_k \\ -\cos\alpha_k & -\sin\alpha_k \end{bmatrix} \cdot (\boldsymbol{x}' - M). \tag{C.64}$$

Since we use the simplified pdf, $\ln Q$ becomes

$$\ln Q(\boldsymbol{x}|L_k) = \ln \left(a \begin{cases} \mathcal{N}(y|0,\sigma^2) & ,|x| \leqslant e \\ \sigma\sqrt{2\pi}\,\mathcal{N}(|x|-e \mid 0,\sigma^2)\,\mathcal{N}(y \mid 0,\sigma^2) & ,|x| > e \end{cases} \right) \tag{C.65}$$

$$= \ln a \; + \; \ln \mathcal{N}(y|0,\sigma^2) \; + \; \begin{cases} 0 & ,|x| \leqslant e \\ \ln \sigma\sqrt{2\pi} \; + \; \ln \mathcal{N}(|x|-e \mid 0,\sigma^2) & ,|x| > e \end{cases} \tag{C.66}$$

and thus

$$\frac{\partial}{\partial x} \ln Q(\boldsymbol{x}|L_k) = \begin{cases} 0 & ,|x| \leqslant e \\ -\frac{x-e}{\sigma^2} & ,x > e \\ -\frac{x+e}{\sigma^2} & ,x < e \end{cases} \tag{C.67}$$

and

$$\frac{\partial}{\partial x} \ln Q(\boldsymbol{x}|L_k) = -\frac{y}{\sigma^2} \tag{C.68}$$

These calculations define $\nabla_{\{\alpha_k\}} \ln p(\{L_k\}|\boldsymbol{X})$. If we equate this to zero, it is clear that deriving an exact expression is impossible because of the sum of sines and cosines.

C.7 Visualization: 6 DoF transformation representation

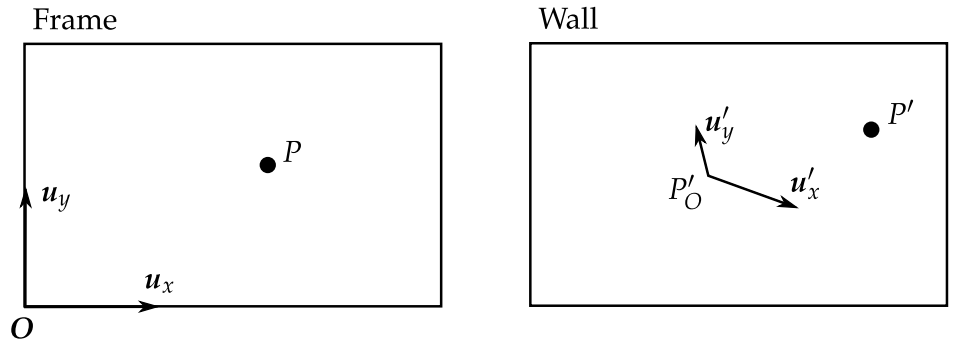


Figure C.5: Transformation of the frame basis $(\boldsymbol{u}_x, \boldsymbol{u}_y)$ and a random point P from a frame to a wall

Following section 9.3.1, we try to represent the 6 DoF transformation as a matrix. The matrix should off course also have 6 independent parameters.

Consider Figure C.5. The basis in the frame with unit vectors $(\boldsymbol{u}_x, \boldsymbol{u}_y)$ and origin $\boldsymbol{O}$ is transformed to the wall in some way. The resulting vectors will not necessarily be perpendicular or equally long. We will now exploit the linearity of our transformation. We can write a random frame point P as

$$P = \alpha \boldsymbol{u}_x + \beta \boldsymbol{u}_y = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \tag{C.69}$$

and, because of linearity:

$$P' - P'_O = \alpha \boldsymbol{u}'_x + \beta \boldsymbol{u}'_y \tag{C.70}$$

$$= \begin{bmatrix} | & | \\ \boldsymbol{u}'_x & \boldsymbol{u}'_y \\ | & | \end{bmatrix} \cdot \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \tag{C.71}$$

which can be summarized as

$$\left[\begin{array}{c} | \\ P' \\ | \\ \hline 1 \end{array}\right] = \left[\begin{array}{cc|c} | & | & | \\ \boldsymbol{u}'_x & \boldsymbol{u}'_y & P'_O \\ | & | & | \\ \hline 0 & 0 & 1 \end{array}\right] \cdot \left[\begin{array}{c} | \\ P \\ | \\ \hline 1 \end{array}\right] \tag{C.72}$$

with six independent parameters in the matrix.

C.8 Visualization: performing linear regression

To help us define the linear regression problem, let us rewrite equation (9.2) from section 9.3.1:

$$P^w = \left[\begin{array}{cc|c} a & c & e \\ b & d & f \end{array}\right] \cdot \left[\begin{array}{c} P^f \\ \hline 1 \end{array}\right]. \tag{C.73}$$

Now, using the following definitions

$$\boldsymbol{x} = \left[\begin{array}{c} P^f \\ \hline 1 \end{array}\right] \tag{C.74}$$

$$\mathsf{W} = \left[\begin{array}{cc} a & b \\ c & d \\ \hline e & f \end{array}\right] \tag{C.75}$$

$$\boldsymbol{y} = P^w \tag{C.76}$$

we rewrite this as

$$\boldsymbol{y}(\boldsymbol{x}) = \mathsf{W}^{\mathrm{T}}\, \boldsymbol{x} \tag{C.77}$$

Our training data consists of a number of couples (P_n^f, P_n^w), which are put into matrices:

$$\mathsf{X} = \left[\begin{array}{c|c} P_1^f & 1 \\ P_2^f & 1 \\ \vdots & \vdots \\ P_N^f & 1 \end{array}\right] \tag{C.78}$$

$$\mathsf{T} = \begin{bmatrix} P_1^w \\ P_2^w \\ \vdots \\ P_N^w \end{bmatrix} \tag{C.79}$$

where we assume all points to be row vectors for notational brevity. The problem is now refined to finding a W that minimizes $|\mathsf{T} - \mathsf{X}\,\mathsf{W}|^2$. This has an exact and simple solution [77]:

$$\mathsf{W} = \left(\mathsf{X}^\mathsf{T}\,\mathsf{X}\right)^{-1} \mathsf{X}^\mathsf{T}\,\mathsf{T} \tag{C.80}$$

C.9 Visualization: multi-room change of basis

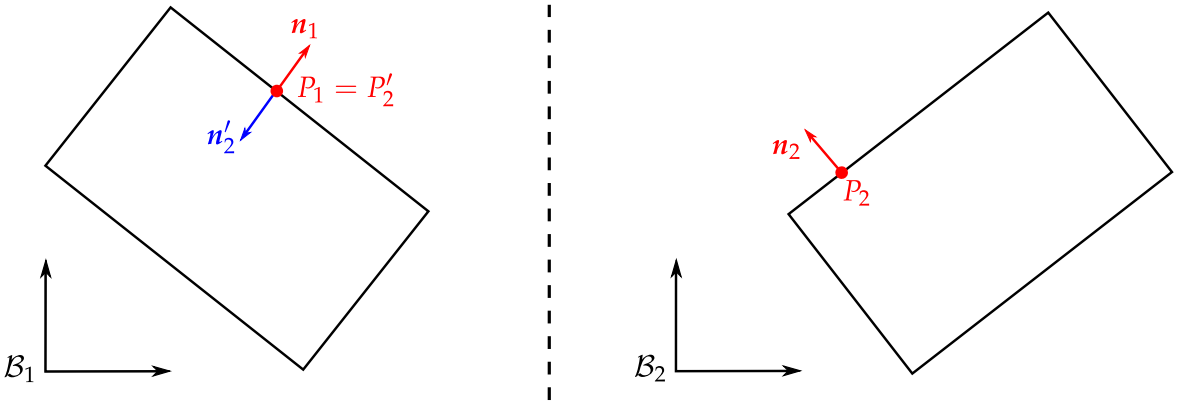


Figure C.6: Multi-room stitch problem.

Consider the situation of section 9.4. If we add normal vectors $\boldsymbol{n}_i$ to both rooms, pointing outward, we get Figure C.6. We want to transform room 2 to base $\mathcal{B}_1$ in a proper way. If we define $\boldsymbol{n}_2'$ and P_2' to be the transformed versions of $\boldsymbol{n}_2$ and P_2, we want that

$$\begin{cases} \boldsymbol{n}_2' = -\boldsymbol{n}_1 \\ P_2' = P_1 \end{cases} \tag{C.81}$$

which defines our change of basis.

We implement this change of basis through three concatenated transformations. First we translate P_2 to the origin, then we rotate room 2 around the origin, and finally we translate

the origin to P_1. Using the same notation as in chapter 5, this gives

$$P|_{\mathcal{B}_1} = \mathsf{Q} \cdot P|_{\mathcal{B}_2} \tag{C.82}$$

$$= \left[\begin{array}{cc|c} 1 & 0 & \\ 0 & 1 & P_1 \\ \hline 0 & 0 & 1 \end{array}\right] \cdot \left[\begin{array}{cc|c} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ \hline 0 & 0 & 1 \end{array}\right] \cdot \left[\begin{array}{cc|c} 1 & 0 & \\ 0 & 1 & -P_2 \\ \hline 0 & 0 & 1 \end{array}\right] \cdot P|_{\mathcal{B}_2} \tag{C.83}$$

with

$$\begin{cases} \cos\theta = \boldsymbol{n}_2 \cdot \boldsymbol{n}_2' \\ \sin\theta = \det\left(\left[\begin{array}{cc} \boldsymbol{n}_2 & \boldsymbol{n}_2' \end{array}\right]\right). \end{cases} \tag{C.84}$$

Experiments confirmed the correctness of these equations.

Publications | D

System Modeling for Active Noise Control with Reservoir Computing

Abstract

This paper investigates the use of reservoir computing for active noise control (ANC). It is shown that the ANC problem can be solved by a concatenation of physically present subsystems. These subsystems can be modelled by reservoirs that are trained, using one shot learning. This approach is compared to genetic algorithms tuning a Volterra filter. Experimental results show that our approach works well as model of the system, meaning that a reservoir trained on white noise performs good on other input signals as well. This is a major advantage over genetic algorithms that perform rather badly on white noise. Furthermore, our approach needs less data and this data can be gathered in one experiment only.

Publications:

- IASTED conference on Signal Processing, Pattern Recognition, and Applications 2012 [78]
- Benelearn conference 2012 [79]

EM-based Wall Fitting in Visually Extracted Noisy Point Clouds for UAV Exploration (submitted)

Abstract

We introduce a robust and efficient technique for room fitting in noisy 2D point clouds. Our method simplifies the exploration problem for UAVs because it estimates the room geometry from visual SLAM feature locations. We use a single camera SLAM implementation, which generates features that roughly indicate the wall positions. To uncover these latent locations, we created a layered model that incorporates a sampling model and prior knowledge about walls. The Expectation Maximization algorithm for Gaussian Mixtures is extended to solve the problem, given this model. We demonstrate our technique on a popular low-cost quadcopter platform and show that it outperforms various alternative techniques.

Publications (submitted):

- 26th Annual Conference on Neural Information Processing Systems 2012 [80]

Bibliography

[1] D. Holz and S. 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)*. IEEE, 2010, pp. 1380–1387.

[2] D. Mellinger, N. Michael, and V. Kumar, "Trajectory generation and control for precise aggressive maneuvers with quadrotors," *The International Journal of Robotics Research*, 2012.

[3] A. Hayes, A. Martinoli, and R. Goodman, "Comparing distributed exploration strategies with simulated and real autonomous robots," in *Proc. of the Fifth Int. Symp. on Distributed Autonomous Robotic Systems DARS-00*, 2000.

[4] I. Rekleitis, G. Dudek, and E. Milios, "Multi-robot collaboration for robust exploration," *Annals of Mathematics and Artificial Intelligence*, vol. 31, no. 1, pp. 7–40, 2001.

[5] P. Biber, H. Andreasson, T. Duckett, and A. Schilling, "3d modeling of indoor environments by a mobile robot with a laser scanner and panoramic camera," in *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings*, vol. 4. IEEE, 2004, pp. 3430–3435.

[6] S. Thrun and J. Leonard, "Simultaneous localization and mapping," in *Springer handbook of robotics*. Springer, 2008, ch. 37, pp. 871–889.

[7] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *Robotics & Automation Magazine, IEEE*, vol. 13, no. 2, pp. 99–110, 2006.

[8] U. Frese, "A discussion of simultaneous localization and mapping," *Autonomous Robots*, vol. 20, no. 1, pp. 25–42, 2006.

[9] D. Hahnel, W. Burgard, D. Fox, and S. Thrun, "An efficient fastslam algorithm for generating maps of large-scale cyclic environments from raw laser range measurements," in *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings*, vol. 1. Ieee, 2003, pp. 206–211.

[10] B. Steux and O. El Hamzaoui, "tinyslam: A slam algorithm in less than 200 lines c-language program," in *11th International Conference on Control Automation Robotics & Vision (ICARCV)*. IEEE, 2010, pp. 1975–1979.

[11] R. Steffen and W. Förstner, "On visual real time mapping for unmanned aerial vehicles," in *21st Congress of the International Society for Photogrammetry and Remote Sensing (ISPRS)*, 2008, pp. 57–62.

[12] M. Blosch, S. Weiss, D. Scaramuzza, and R. Siegwart, "Vision based mav navigation in unknown and unstructured environments," in *IEEE international conference on Robotics and automation (ICRA)*. IEEE, 2010, pp. 21–28.

[13] S. Weiss, M. Achtelik, L. Kneip, D. Scaramuzza, and R. Siegwart, "Intuitive 3d maps for mav terrain exploration and obstacle avoidance," *Journal of Intelligent & Robotic Systems*, vol. 61, no. 1, pp. 473–493, 2011.

[14] A. Doucet, S. Godsill, and C. Andrieu, *On sequential simulation-based methods for Bayesian filtering*. Dept. of Engeneering, University of Cambridge, 1998.

[15] J. Sprickerhof, A. Nüchter, K. Lingemann, and J. Hertzberg, "An explicit loop closing technique for 6d slam," in *4th European Conference on Mobile Robots (ECMR), Mlini/-Dubrovnik, Croatia*, 2009.

[16] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard, "An evaluation of the RGB-D SLAM system," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, St. Paul, MA, USA, May 2012.

[17] A. Nüchter, K. Lingemann, J. Hertzberg, and H. Surmann, "6d slam—3d mapping outdoor environments," *Journal of Field Robotics*, vol. 24, no. 8-9, pp. 699–722, 2007.

[18] N. Engelhard, F. Endres, J. Hess, J. Sturm, and W. Burgard, "Real-time 3d visual slam with a hand-held rgb-d camera," in *Proc. of the RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum*, Vasteras, Sweden, April 2011.

[19] T. Lemaire, C. Berger, I. Jung, and S. Lacroix, "Vision-based slam: Stereo and monocular approaches," *International Journal of Computer Vision*, vol. 74, no. 3, pp. 343–364, 2007.

[20] C. Mei, G. Sibley, M. Cummins, P. Newman, and I. Reid, "A constant time efficient stereo slam system," in *Proceedings of the British Machine Vision Conference (BMVC)*, 2009.

[21] A. Davison, "Real-time simultaneous localisation and mapping with a single camera," in *Ninth IEEE International Conference on Computer Vision. Proceedings*. Ieee, 2003, pp. 1403–1410.

[22] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *ISMAR 2007. 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*. IEEE, 2007, pp. 225–234.

[23] K. Ho and P. Newman, "Loop closure detection in slam by combining visual and spatial appearance," *Robotics and Autonomous Systems*, vol. 54, no. 9, pp. 740–749, 2006.

[24] H. Strasdat, J. Montiel, and A. Davison, "Scale drift-aware large scale monocular slam," in *Proceedings of Robotics: Science and Systems (RSS)*, vol. 2, no. 3, 2010, p. 5.

[25] C. Estrada, J. Neira, and J. Tardós, "Hierarchical slam: Real-time accurate mapping of large environments," *Robotics, IEEE Transactions on*, vol. 21, no. 4, pp. 588–596, 2005.

[26] A. Tayebi and S. McGilvray, "Attitude stabilization of a vtol quadrotor aircraft," *IEEE Transactions on Control Systems Technology*, vol. 14, no. 3, pp. 562–571, 2006.

[27] A. Bachrach, R. He, and N. Roy, "Autonomous flight in unknown indoor environments," *International Journal of Micro Air Vehicles*, vol. 1, no. 4, pp. 217–228, 2009.

[28] S. Grzonka, G. Grisetti, and W. Burgard, "Towards a navigation system for autonomous indoor flying," in *IEEE International Conference on Robotics and Automation (IRCA)*. Ieee, 2009, pp. 2878–2883.

[29] M. Jama and D. Schinstock, "Parallel tracking and mapping for controlling vtol airframe," *Journal of Control Science and Engineering*, vol. 2011, 2011.

[30] S. Thrun, M. Diel, and D. Hähnel, "Scan alignment and 3-d surface modeling with a helicopter platform," in *Field and Service Robotics*. Springer, 2006, pp. 287–297.

[31] W. Morris, I. Dryanovski, and J. Xiao, "3d indoor mapping for micro-uavs using hybrid range finders and multi-volume occupancy grids," in *RSS 2010 workshop on RGB-D: Advanced Reasoning with Depth Cameras, Zaragoza, Spain*, 2010.

[32] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *IEEE International Symposium on Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings*. IEEE, 1997, pp. 146–151.

[33] C. Bills, J. Chen, and A. Saxena, "Autonomous mav flight in indoor environments using single image perspective cues," in *IEEE international conference on Robotics and automation (ICRA)*. IEEE, 2011, pp. 5776–5783.

[34] M. Cummins and P. Newman, "Fab-map: Probabilistic localization and mapping in the space of appearance," *The International Journal of Robotics Research*, vol. 27, no. 6, pp. 647–665, 2008.

[35] R. Hartley, A. Zisserman, and I. ebrary, *Multiple view geometry in computer vision*. Cambridge Univ Press, 2003, vol. 2.

[36] R. Castle, G. Klein, and D. Murray, "Video-rate localization in multiple maps for wearable augmented reality," in *12th IEEE International Symposium on Wearable Computers (ISWC)*. IEEE, 2008, pp. 15–22.

[37] R. Newcombe, S. Lovegrove, and A. Davison, "Dtam: Dense tracking and mapping in real-time," in *IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2011, pp. 2320–2327.

[38] G. Nützi, S. Weiss, D. Scaramuzza, and R. Siegwart, "Fusion of imu and vision for absolute scale estimation in monocular slam," *Journal of Intelligent & Robotic Systems*, vol. 61, no. 1, pp. 287–299, 2011.

[39] J. Sola, A. Monin, M. Devy, and T. Vidal-Calleja, "Fusing monocular information in multicamera slam," *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 958–968, 2008.

[40] M. Kaess and F. Dellaert, "Visual slam with a multi-camera rig," 2006.

[41] M. Kaess and F. Dellaert, "Probabilistic structure matching for visual slam with a multi-camera rig," *Computer Vision and Image Understanding*, vol. 114, no. 2, pp. 286–296, 2010.

[42] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary computation*, vol. 9, no. 2, pp. 159–195, 2001.

[43] N. Hansen, "The cma evolution strategy: a comparing review," *Towards a new evolutionary computation*, pp. 75–102, 2006.

[44] C. Igel, "Neuroevolution for reinforcement learning using evolution strategies," in *The 2003 Congress on Evolutionary Computation (CEC)*, vol. 4. IEEE, 2003, pp. 2588–2595.

[45] Y. Nagata, "The lens design using the cma-es algorithm," in *Genetic and Evolutionary Computation (GECCO)*. Springer, 2004, pp. 1189–1200.

[46] M. Hasenjäger, B. Sendhoff, T. Sonoda, and T. Arima, "Three dimensional evolutionary aerodynamic design optimization with cma-es," in *Proceedings of the 2005 conference on Genetic and evolutionary computation*. ACM, 2005, pp. 2173–2180.

[47] O. Shir and T. Bäck, "Niche radius adaptation in the cma-es niching algorithm," *Parallel Problem Solving from Nature-PPSN IX*, pp. 142–151, 2006.

[48] C. Bishop, "Continuous latent variables," in *Pattern recognition and machine learning*. Springer, 2006, ch. 12, pp. 559–603.

[49] D. Lee, A. Gupta, M. Hebert, and T. Kanade, "Estimating spatial layout of rooms using volumetric reasoning about objects and surfaces," in *Proceedings of the 24th Annual Conference on Neural Information Processing Systems*, 2010.

[50] V. Hedau, D. Hoiem, and D. Forsyth, "Thinking inside the box: Using appearance models and context based on room geometry," *Computer Vision–ECCV 2010*, pp. 224–237, 2010.

[51] S. Yu, H. Zhang, and J. Malik, "Inferring spatial layout from a single image via depth-ordered grouping," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, 2008, pp. 1–7.

[52] C. Lara-Alvarez, L. Romero, J. Flores, and C. Gomez, "A simple sample consensus algorithm to find multiple models," *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, pp. 918–925, 2009.

[53] M. Fischler and R. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[54] E. Vincent and R. Laganiére, "Detecting planar homographies in an image pair," in *Proceedings of the 2nd International Symposium on Image and Signal Processing and Analysis, 2001. ISPA 2001.* IEEE, 2001, pp. 182–187.

[55] Y. Kanazawa and H. Kawakami, "Detection of planar regions with uncalibrated stereo using distributions of feature points," in *British Machine Vision Conference*, vol. 1. Citeseer, 2004, pp. 247–256.

[56] M. Zuliani, C. Kenney, and B. Manjunath, "The multiransac algorithm and its application to detect planar homographies," in *IEEE International Conference on Image Processing, 2005. ICIP 2005.*, vol. 3. IEEE, 2005, pp. III–153.

[57] D. Forsyth and J. Ponce, *Computer vision: a modern approach*. Prentice Hall Professional Technical Reference, 2002.

[58] R. Toldo and A. Fusiello, "Robust multiple structures estimation with J-linkage," *Computer Vision–ECCV 2008*, pp. 537–547, 2008.

[59] P. Hough, "Method and means for recognizing complex patterns," Dec. 18 1962, US Patent 3,069,654.

[60] R. Duda and P. Hart, "Use of the Hough transformation to detect lines and curves in pictures," *Communications of the ACM*, vol. 15, no. 1, pp. 11–15, 1972.

[61] A. Bandera, J. Pérez-Lorenzo, J. Bandera, and F. Sandoval, "Mean shift based clustering of hough domain for fast line segment detection," *Pattern Recognition Letters*, vol. 27, no. 6, pp. 578–586, 2006.

[62] K. Fukunaga and L. Hostetler, "The estimation of the gradient of a density function, with applications in pattern recognition," *IEEE Transactions on Information Theory*, vol. 21, no. 1, pp. 32–40, 1975.

[63] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603–619, 2002.

[64] T. Chin, H. Wang, and D. Suter, "Robust fitting of multiple structures: The statistical learning approach," in *IEEE 12th International Conference on Computer Vision, 2009*. IEEE, 2009, pp. 413–420.

[65] H. Wang, T. Chin, and D. Suter, "Simultaneously fitting and segmenting multiple-structure data with outliers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 99, pp. 1–1, 2011.

[66] H. Isack and Y. Boykov, "Energy-based geometric multi-model fitting," *International Journal of Computer Vision*, vol. 97, no. 2, pp. 123–147, 2012.

[67] C. Bishop, "Mixture models and em," in *Pattern recognition and machine learning*. Springer, 2006, ch. 9, pp. 423–460.

[68] A. Dempster, N. Laird, and D. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 1–38, 1977.

[69] G. McLachlan and T. Krishnan, *The EM algorithm and extensions*. Wiley New York, 1997, vol. 274.

[70] V. Vapnik, S. Golowich, and A. Smola, "Support vector method for function approximation, regression estimation, and signal processing," in *Advances in Neural Information Processing Systems 9*. MIT press, 1997.

[71] V. Vapnik, *The nature of statistical learning theory*. Springer-Verlag New York Inc, 2000.

[72] C. Bishop, "Combining models," in *Pattern recognition and machine learning*. Springer, 2006, ch. 14, pp. 653–676.

[73] I. Sutherland, R. Sproull, and R. Schumacker, "A characterization of ten hidden-surface algorithms," *ACM Computing Surveys (CSUR)*, vol. 6, no. 1, pp. 1–55, 1974.

[74] M. Vandewettering *et al.*, "Point in polygon, one more time.." *Ray Tracing News*, vol. 3, no. 4, 1990.

[75] F. Zhang, "Quaternions and matrices of quaternions," *Linear algebra and its applications*, vol. 251, pp. 21–57, 1997.

[76] J. Kuipers, *Quaternions and rotation sequences*. Princeton university press Princeton, NJ, USA:, 1999.

[77] C. Bishop, "Linear models for regression," in *Pattern recognition and machine learning*. Springer, 2006, ch. 3, pp. 137–177.

[78] J. Nyman, K. Caluwaerts, T. Waegeman, and B. Schrauwen, "System modeling for active noise control with reservoir computing," in *Signal Processing, Pattern Recognition, and Applications 2012, IASTED*. IASTED, 2012.

[79] J. Nyman, K. Caluwaerts, T. Waegeman, and B. Schrauwen, "System modeling for active noise control with reservoir computing," in *Annual Belgian-Dutch Conference on Machine Learning*. Benelearn, 2012.

[80] J. Nyman, K. Caluwaerts, T. Waegeman, and B. Schrauwen, "Em-based wall fitting in visually extracted noisy point clouds for uav exploration (submitted)," in *Proceedings of the 26th Annual Conference on Neural Information Processing Systems*, 2012.