Autonomous Outdoor Navigation with a Robotic R/C Car

Autonome Außenraumnavigation mit einem Modellfahrzeug

René Wagner

Bremen, December 14, 2009

Diplomarbeit am Fachbereich 3 Mathematik und Informatik
Universität Bremen
Betreuung und Erstgutachter: Prof. Dr.-Ing. Udo Frese
Zweitgutachter: Dr.-Ing. Thomas Röfer

Abstract

Imagine a world in which cars drive themselves. Accidents would be a thing of the past because based on a variety of sensor data computers guarantee collision free driving. You would no longer get stuck in a traffic jam – the car chooses the optimal route and coordinates its decisions with other cars. As a side effect this would reduce pollution and noise from stop-and-go traffic. Daily commuters, however, would particularly welcome the newly reclaimed time they can now use to take another quick nap, read the newspaper or check e-mail.

This is the vision of driverless cars – cars that drive themselves. The 2005 DARPA Grand Challenge and the 2007 Urban Challenge race events were important milestones in reaching this goal. Robotic cars traversed a 132 mile desert course completely without human interaction and handled driving in urban traffic including but not limited to the ability of respecting traffic rules, of performing lane changes, merging into traffic, parking and dealing with unforseen events such as road blocks.

This thesis repeats the Grand Challenge experiment on a smaller scale – using an R/C car as the basis of the vehicle and an accordingly down-scaled "race track" consisting of a network of sidewalks or paths in a public park. Based on an existing platform, the SAMSMobil, and drawing inspiration from Stanley, the winning robot of the 2005 Grand Challenge, a complete robotic system is developed which uses sensor data from a planar laser range finder tilted forward, wheel encoders, an inertial measurment unit (IMU) and a Global Positioning System (GPS) receiver to handle the required tasks of localization, mapping, path planning and control.

The localization and mapping performance is evaluated based on log data and the complete system is tested in autonomous outdoor runs during which the robot demonstrates autonomous outdoor navigation abilities although the, relative to the small size of the vehicle, low precision of the available sensors does not allow it to traverse the full test track.

Erklärung

Hiermit versichere ich, René Wagner, diese Diplomarbeit ohne fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen sind, sind als solche kenntlich gemacht.

Unterschrift

Acknowledgements

I would like to thank the following individuals and organizations:

Udo Frese for supporting my idea for this thesis thereby allowing me to work on this exciting topic, the various fruitful discussions we had and the advice he provided. Thomas Röfer for volunteering to be the secondary reviewer.

Jörg Kurlbaum and Oliver Birbach who shared some of the sensor hardware with me. Jörg Kurlbaum particularly for providing some driver code which I use to talk to the MTi-G IMU. Similarly, Holger Täubig for instrumental advice with regard to the communication protocol of the ROD4plus laser range finder. Bernd Gersdorf for advice on the usage of some of the tools in the Cartesium workshop. Christoph Hertzberg for various fruitful discussions and help with some LaTeX oddities. Thomas Röfer and XSens technical support for suggesting ways to debug issues with the MTi-G magnetometer calibration tool. Mattias Werner for providing a Windows XP laptop when it turned out that this was the only way to get the tool to work. Everyone else in the lab for the friendly working atmosphere.

Janosch Machowinski and Holger Täubig for their assistance in outdoor tests with the SAMSMobil. I am particularly grateful for Holger Täubig's assistance in the final outdoor experiment on a Saturday in which the robot demonstrated autonomous navigation abilities.

Furthermore, I am grateful for the hardware the Real-Time Vision Group at University of Bremen and the DFKI research group Safe and Secure Cognitive Systems provided for use as part of this thesis.

Finally, this thesis would not have been possible without getting the low level control of the SAMSMobil to work before the start of the thesis. I would like to thank Holger Täubig for supervising me in this undertaking, Janosch Machowinski and Pierre Willenbrock for recommending the STM32 microcontroller and providing advice on its use, Bernd Gersdorf and Pierre Willenbrock for advice on part of the electronics of the microcontroller board, Udo Frese and Bernd Gersdorf for suggestions on the control approach, and Christoph Lüth for general encouragement and his help with various administrative tasks.

René Wagner

Contents

1 Introduction **1**
1.1 Motivation 1
1.2 Objectives 2
1.3 A Guide to this Thesis 3

2 Related Work **4**

3 General Approach and Contributions of this Thesis **7**

4 The SAMSMobil Hardware Platform **9**
4.1 Existing Hardware Platform 9
4.2 Control Interface 11
4.3 Modifications as Part of this Thesis 11

5 Software Environment and Tools **20**
5.1 Operating System 20
5.2 Robotics Software Frameworks 22
5.3 Interprocess Communication with Isil 23
5.4 Google Earth as a Geo-Spatial Data Editor 24
5.5 Visualization 26
5.6 Simulation 27

6 Sensors **30**
6.1 XSens Inertial Measurement Unit (IMU) 30
6.2 Wheel Encoders 32
6.3 Positioning Based on the Global Positioning System (GPS) 33
6.4 Leuze rotoScan ROD4plus Laser Range Finder 42
6.5 Update Rates and Data Synchronization 42

7 Coordinate Systems and Sensor Calibration **46**
7.1 WGS84 (GPS) Coordinates 46
7.2 Coordinate Systems 47
7.3 Laser Range Finder Calibration 47
7.4 IMU Calibration 48
7.5 IMU and Laser Range Finder Cross-Calibration 51

8 Software Modules and Control Flow **55**
8.1 Overview 55
8.2 Drivers 55
8.3 Pose Estimator 57
8.4 Mapper 58
8.5 Planner 58
8.6 Path Controller 58

8.7 Health Monitor and Guard . . . 58

9 Localization **60**
9.1 State Estimation vs. Localization vs. SLAM . . . 60
9.2 Which Approach on the SAMSMobil? . . . 63
9.3 Bayes Filters . . . 64
9.4 The Kalman Filter . . . 65
9.5 The Unscented Kalman Filter (UKF) . . . 66
9.6 Extension of the UKF to Manifolds . . . 71
9.7 Generic Manifold-UKF Implementation . . . 74
9.8 Manifold-UKF-Based Localization for the SAMSMobil . . . 77

10 Mapping **91**
10.1 Problem Statement . . . 91
10.2 Smooth Coordinates . . . 91
10.3 Interpolation of Poses . . . 92
10.4 Pre-Processing of Laser Range Scans . . . 93
10.5 A Generic 2D Map Container . . . 93
10.6 Probabilistic Terrain Analysis (PTA) . . . 95
10.7 Ground Expectation Mapping (GEM) . . . 101
10.8 Propagation of Obstacle Information . . . 107

11 Path Planning and Control **109**
11.1 Overview . . . 109
11.2 Path Representation . . . 110
11.3 Local Path Planning . . . 111
11.4 Path Control . . . 114

12 Experiments **116**
12.1 Experiments in the Simulation Environment . . . 116
12.2 Outdoor Experiments . . . 117

13 Conclusions and Future Work **143**

Bibliography **145**

List of Figures

4.1 The SAMSMobil in its final pure-SAMS form. **Left:** The PC/104 board and DC/DC converter are mounted on the old battery compartment. The STM32 microcontroller board sits behind the laser range finder. **Right:** During a test run with the cordless gamepad. . . . 10
4.2 **Left:** The frame fits exactly into the upper part of the plastics chassis. **Right:** Of the original upper assembly only the base plate is re-used in the construction of the new frame. 12
4.3 **Left:** The laser range finder is attached to an adapter plate which also holds the STM32 microcontroller board. **Right:** The same unit attached to the frame. . . . 13
4.4 **Top:** The new cabling that connects the components above the base plate with those mounted in the chassis. **Bottom:** The PC/104 stack, solid state disk (SSD) and STM32 microcontroller board are mounted and the e-stop button connected. . . . 14
4.5 **Left:** A custom-built antenna mount allows for attaching the WiFi antenna to the frame and takes care of insulation. **Right:** Side view of the PC/104 stack, the vertically mounted SSD on the base plate, and the STM32 board (USB-A connector). . . . 15
4.6 One of the first test runs with the rebuilt SAMSMobil. Note the battery packs below the laser range finder. . . . 16
4.7 **Left:** With the laser range finder removed the grommets protecting the cables that pass through the base plate and the solid state disk (SSD) right behind them are clearly visible. **Right:** Shorter aluminium profiles have been installed to lower the center of gravity. . . . 17
4.8 **Left:** The stereo-camera/IMU sensor head is mounted for the first time. Note the missing IMU. **Right:** Rear view of the SAMSMobil in its final configuration with all sensors (including the XSens IMU and the new Navilock GPS receiver). . . . 18
4.9 Overview of the IMU/odometry synchronization process. The MTi-G triggers the STM32. IMU and odometry messages are matched by their sequence numbers. . . . 19

5.1 The zig-zag-line representing the goal path (left) and obstacle polygons for the simulation (right). . . . 26
5.2 Visualization of the state estimation and PTA mapping results from an early test run. . 28
5.3 Visualization of the simulated robot pose and laser range scans (left) and additionally with the local map generated by the mapper (right). . . . 29

6.1 Determining range from time of flight in GPS. . . . 35
6.2 Positioning with pseudoranges from three GPS satellites. The pseudoranges determine the radius of each sphere. The positioning solution lies (approximately) where the three spheres intersect. . . . 36
6.3 GPS multipath effects . . . 38
6.4 Screenshot of *u-center* showing NMEA output along with the visualized GPS data. . . . 41
6.5 40 consecutive raw ROD4plus scans showing the intersection with the ground plane in an indoor setting. Note the different scale on the two axes. The outliers are caused by table and chair legs. . . . 43
6.6 Sensor data sources (rectangular) and some of the products derived from them (rounded). 43

7.1 Laser range finder calibration setup . . . 48
7.2 The surface of the calibration object as seen by the laser range finder (100 consecutive scans) . . . 49
7.3 Laser range finder calibration dataset . . . 49
7.4 Screenshot of the 2D calibration report produced by the XSens Magnetic Field Mapper. 51
7.5 A level on top of the laser range finder case (left) and on the IMU/stereocamera head mounts (right). Up to the precision of the level the y-orientation of the two should be identical. . . . 52
7.6 Assuming a pure translation for LIDAR2IMU the simulated range scan (green crosses) does not match the actual range scan (red line). The plot shows the scan plane (both axes in mm). . . . 52
7.7 Actual and simulated range scans for a few samples from a test data set after calibration of LIDAR2IMU. The plot shows the scan plane (all axes in m). . . . 54

8.1 The SAMSMobil outdoor navigation processing pipeline. Note that the STM32 driver appears twice and that UKF Σ is a pose estimator output drawn separately for clarity. . 56

9.1 Bayesian network representing the state estimation problem (Image source: Thrun et al. (2005a)). . . . 61
9.2 Bayesian network representing the localization problem (Image source: Thrun et al. (2005a)). . . . 62
9.3 Bayesian networks representing the online and full SLAM problems. The shaded areas are to be estimated. . . . 63
9.4 Constraint graph used by GraphSLAM (Image source: Thrun et al. (2005a); original annotations removed). . . . 63
9.5 Mahalanobis distance of accelerometer measurements. The rear end of the robot was lifted a few centimeters and dropped several times. The resulting large accelerations violate the "at rest" assumption and are to be filtered out. . . . 84
9.6 The "beams" of the virtual range sensors used to detect path boundaries are illustrated by the four dashed lines. The first obstacle cell intersected by each line determines the range measurement. Note that in the actual implementation the grid has a higher resolution and that the virtual sensor beams are rasterized using the supercover line approach from section 11.3.3 on page 113. . . . 88
9.7 Part of the left hand side path boundary map (thick yellow line segments) for our test track in Google Earth. . . . 88

10.1 Bayesian network representing the problem of mapping with known poses (Image source: Thrun et al. (2005a)). . . . 92
10.2 The local obstacle map is represented as a grid with the robot (large dot) at its center. Points are mapped into the grid through division by the cell side length and integer rounding. In this example the side length of cells is 0.05m. . . . 94
10.3 In a simplistic grid implementation previously entered obstacles (left) need to be moved to retain their relative position if the robot position changes (right). . . . 94
10.4 An illustration of 2D grid wraparounds. The big dot again marks the map center. The regions A, B and C are mapped to A', B' and C' respectively by the wraparound. . . . 95
10.5 Every time a point is entered into the map our PTA update procedure applies the PTA obstacle test to all cells in a 3x3 neighborhood. . . . 97

10.6 Automatic labeling of the grid cell drivability based on log data. The robot drove from the lower-right corner to the upper-left corner in the map. The white stretch is labeled as drivable because it was traversed by the robot. One stretch to the left and one to the right are assumed to correspond with the path and thus marked as non-drivable. All other cells are ignored in the machine learning process. 99
10.7 Left: A lidar beam originating in O touches the curb (dashed line) in P and hits the road below in P'. P and P' are entered in map cells (shaded) more than the 3x3 cell neighborhood away from each other. Right: A cross section of the situation with the sidewalk, the curb, and the road below (from left to right). 100
10.8 The classification of a ramp changes depending on the chosen grid cell size in PTA. The same gradient that is classified as drivable with a small cell size (left) is classified as an obstacle if the cell size is larger (right). 100
10.9 The visualization results that motivated the development of GEM. In front of the robot the range scans stem from hitting the ground on the drivable path. A human observer can easily distinguish range scans originating from positive or negative obstacles from these. 101
10.10 Expected ground plane intersection vs. actual laser range scan endpoints in an indoor setting. The plot shows the scan plane. 102
10.11 Expected ground plane intersection vs. actual laser range scan endpoints in an outdoor setting. The plot shows the scan plane. 102
10.12 Positive obstacles (left) and negative obstacles (right) in GEM. The actual range measured is $r_a = \left|\overline{OP}\right|$ while a range of $r_e = \left|\overline{OP'}\right|$ was expected assuming there is nothing but the ground plane (dashed line). Obviously, anything beyond $min(r_a, r_e)$ must be considered as no longer drivable. 103
10.13 Effect of a single scan (dashed lines) acquired in a corridor scenario on the GEM map. Gray cells are left unchanged. Endpoints classified as HIT add to hits(m_i) as indicated by the red cells which correspond to the corridor walls to the left and right. Endpoints classified as MISS increase misses(m_i) for each cell filled in white which is the case for the endpoints hitting the ground at the top. The two gaps in between the red and white bars are caused by UNDECIDED classifications. 108

11.1 Consecutive application of two path offset functions with different parameters and in opposing directions. 109
11.2 The local obstacle map before (left) and after obstacle extrusion (right). 112
11.3 The tree formed by the application of candidate path offset functions with $r = 0.8$ and $i \in \{0, \ldots, 9\}$ in both directions. Evaluation of candidates ends at a lookahead distance of 4 m. 113
11.4 Valley function with sensed path boundaries at -1 m and 1.5 m relative to the current position perpendicular to the path. 113
11.5 Bresenham line (left) vs. supercover line (right). The Bresenham line is visually pleasant but does not activate some cells although the straight line intersects them. 114

12.1 The obstacle course is defined as a set of polygons in Google Earth. To get past the two solid white obstacles, the planner needs to apply multiple different offset functions. . . 116
12.2 The robot after traversing the obstacle course in simulation. The local map is visualized as a wireframe surface. 117
12.3 A sequence of visualization screenshots (top-left to bottom-right) showing the simulated robot as it succesfully negotiates the two test obstacles. 118
12.4 The main test track in Google Earth 119
12.5 Plot view of the 246m long target trajectory which the robot is instructed to traverse. . 120

12.6 UKF position estimates with GPS and magnetometer measurements disabled. Note that, on the scale of the graph, σ_φ and σ_ϑ overlap as both are dominated by the same accelerometer measurements. 122
12.7 UKF position estimates with magnetometer measurements enabled and GPS disabled. Again, σ_φ and σ_ϑ overlap. 123
12.8 Raw GPS data from several test runs. The obvious quantization steps stem from the limited number of decimal digits transferred through the NMEA interface. 124
12.9 Raw vs. offset-corrected GPS. The offset correction compensates for a constant offset determined at the known initial position. 125
12.10 Raw GPS altitude data from several test runs. 126
12.11 UKF position estimates with GPS measurements enabled. Here, too, σ_φ and σ_ϑ overlap. 127
12.12 Path boundary map. The dashed line represents a disabled path boundary. See text for details. 128
12.13 UKF position estimates with GPS and path boundary measurements enabled. Again, σ_φ and σ_ϑ overlap. 130
12.14 UKF position estimates with GPS projected onto the path. Again, σ_φ and σ_ϑ overlap. . 131
12.15 Global PTA map. 133
12.16 Excerpts from the global PTA map. 134
12.17 More excerpts from the global PTA map. 135
12.18 Global GEM map. 136
12.19 Excerpts from the global GEM map. 137
12.20 More excerpts from the global GEM map. 137
12.21 UKF position estimates and control commands from the 43m autonomous run. 139
12.22 The SAMSMobil under autonomous control by the complete software stack in the 43 m test run. The real robot is shown on the left, its internal state belief and the obstacle map on the right, first shortly after the start of the test track (top), then while initiating a path centering response (middle). and after completing it (bottom). 140
12.23 The SAMSMobil under autonomous control by the complete software stack in the 43 m test run: The bycicle and parking meter which had previously been a problem are passed successfully (top) and the lamp post is also avoided (middle and bottom). 141
12.24 The SAMSMobil under autonomous control by the complete software stack in the 43 m test run: The robot drives nicely centered on the sidewalk (top) but classifies the driveway as an obstacle (middle), veers to the side and blows the motor controller fuse (bottom). 142

List of Tables

6.1 MTi-G performance characteristics in “calibrated data” mode (XSens Technologies B.V., 2008b). 32
6.2 Initial *u-blox 5* configuration. 41
6.3 Revised *u-blox 5* configuration. 42
6.4 Sensors, connection types and update rates. 42

9.1 Square root of the initial covariance diagonal entries. For reasons of numerical stability all 0 entries are actually very small but non-zero values. 90

12.1 Active process and measurement models in the different filter configurations. 121
12.2 Average computation time in μsec for the respective process and measurement models when operating on the log data from Run 2. 129
12.3 The learned PTA parameter set which has been used to generated the PTA maps in this section. 132
12.4 The GEM parameter set which has been used to generated the GEM maps in this section. 134

List of Algorithms

1 ClassifyEndpoints(S_a, S_e) . . . 103
2 MarkAs(c, φ, r_a, r_e) . . . 103
3 SimulateRangeScan(L2W, n_g) . . . 104

4 ApplyOffsetFunction(n, i, r) . . . 111

Chapter 1

Introduction

As part of this thesis a robotic system based on an originally radio-controlled (R/C) car was designed and implemented that is able to autonomously navigate an outdoor path defined as a sequence of waypoints while detecting and (if possible) avoiding obstacles. To reach this goal an existing platform, the SAMSMobil, was equipped with additional sensors and a newly developed software system comprised of components for localization, mapping, planning and control. The robot and the algorithms behind it were evaluated based on outdoor tests during which autonomous outdoor navigation abilities were demonstrated.

1.1 Motivation

The vision behind this work is that of driverless cars – cars that drive themselves. Few pieces of technology shape our daily lives as much as cars. They give us mobility and the freedom to decide where we want to go and when. Cars enable billions of people world wide to live in one place and commute to work somewhere else bridging the gap between rural areas and economic centers where the jobs are. All this does not come without a price – every day people die of car accidents, the ever increasing traffic causes noise and pollution and thus significantly reduces the quality of life particularly in large cities.

Imagine a world in which cars drive themselves. Accidents would be a thing of the past because based on a variety of sensor data computers guarantee collision free driving. You would no longer get stuck in a traffic jam – the car chooses the optimal route and coordinates its decisions with other cars. As a side effect this would reduce pollution and noise from stop-and-go traffic. Daily commuters, however, would particularly welcome the newly reclaimed time they can now use to take another quick nap, read the newspaper or check e-mail.

In aviation this has long been a reality. On virtually all commercial flights today an autopilot takes care of everything but takeoff and landing. Air traffic control is also computerized. Most airliners are equipped with computers that monitor air traffic around them and sound a collision alert to warn the pilots if necessary. In the case of the mid-air collision of two airliners above Überlingen near Lake Constance it is believed that the crash could have been avoided if the pilots involved had followed the instructions issued by their collision warning systems (Anonymous, 2009).

In popular culture driverless cars have been established by the 1980s television series “Knight Rider” featuring the artificial intelligence (AI) controlled car “KITT” where AI refers to the science fiction, strong AI meaning of the term, complete with a notion of ethical behavior, as well as hints at emotions and humor. Up until recently reality could not have been farther away from this since cars have not even been able to drive for extended periods of time without human intervention.

A set of competitions initiated by the US Defense Advanced Research Projects Agency (DARPA), the 2004 and 2005 DARPA Grand Challenge events and the 2007 DARPA Urban Challenge, have changed this. The goal of the 2004 Grand Challenge was for a robotic vehicle to navigate an unrehearsed 142 mile desert course through the Mojave desert defined by a sequence of waypoints without human intervention and complete the course within a time limit that ensures an average speed of at least 20mph. In 2004, none of the contestants finished the race. The 2005 Grand Challenge had basically the same rules except for the different, shorter (132 miles) and slightly easier race course. While in 2004 most teams had their robots follow the provided GPS waypoints more or less blindly, in 2005 the prevailing approach was to solve the complete problem of state estimation, mapping and path planning. Five vehicles finished the race in 2005, the winner was Stanley, a modified Volkswagen Touareg developed by Stanford Racing, a team primarily consisting of Stanford students and researchers and some Volkswagen engineers (Thrun et al., 2006b).

The 2007 Urban Challenge required robotic vehicles to operate in an urban scenario involving fairly dense traffic (simulated by stunt drivers) as well as parking lot driving situations. In addition to the basic autonomous driving and obstacle avoidance abilities seen in the Grand Challenge vehicles thus particularly had to handle moving obstacles, detect and evaluate road signs and traffic lights and respect traffic rules (Montemerlo et al., 2008).

Some of the technology shown in the Grand and Urban Challenge events is slowly making it into applications in the car industry. E.g., Volkswagen recently announced an autonomous prototype vehicle based on the Audi TT which is much closer to a production vehicle, on the outside the only differentiation from an ordinary car are two GPS antennae (Burns, 2009). In the research community it has become fairly quiet since the Urban Challenge. Most research groups with the respective funding and automotive expertise required had already participated in the DARPA events and the entry barrier for others is rather high. Instrumenting a full size car is both financially and in terms of man-power and automotive expertise simply a huge undertaking.

1.2 Objectives

The goal of this thesis is to repeat the Grand Challenge experiment on a smaller scale – using an R/C car as the basis of the vehicle and an accordingly down-scaled "race track" consisting of a network of sidewalks or paths in a public park.

More specifically, an existing R/C car based robotic vehicle, the SAMSMobil, is to be equipped with additional sensors, a GPS receiver, an inertial measurement unit (IMU) and a laser range finder[1]. The combination of odometry, GPS and an IMU for state estimation and laser range finders for environment perception has been the single most popular and successful sensor setup in the Grand Challenge.

On the software side, a set of components for state estimation, mapping, planning and control are to be developed. Based on outdoor tests at a to be chosen test site the performance of the implemented algorithms is to be evaluated. The primary questions to be addressed are whether the Grand Challenge approaches can be transferred to a system comprised of a smaller robot and less precise (more affordable) and whether the success of GPS dominated state estimation techniques and the complete absense of simultaneous localization and mapping (SLAM) approaches in the Grand Challenge indicate that SLAM is in fact not needed for outdoor mobility.

A few constraints need to be respected:

- The hardware platform is to be shared with the DFKI SAMS project.

[1]This will sometimes also be referred to as lidar in analogy to radar.

- It should remain lightweight and small enough to be carried by a single person so that it can be transported to tradeshows and other events with ease.
- Existing sensors are to be re-used. This affects the laser range finder (Leuze ROD4+) and the IMU (XSens MTi-G).

1.3 A Guide to this Thesis

After this introduction, chapter 2 will present some related work. Chapter 3 will introduce the approach chosen for this thesis and summarize the contributions made as part of it.

Chapter 4 then presents the SAMSMobil hardware platform as it existed before the start of this thesis and illustrate the modification that have been applied since. The software side of the platform will be the subject of chapter 5 which will also introduce some existing and some newly developed tools used as part of this thesis.

Chapter 6 introduces the various sensors used as part of this thesis and their theory of operation. It will also discuss the important topic of sensor data synchronization. In chapter 7 the coordinate systems used in the following chapters will be introduced. It also deals with the topic of sensor calibration.

Chapter 8 then gives an overview of the various components of the software stack developed as part of this thesis and illustrates the control flow in the system. Chapters 9 and 10 form the core of this thesis and describe the localization and mapping components as well as the theoretical foundations behind them in detail. Chapter 11 then discusses the path planning and control components.

The evaluation of the complete system based on simulation and outdoor experiments is documented in 12. Finally, chapter 13 will conclude this thesis and provide an outlook towards potential future work.

Chapter 2

Related Work

This thesis touches upon a wide variety of topics within the broad field of robotics ranging from mechanical robot design, to software architectures for robotic systems, state estimation, mapping, planning and control. This section intends to highlight several key research works which can be seen as milestones in the development of driverless cars and autonomous mobile robots over the past few decades. Later chapters will additionally reference individual detail solutions where appropriate.

Pioneering work on driverless cars was done by Dickmanns (e.g., Dickmanns, 1992, 1994) who lead the development of a series of vans and passenger cars which used computer vision for guidance and achieved high speed (>100km/h) autonomous driving including lane changes in highly structured road environments such as the German Autobahn. A key challenge was the limited processing power available at the time. Dickmanns addressed this with active vision techniques and a parallel processing hardware architecture based on transputers. Similar work was done independently in the NavLab project at CMU (e.g., Aubert et al., 1990) and included experiments with early laser range finders (Goto & Stentz, 1987).

Throughout the 1990s and the early 2000s advances in processor design and fabrication meeting or exceeding Moore's law have enabled the development of the field of probabilistic robotics which culminated in the book of the same title by Thrun et al. (2005a). Its origins may be seen in the success of the seminal work by Elfes (1987) and Moravec (1988) on occupancy grid maps which represent the world as a discrete two-dimensional grid containing the probability of cells being occupied.

While early probabilistic approaches still employed comparably unreliable sonar distance sensors for mapping and localization, the widespread availability of planar laser range finders such as the popular SICK LMS series enabled the development of a variety of simultaneous localization and mapping (SLAM) techniques for predominantly planar environments. Examples include the FastSLAM algorithm (Montemerlo et al., 2002, 2003b) which estimates the pose (and trajectory) of a robot relative to a set of discrete features such as trees in an outdoor setting and GMapping (Grisetti et al., 2005, 2006) which does the same based on a grid representation of an indoor office environment or structured outdoor environments dominated by buildings. Wurm et al. (2007) combine feature based and grid based SLAM in a single particle filter algorithm which uses a learned strategy to switch between the two approaches internally. Common to all these approaches is that they rely on scans from a laser range finder mounted parallel to the ground plane and require overlap to match data from consecutive scans as will be discussed in detail in section 9.2 on page 63.

The assumption of a planar environment is dropped in six degrees of freedom (6DOF) SLAM approaches such as that by Nüchter et al. (2004b,a) who perform scan matching on three-dimensional (3D) point clouds consisting of laser scan endpoints. On the Kurt3D robot these are acquired by a planar (2D) range finder that is tilted about a horizontal rotation axis by a servo motor. Each 3D scan thus consists of a set of 2D scans acquired in different range finder orientation angles while the robot is at rest. The downsides of this approach are the need for these intermittent stops of the vehicle and the fairly high computational costs involved in the algorithm.

In parallel to the above localization and mapping approaches planning and control techniques for autonomous navigation have been developed. The size of the planning space and the need to react in real time is a predominant issue in this context leading to approaches that aim to reduce the possible state space, e.g., by dividing the environment into a discrete set of lanes (Ko et al., 1998) or plan directly in the space of (translational and rotational) velocities within a so called dynamic window around the current velocity which is confined by which velocities can be reached given the respective robot dynamics (Fox et al., 1997).

A SLAM approach somewhere in between the ones above is based on a concept called multilevel surface (MLS) maps (R.Triebel et al., 2006; Kümmerle et al., 2008) which represent the environment in a two-dimensional grid as a set of multiple so called surface patches per cell. Each surface patch can either be vertical or horizontal as in modelled as a Gaussian where the mean reflects the vertical position in the cell and the variance the extent of the patch. This compact representation (compared to complete point clouds) makes SLAM techniques based on it much less resource hungry.

The 2004 DARPA Grand Challenge then marked a shift from topic oriented research back to the development of complete robotic systems. The algorithms employed by the 2004 contestants were rather simple and will not be discussed here. This changed in the 2005 edition of the race which is particularly interesting in so far as it highlights which approaches can be implemented within certain time constraints and which work under race conditions. All finalists employed a two stage strategy consisting of separate state estimation and mapping based on the state estimation result. Some teams use the output of commercial units combining GPS and inertial navigation systems (INS) directly as the pose estimate (Leedy et al., 2007), others use similar units which also incorporate odometry information (Trepagnier et al., 2007) or do their own pose estimation based on, e.g., extended Kalman filtering (Daily et al., 2007). When it comes to environment perception, laser range finders where by far the most popular and successful type of sensor in the Grand Challenge. While Trepagnier et al. (2007) placed two planar laser range finders such that the scan planes are oriented vertical to the ground and actively rotated them, most teams used planar laser range finders mounted in various fixed orientations.

The algorithms that powered the winning robot Stanley (Thrun et al., 2006b) were possibly the most advanced ones in the contest. The sensor setup is representative of the competition as a whole: Five roof mounted SICK laser range finders are pointed forward in different, fixed tilt angles and scan the environment up to 25 m ahead of the vehicle. Two radar sensors were also attached to the sensor rack on the roof to provide long range obstacle detection abilities but were never used in the actual race. Instead, the is done based on the video stream from a color camera. For vehicle pose estimation Stanley has a survey grade (L1,L2) Omnistar DGPS receiver, an inertial measurement unit mounted in the trunk and access to the steering angle and wheel velocities through the vehicle's control bus. Stanley's software stack runs as a set of processes on two networked Pentium M based computing nodes mounted in the trunk (a third is available as a backup) using Linux as the operating system and CMU's IPC package (Montemerlo et al. (2003a), see section 5.2 on page 22 for details) for interprocess communication. Pose estimation is done with the help of an unscented Kalman filter (Julier & Uhlmann, 1997) which will be discussed in detail in section 9.5 on page 66. Thrun et al. use two different process models – one for lower speeds based on vehicle odometry and a motion model of the Ackermann drive vehicle, and another one for higher speeds which assumes a sliding mass motion model and is used whenever GPS data is available. Short range mapping is handled by a newly developed algorithm called Probabilistic Terrain Analysis (PTA) which considers vertical distance information in the range finder data to classify terrain as drivable or non-drivable. A set of learned parameters of a Markov error chain model of the perception pipeline allow Stanley to discard data that would otherwise cause "phantom obstacles" to appear in its map. The exact algorithm will be discussed in section 10.6 on page 95. The short range, lidar based classification information is then used in an online learning procedure (Dahlkamp et al., 2006) to determine a classifier that allows a computer vision component to tell drivable from non-drivable pixels in the video stream and thus provide long range obstacle detection. Like all other robots in the race, Stanley gets its goal trajectory as a sequence of GPS waypoints from the organizers. This road description also contains the width of the road and a speed limit for each path segment. Before it is passed to Stanley Thrun et al. run the road description through a smoothing algorithm which aims to find the smoothed path possible within the given road boundaries. Online path planning on Stanley operates in an offset space relative to the goal trajectory and will be discussed in chapter 11 on page 109. The path controller essentially tries to keep the front wheel parallel to the trajectory chosen by the planner.

While SLAM was not employed in the Grand Challenge events, Lamon et al. (2006) later used the MLS map approaches above for outdoor SLAM on a robotic car called Smarter.

The 2007 Urban Challenge saw many evolutionary improvements to approaches that had been used successfully in the 2005 Grand Challenge. Key innovations were driven by the use of a new generation of laser range finders which deliver a dense 3D cloud of endpoints rather than just a planar slice of the environment in a single scan. Popular devices of this type include the ($60k) Velodyne unit used on, e.g., Junior, the new entry by Stanford Racing. Other innovations are related to planning/replanning, the handling of dynamic obstacles and traffic rules as well as cooperative behavior, e.g., when other cars make mistakes in merging situations.

Chapter 3

General Approach and Contributions of this Thesis

Most design decisions in this thesis are driven by the constraints outlined in section 1.2 on page 2 or the available sensors. The main sensor used for environment perception is the ROD4plus laser range finder.

A key issue in outdoor navigation is that unlike in office environments obstacles are not predominantly vertical and usually exceed to the ceiling. Even small obstacles such as a stone no larger than a few centimeters in diameter comprise an untraversable obstacle to the SAMSMobil. Worse yet, negative obstacles such as holes in the ground or curbs approached from the top of a sidewalk pose serious hazards to the small vehicle.

To detect these obstacles the laser range finder is tilted slightly forward. The tilt angle can be seen as a trade-off between the range covered for obstacle avoidance (the earlier the robot knows about an obstacle the easier it is to plan and execute a path that avoids it) and the precision and resolution at which the environment can be scanned, the closer the better. The SAMSMobil is too small to carry an active tilt unit in addition to the already heavy ROD4plus. Thus, for the purposes of this thesis, a fixed tilt angle[1] of about ten degrees is used which means that the laser scan plane hits the ground about two meters ahead of the vehicle.

As will be discussed in detail in section 9.2 on page 63 this rules out simultaneous localization and mapping (SLAM) approaches. Instead, the same two-stage process of state estimation/localization and separate mapping that was so popular in the Grand and Urban Challenge events will also be used in this thesis.

More specifically, the sensor setup basically resembles that of Stanley except that the vehicle is much smaller and only a single scan plane is used compared to the five scan planes on Stanley. Although camera hardware is available on the SAMSMobil, computer vision is not used due to time constraints. Still, Stanley serves as the main inspiration not only because it won the 2005 Grand Challenge and thus must be seen as the state of the art in field robotics but primarily because of the philosophy behind it which Thrun et al. summarize as:

> *"Treat autonomous navigation as a software problem."* (Thrun et al., 2006b).

At the start of this thesis, the low level (PID) control of the SAMSMobil was working. As part of this thesis the hardware platform is first modified to carry the additional sensors. A recurring theme is that of data synchronization of between different sensor data sources. For the wheel encoders and the inertial measurement unit (IMU) a method is developed to do this with the help of a hardware trigger

[1] In fact the tilt angle can be adjusted easily but this requires manually loosening two screws and cannot be done at runtime.

line. For the remaining sensors (ROD4plus and a consumer grade GPS receiver) this is done as well as possible in software.

Apart from the hardware platform, a software environment is established on the SAMSMobil. Debian GNU/Linux is chosen as the operating system, a set of robotics software frameworks are evaluated and a new inter process communication toolkit is developed which forms the basis of the distributed software architecture behind the SAMSMobil software stack. Additionally, an OpenGL based tool specficially designed for visualizing the internal state belief and the obstacle map of the SAMSMobil and a simulation environment are developed.

Before the various sensors are used, their theory of operation is analyzed. GPS is given a special treatment since understanding of its underlying operating principles is essential in the development and evaluation of the state estimation filter. The sensors are also calibrated, in particular a cross calibration procedure of the laser range finder and the IMU pose based on a least squares optimization method is developed and used.

With the hardware and software platform in place, a variety of software modules is developed ranging from drivers to the core modules for state estimation/localization, mapping, planning and control.

A focus of this thesis is the development of a mathematically sound, robust state estimation and localization component which uses an unscented Kalman filter (UKF) to fuse the available, noisy and partly inaccurate sensor data into a state belief that allows the robot to navigate based on it. The original literature on the UKF is revisited with a focus on parameter choices and stability. An existing variant of the UKF algorithm that uses manifolds for the state representation is then extended to also handle manifold measurements. The resulting algorithm is turned into a generic and fast C++ implementation. Based on this a filter consisting of a SAMSMobil specific process model and a variety of measurement models is designed and implemented.

The mapping component is initially a re-implementation of the Probabilistic Terrain Analysis (PTA) approach that powered Stanley but this is discarded because it fails to detect several types of obstacles. Instead, a new algorithm is developed which, based on matches or mismatches with an expected range scan given the pose of the robot and the ground plane, classifies range scans and thereby detects all relevant obstacles in the test track environment, in particular negative obstacles such as curbs which are “missed” by PTA.

The path planner, which is responsible for obstacle avoidance and keeping the vehicle on the sensed path, again follows the approach used on Stanley although a few key parts of the algorithm (particularly the way how candidate trajectories are generated) are newly developed since they are not documented in the respective papers. The path controller is implemented based on an algorithm originally used on the Rolland autonomous wheelchair[2].

In a set of outdoor experiments first log data is collected which is then used to evaluate the state estimation/localization and mapping components. The planner and path controller components are first tested in the simulation environment. The complete system is tested in autonomous outdoor runs during which the robot demonstrates autonomous outdoor navigation abilities although the, relative to the small size of the vehicle, low precision of the available sensors does not allow it to traverse the full test track.

[2] `http://www.informatik.uni-bremen.de/rolland/index_e.htm`

Chapter 4

The SAMSMobil Hardware Platform

This chapter introduces the SAMSMobil robot which served as the hardware platform for the development work and experiments conducted as part of this thesis.

4.1 Existing Hardware Platform

The development of the first version of the SAMSMobil was the subject of a previous Master's thesis (Plasswich, 2008) within the context of the SAMS project. The SAMS project developed a software component that dynamically calculates the shape of an area around a vehicle based on its current translational and rotational velocity and a mathematical vehicle model such that safe, collision free operation of the vehicle is guaranteed if there are no obstacles within that area as reported by a planar laser range finder. This software component was certified for use in safety critical applications (i.e., in environments shared with humans) based on a verification process through formal methods and testing (Frese et al., 2008a). Intended application areas include industrial automation and service robotics.

The idea behind the development of the SAMSMobil was to turn an off-the-shelf R/C car into a robotic platform that could be used to demonstrate the SAMS safety component to lab visitors and at trade shows. The R/C car is based around a differential drive setup of two geared DC motors each of which drives three wheels on the left and three on the right. After the completion of Sven Plasswich's thesis, work on the SAMSMobil continued in order to address a variety of remaining problems with the microcontroller board (defective hardware), the motor controllers, and the control approach.

The microcontroller board was replaced with a new board based on an Olimex STM32-H103 daughter board with a microcontroller from the STM32 family by ST Microelectronics. The STM32 combines an ARM Cortex M3 core with a wide variety of peripherals. The custom-built motherboard adds various connectors, a standard MAX323 based level shifting circuit for a serial (RS232-C) port and protective circuitry for some of the I/O lines.

The microcontroller firmware was rewritten from scratch based on the STM32 firmware library provided by ST Microelectronics . A key part of the new firmware is a communication protocol on top of the STM32's USB interface which allows reliable data transmission through HDLC-like framing and CRC32 checksums. This was instrumental in so far as it enables live monitoring of the code running on the microcontroller from a PC or laptop computer attached via USB. The new communication interface also replaces the error-prone analog link to the R/C receiver which (in combination with the respective transmitter) had served as the user interface and caused much trobule before. Instead, commands now come from a gamepad attached to the host and forwarded to the microcontroller via USB.

Another issue was caused by the motor controllers which deliberately introduce a delay when receiving a reverse command directly after a forward command to protect the attached motor. These were replaced

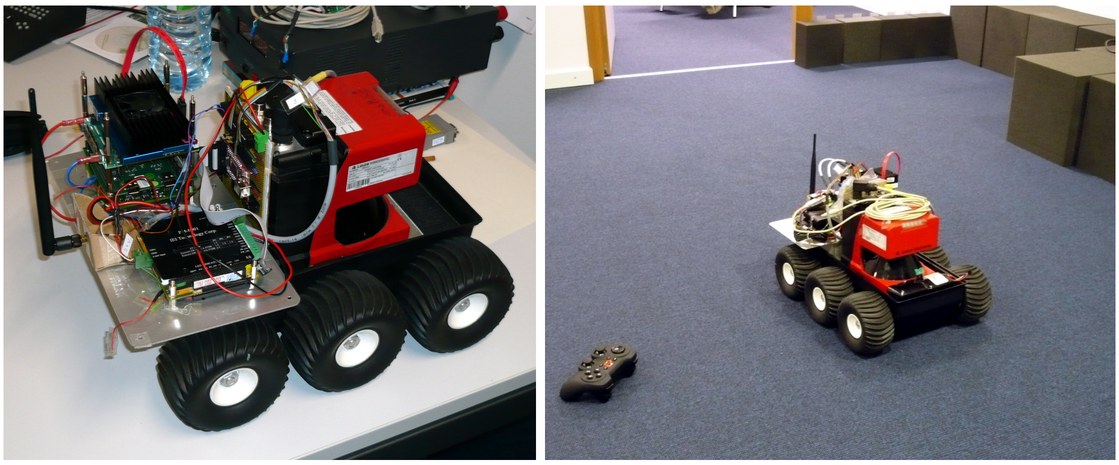


Figure 4.1: The SAMSMobil in its final pure-SAMS form. **Left:** The PC/104 board and DC/DC converter are mounted on the old battery compartment. The STM32 microcontroller board sits behind the laser range finder. **Right:** During a test run with the cordless gamepad.

with new, programmable controllers that allowed the protection features to be turned off such that it became possible to steer the robot in different directions and apply (PID) control techniques. A left turn requires a forward command to be executed by the right hand side controller and a reverse command by that on the left. A direct transition from a left turn to a right turn is only possible if this can be reversed immediately.

The control approach by Plasswich (2008) is kept in general in the sense that the new firmware still separately controls the velocity of each motor via PID control. The implementation, however, was rewritten based on a recursive controller formulation (Bräunl, 2003) with additional counter measures against integrator windup and other causes of overshoots. The control loop runs at fixed intervals defined by a hardware timer. Adequate turning performance proved difficult to achieve primarily due to the high amount of friction caused by the rubber wheels. This issue was eventually solved based on an approach by Lingemann et al. (2005) who propose to modify the control loop by the addition of two feed forward terms such that the control output C is calculated as:

$$C_{l/r} = F_{v,l/r} \cdot v_{set} \mp F_{\omega,l/r} \cdot \omega_{set} + PID(v_{set,l/r}, v_{actual,l/r}) \tag{4.1}$$

The four parameters $F_{v,l}$, $F_{\omega,l}$, $F_{v,r}$ and $F_{\omega,r}$ depend on the vehicle dynamics and the surface characteristics (primarily how much friction is caused). Lingemann et al. (2005) tune these manually. On the SAMSMobil the parameter set is learned from log data acquired with all parameters set to 0 using a least squares optimization approach.

The final version of the SAMSMobil as it was when this thesis started additionally includes a PC/104 board to run the SAMS safety component and code interfacing to the gamepad and the STM32 via USB (Figure 4.1).

4.2 Control Interface

For the purposes of this thesis the SAMSMobil hardware is largely treated as a black box robot providing a standard v-ω-interface to control the translational and rotational velocity of the vehicle.

To this end, the microcontroller expects to receive control commands containing the respective target values. If no new commands or special keep-alive packets (much shorter to save bandwidth) are received within a 100ms timeout the microcontroller assumes the host has run into some form of error condition and immediately turns off both motors.

In the reverse direction the STM32 sends odometry information (wheel velocities, absolute wheel position, c.f. section 6.2 on page 32) along with various statistics on the PID control loop back to the host.

4.3 Modifications as Part of this Thesis

Various modifications have been applied to the SAMSMobil hardware and microcontroller firmware since work on this thesis began.

4.3.1 New Upper Assembly

Probably the most invasive modification is the design and integration of a new upper assembly. The primary goal was to allow the laser range finder to be mounted in the forward-tilted orientation described in chapter 3 on page 7. At the same time it should still be possible to operate the SAMSMobil in a SAMS configuration where the laser scan plane would be parallel to the ground. Thus the upper assembly would need to support a reconfiguration between the two setups with minimum effort. The solution developed as part of this thesis is based on aluminium profiles and hinges by *item industrietechnik GmbH*: A frame was constructed consisting of a base and two central “arms” on each side which hold the laser range finder and are supported from behind by one bar each. Only two screws need to be loosened to adjust the angle of the central arms as is best visible in Figure 4.5 on page 15.

item industrietechnik GmbH was provided with a set of specifications and cut the aluminium profiles to length accordingly. Of the original upper assembly only the base plate was re-used. The new frame has the same outer dimensions and is mounted on top of it and thus fits exactly into the plastics chassis. A photo demonstrating this and another one showing the fully assembled frame can be found in Figure 4.2 on the following page.

To mount the laser range finder a custom adapter plate was built. The range finder is attached to the adapter plate via four screws from behind as illustrated in Figure 4.3 on page 13 (left). The complete unit is then mounted on the frame with four more screws from the front. Additionally, the adapter plate holds the STM32 microcontroller board on its back (Figure 4.3 on page 13, left).

Along with the new mechanical assembly, changes to the electrical wiring have also been implemented. New cables to connect the components above the base plate with those in the chassis were needed (Figure 4.4 on page 14, top). Rubber grommets now protect the cables as they pass through holes in the base plate (Figure 4.7 on page 17, left). Additionally, an emergency stop (e-stop) button (Figure 4.4 on page 14, bottom) now cuts power to the motor circuit relay and thus also cuts power to the motor controllers when pushed. Newly added fuses protect both the motor circuit (25A, slow) and the main circuit (2A, fast).

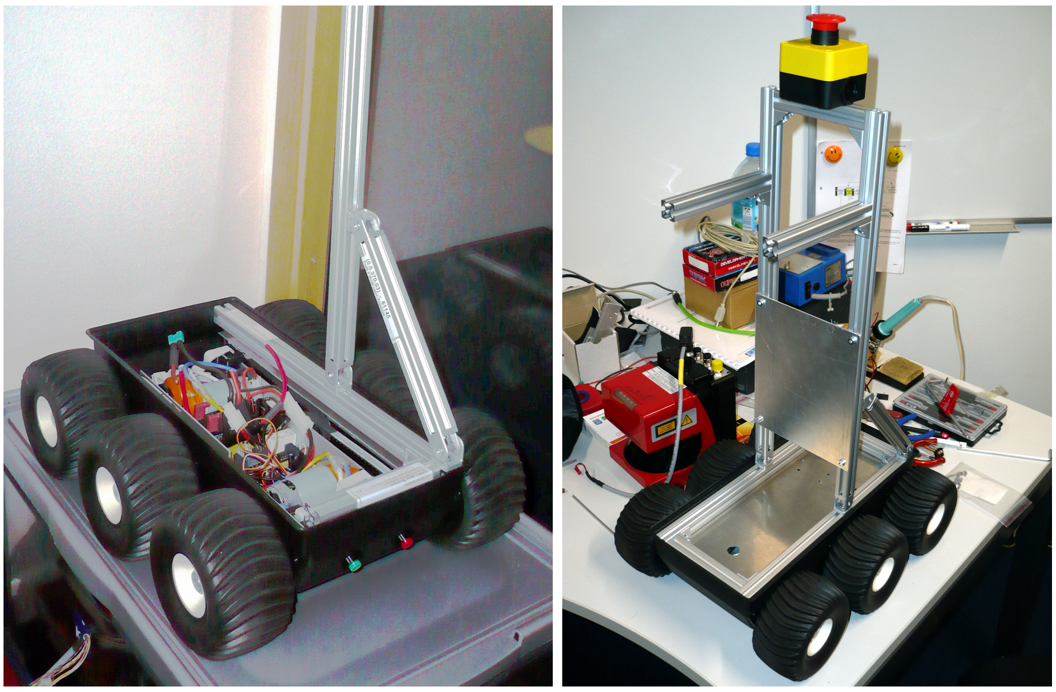


Figure 4.2: **Left:** The frame fits exactly into the upper part of the plastics chassis. **Right:** Of the original upper assembly only the base plate is re-used in the construction of the new frame.

The PC/104 stack now finds its place in between the two support bars at the rear end (Figure 4.4 on page 14, bottom). Unfortunately, the cables to and from the DC/DC converter which needs to sit at the top of the stack for heat dissipation reasons exit its case to the left and to the right when directly attached to the PC/104 stack. To fit the DC/DC converter despite the fact that this is exacly the position of the two support bars another adapter plate was made from sheet aluminium which allows the DC/DC converter to be mounted after a 90° rotation. The power cables now exit at the front and at the back (Figure 4.5 on page 15, right).

The solid state disk (SSD) sits right in front of the PC/104 stack (Figure 4.7 on page 17, left). The old laser range finder ethernet cable turned out to be too short (Figure 4.4 on page 14, bottom) and has been replaced with a newly assembled one. Finally, a custom-built antenna mount made from plastics keeps the WiFi antenna firmly mounted to the frame but also takes care of proper electrical insulation. The rebuilt SAMSMobil with all these changes applied is shown during a test run in Figure 4.6 on page 16.

Although the new upper assembly generally did its job and the SAMSMobil worked flawlessly one issue came up during the initial test runs after the rebuild: The long lever arm of the frame in combination with the already high center of gravity turned out to be problematic for the maneuverability of the vehicle. Abrupt stops were particularly problematic due the risk of the vehicle toppling over. To remedy this the aluminium profiles that form the two arms were replaced by shorter ones as illustrated in Figure 4.7 on page 17.

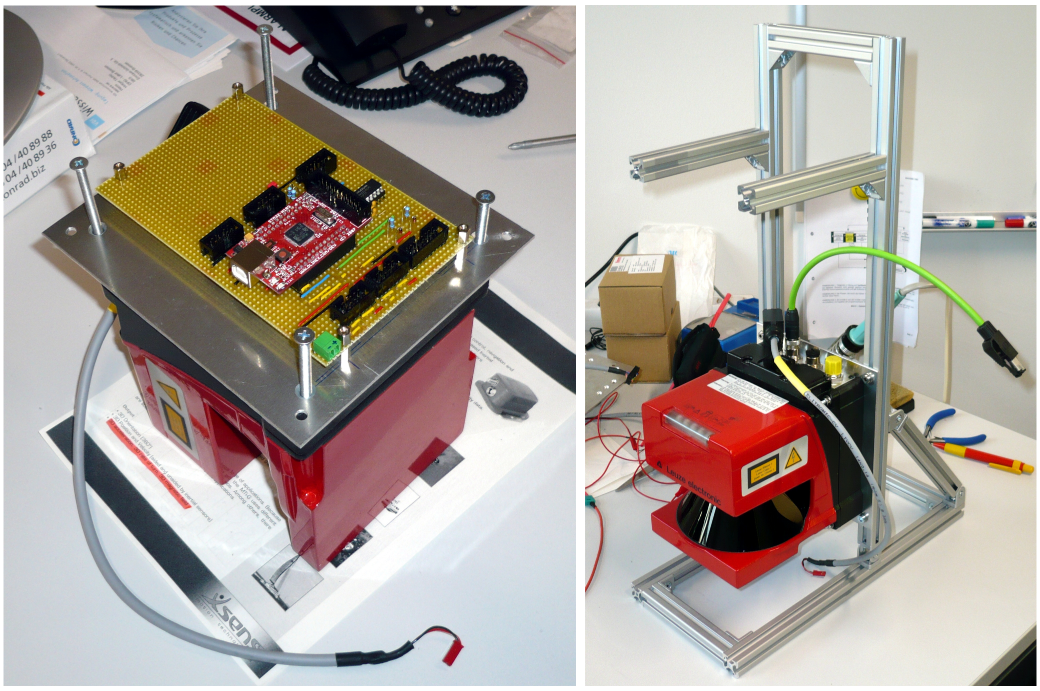
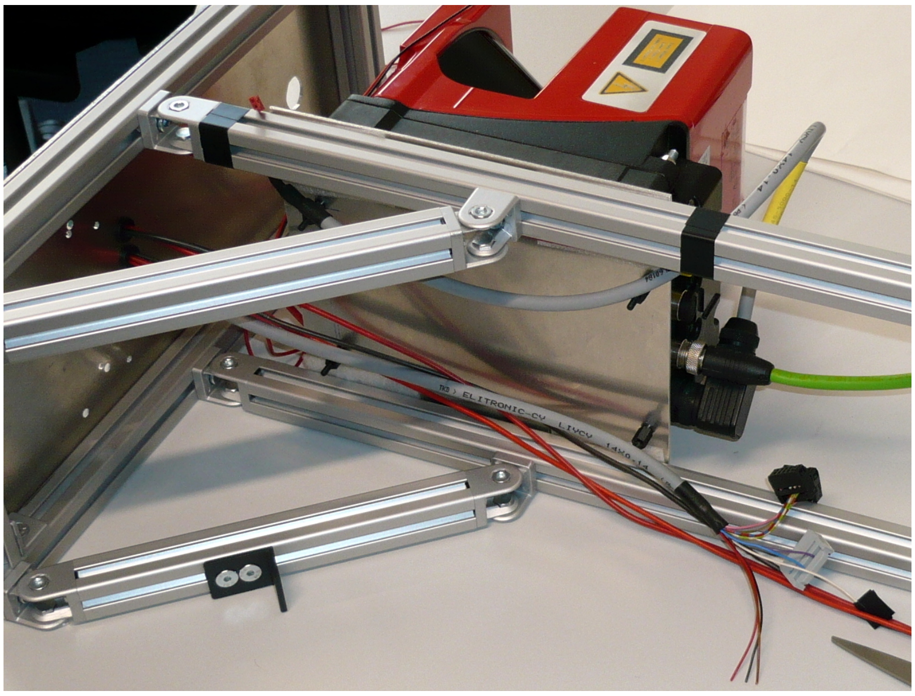


Figure 4.3: **Left:** The laser range finder is attached to an adapter plate which also holds the STM32 microcontroller board. **Right:** The same unit attached to the frame.

4.3.2 Integration of Additional Sensors

The original SAMSMobil only had wheel encoders and the laser range finder. For the purposes of this thesis we additionally need an inertial measurement unit (IMU) and a GPS receiver. The IMU provided for use in this thesis is an XSens MTi-G unit which is part of a stereo camera head which was shared with the SFB/TR 8 Spatial Cognition project A7-[FreePerspective]. Thus the camera head/IMU unit had to be mounted as a whole (to keep the cross calibration intact) and a key requirement was the ability to quickly attach and detach the unit. This was solved by drilling two holes through the base plate of the camera head and attaching it via two angle brackets as can be seen in Figure 4.8 on page 18 (left).

As for the GPS receiver, the original plan was to use the one built into the MTi-G which unfortunately had to be discarded as we will discuss in section 6.3.13 on page 40. Instead, a Navilock NL-402U USB GPS receiver is now used. In either case it is necessary to mount the antenna (or the receiver including the antenna in the case of the Navilock unit) directly on top of a flat surface to avoid multipath effects. Also, the ground should ideally not be “visible” to the antenna. This was addressed (to the extent possible on a small vehicle such as the SAMSMobil) with another sheet of aluminium that is attached to the frame as shown in Figure 4.8 on page 18 (right).



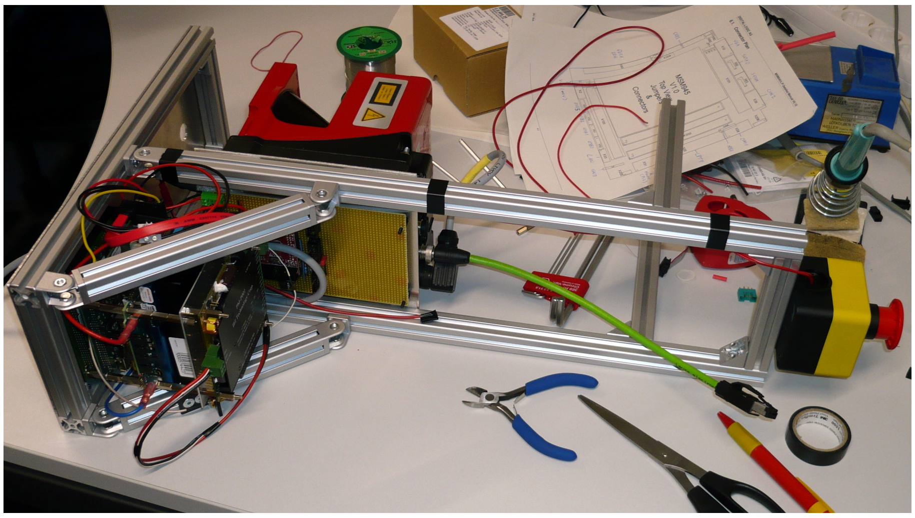


Figure 4.4: **Top:** The new cabling that connects the components above the base plate with those mounted in the chassis. **Bottom:** The PC/104 stack, solid state disk (SSD) and STM32 microcontroller board are mounted and the e-stop button connected.

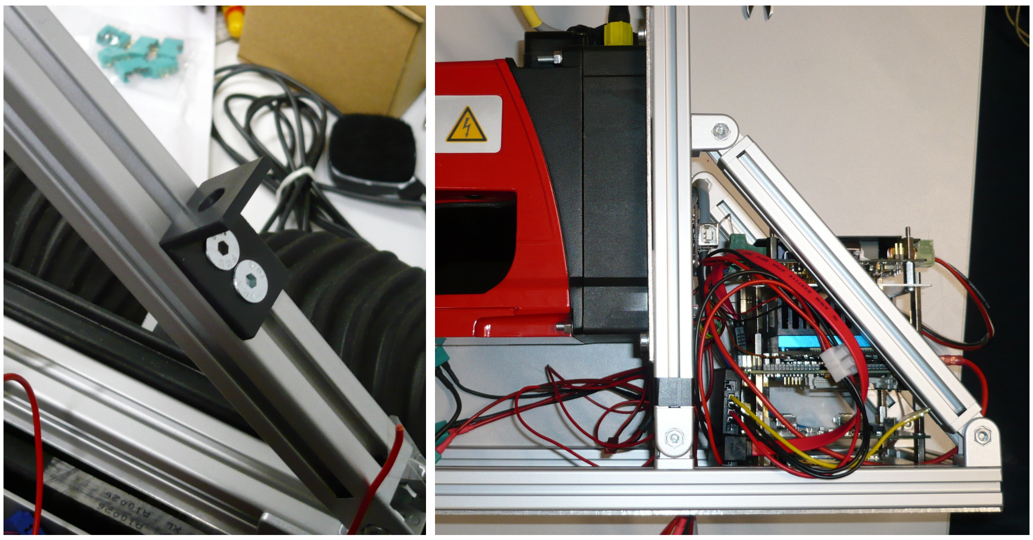


Figure 4.5: **Left:** A custom-built antenna mount allows for attaching the WiFi antenna to the frame and takes care of insulation. **Right:** Side view of the PC/104 stack, the vertically mounted SSD on the base plate, and the STM32 board (USB-A connector).

4.3.3 IMU and Odometry Data Synchronization via Externally Triggered Odometry

The final change to the hardware and firmware consists of the addition of an external trigger line to the STM32 board to allow the MTi-G IMU to trigger odometry measurements. This is necessary to ensure synchronization between the two data streams which we will fuse as part of the state estimation software component (section 9.8.4 on page 81). For such applications the MTi-G can be configured to output a rectangular signal at the same frequency as IMU measurements are recorded (in our case 100Hz). A box with a 3.3V → 5V level shifting circuit and connectors had already been built by Jörg Kurlbaum of the SFB/TR 8 A7 project so that only an adapter cable had to be assembled that connects the level shifting box with an I/O port of the STM32 board.

The STM32 is configured to generate interrupts whenever an appropriate edge on the respective I/O line is detected. The corresponding new interrupt handler, every time it is triggered,

- returns without action if not enough time has passed since the last time it was triggered by means of a hardware timer/counter to filter out noise on the input line,
- otherwise remembers the new timer value,
- reads the current encoder values from the corresponding hardware counters,
- generates a sequence number,
- and queues this data as an odometry message to be sent over USB.

This allows IMU and odometry measurements to be matched by sequence numbers as illustrated in Figure 4.9 on page 19 and replaces the odometry measurements which had previously been generated and sent from the PID control loop.

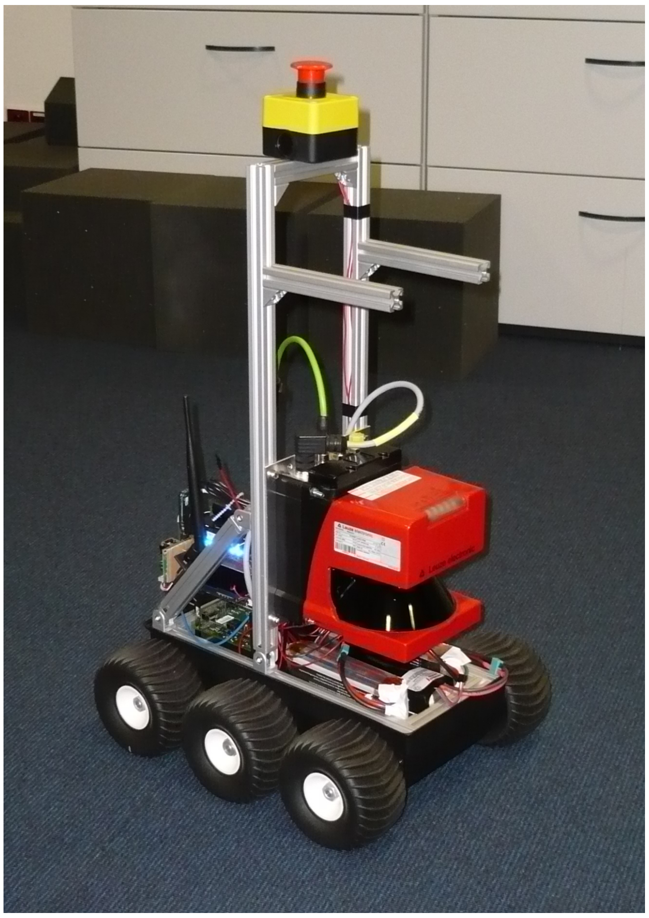


Figure 4.6: One of the first test runs with the rebuilt SAMSMobil. Note the battery packs below the laser range finder.

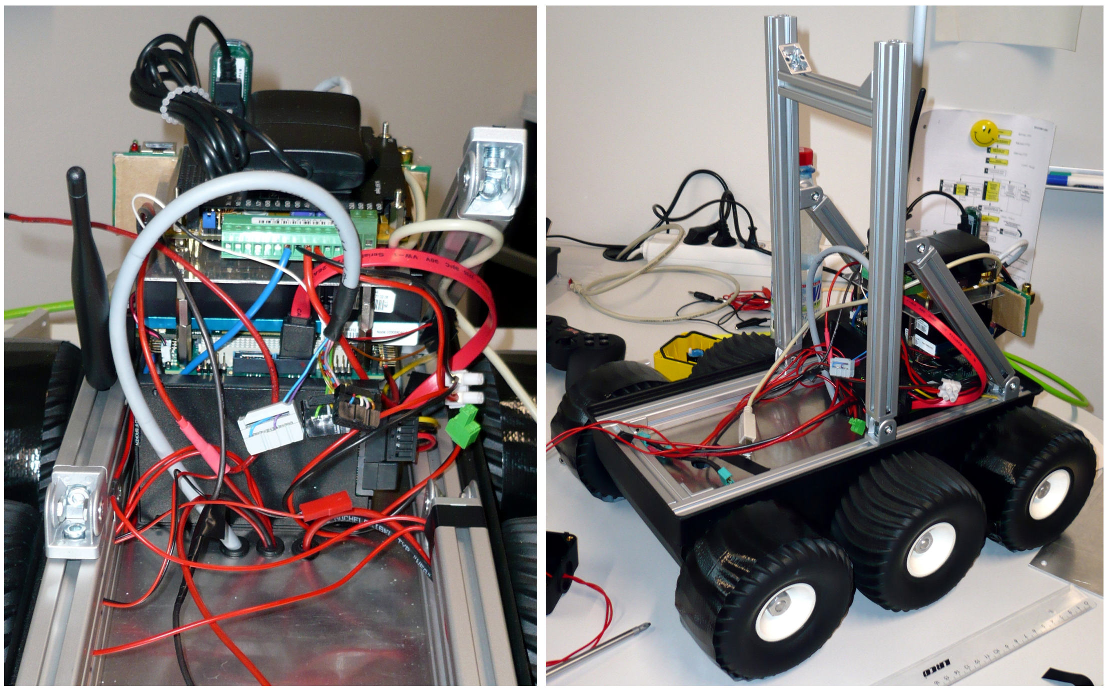


Figure 4.7: **Left:** With the laser range finder removed the grommets protecting the cables that pass through the base plate and the solid state disk (SSD) right behind them are clearly visible. **Right:** Shorter aluminium profiles have been installed to lower the center of gravity.

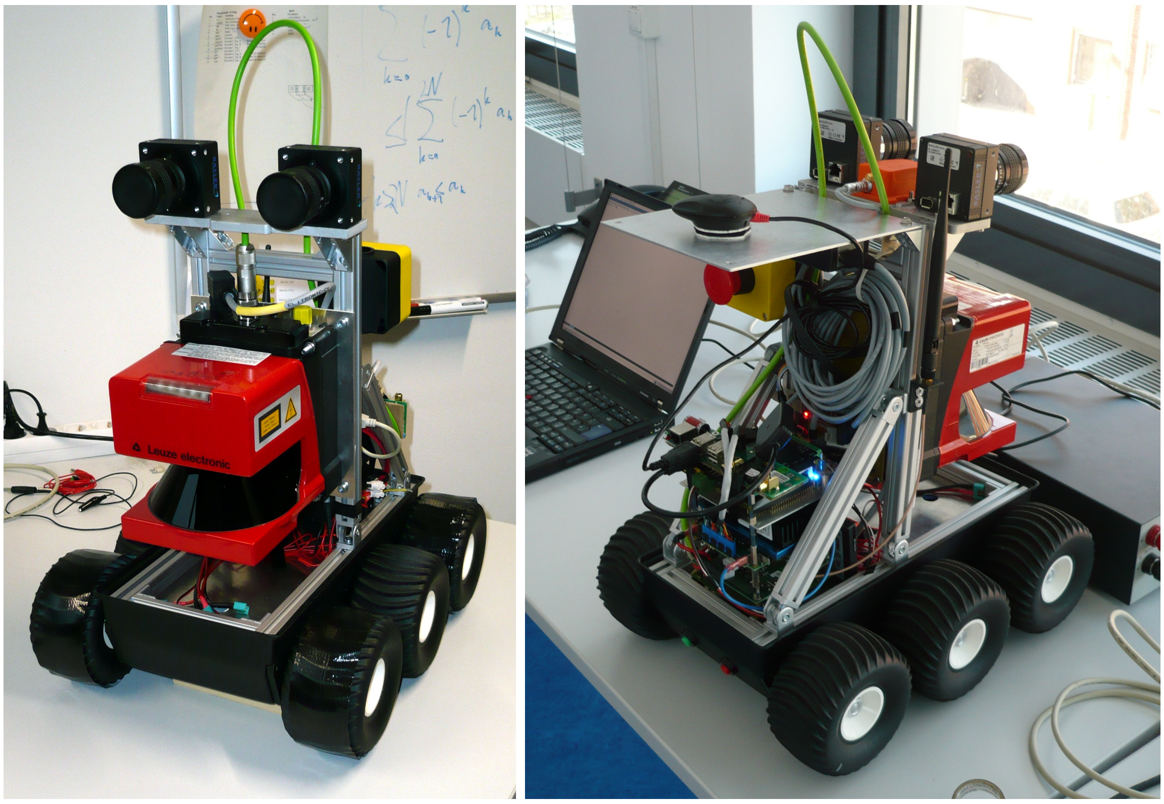


Figure 4.8: **Left:** The stereo-camera/IMU sensor head is mounted for the first time. Note the missing IMU. **Right:** Rear view of the SAMSMobil in its final configuration with all sensors (including the XSens IMU and the new Navilock GPS receiver).

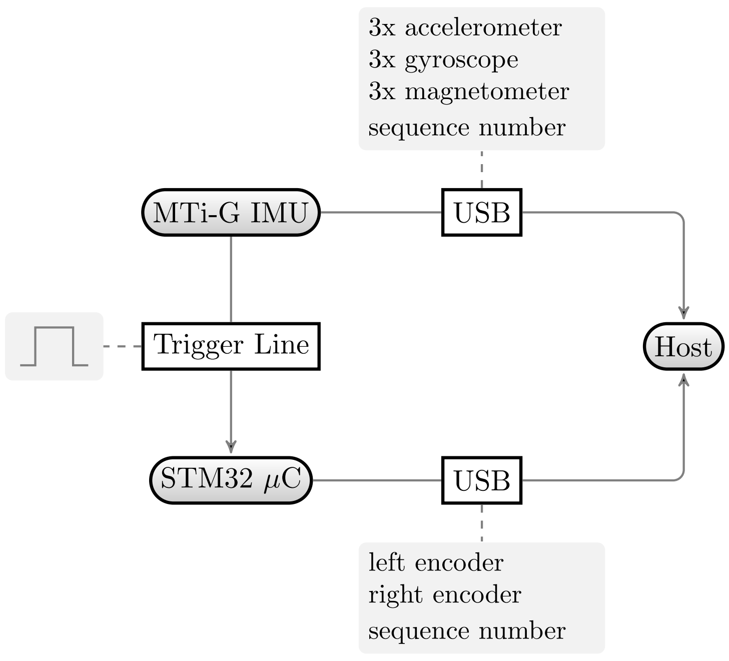


Figure 4.9: Overview of the IMU/odometry synchronization process. The MTi-G triggers the STM32. IMU and odometry messages are matched by their sequence numbers.

Chapter 5

Software Environment and Tools

While the previous chapter discussed the SAMSMobil hardware this chapter will look at the software side of the platform. We will start with the operating system, move on to interprocess communication, and finally discuss the use of Google Earth and newly developed tools for visualization and simulation.

5.1 Operating System

An operating system (OS) for the SAMSMobil had to meet the following requirements:

1. Support headless operation (no monitor/keyboard/mouse).
2. Provide excellent driver support.
3. Provide the option of realtime capabilities.
4. The installed size should be minimal since the SSD is rather small and space is required for logging purposes.
5. Source code availability would be beneficial in the diagnosis of any potential issues.

Although possible through third-party add-ons headless operation with Windows variants is rather cumbersome which narrows candidates down to Unix-like operating systems. Of these Linux offers the best driver support. In this regard it is the only non-Windows operating system that is supported by *Digital Logic*, the manufacturer of the PC/104 board. Linux is well known for its open source nature and also provides some built-in realtime capabilities which can be extended through patches (most notably the -rt patchset).

For the SAMSMobil Debian GNU/Linux 5.0 ("Lenny") was chosen as the Linux distribution because it ships a 2.6.26 Linux kernel which is very close to the "vanilla" kernel.org kernel (leading to fewer surprises due to distribution specific patches), comes with a vast amount of packages, and makes the setup of a minimal system particularly easy via the debootstrap utility. Last but not least the author of this thesis has the most experience with Debian, particularly on embedded systems .

5.1.1 OS-Level Modifications

As hinted at above a minimal Debian system was installed via debootstrap. Apart from the usual setup of network interfaces and other standard settings the following modifications were applied.

SSD Related Issues

The SAMSMobil uses a solid state disk (SSD) for storage which is based on flash memory. Since flash memory only supports a limited number of (erase-) write cycles a number of *tmpfs* RAM-disk-like filesystems are mounted on selected parts of the filesystem tree (most notably /tmp, /var/tmp and parts of /var/cache) to avoid temporary files from hitting the disk. Note that although tmpfs filesystems have a maximum allowed size that can be set at mount time they do not consume memory unless files are written and actually require memory.

To further reduce write access, the system runs without a swap partition. A special flash optimized filesystem for the rest of the system on the other hand is not needed since writes are limited and in contrast to raw flash devices SSDs do their own wear leveling internally.

Continuous System Time

The PC/104 stack on the SAMSMobil lacks an RTC clock battery so that every time it is powered down the system time is lost since the power button cuts power completely. On a Linux system this means that on power-up the system time is set to the start of the Unix epoch (1/1/1970 00:00). This is not a huge problem in itself but it makes log files hard to identify. In particular, timestamps from consecutive tests will in general not be continuous.

To work around this the current system time is dumped to a text file `/var/date` from a `cron` job which runs once every minute. As part of the boot sequence (`/etc/rc.local`) the system time is restored from this text file.

GPS as a Clock Source

The GPS receiver is attached via USB which may cause delays in the delivery of positioning data due to buffering. To prevent old positioning data from being used as (current) measurements in the pose estimator we reject data with a timestamp that is too old as discussed in section 9.8.7 on page 84. This can only work if the system clock is synchronized with GPS time.

This is achieved through a combination of GPSD and ntpd. Ntpd is configured to accept reference clock information through a shared memory segment. GPSD then passes GPS timestamps through this shared memory segment. Both is documented in the `GPSD(8)` manual page. A problem in this setup arises due to the fact that `ntpd` rejects time corrections which exceed a certain limit as an anti-spoofing measure. We avoid this situation (which normally happens every time the system is powered on due to the lack of an RTC battery) by “bootstrapping” the system time with the help of a custom utility called `gpsdate` which reads the current GPS time from GPSD, converts it appropriately and uses it to set the system clock via `settimeofday(2)`.

WiFi

Originally, the wireless card in the SAMSMobil was intended to be used in adhoc mode during outdoor runs (to avoid the need to bring a dedicated wireless access point). It turns out that as of May 2009 adhoc mode support in the Linux kernel is in a particularly bad shape with virtually all wireless drivers if it is supported at all. Instead of searching for a combination of wireless chipsets and drivers that work in adhoc mode the WiFi card was replaced with one based on an Atheros chipset which supports master mode and can thus act as an access point.

Again, the driver situation is far from ideal due to an ongoing transition from the partly binary-only Madwifi driver to a set of new, fully open source drivers. As of May 2009 master mode support in the new drivers is still experimental while the old Madwifi driver is largely unsupported by the respective developers since they want to "encourage" development of the new drivers.

Still the best choice appeared to be to go with the Debian provided source packages of the Madwifi driver. The driver was compiled using the Debian `module-assistant(8)` utility which largely automates the configuration and build process. The author eventually got this to work with `hostapd(8)` which implements the actuall access point (AP) functionality. WPA2 support was intially enabled but had to be turned off to work around connection drops during outdoor tests which do not happen in an unencrypted configuration. Since all communication with the software on the SAMSMobil happens via encrypted SSH connection and key-based authentication this is not a large security issue.

5.2 Robotics Software Frameworks

There appears to be consensus in the robotics community that some form of software framework is useful in splitting the complexity of current robotics systems into smaller, better manageable components. A single gold standard, however, has not been established yet.

As part of this thesis several frameworks have been evaluated with a focus on open source implementations that could be used without modification on the SAMSMobil. Three main categories of toolkits/frameworks can be identified by their use of the following paradigms:

1. Each component lives in its own process and different components communicate with each other by message passing via some form of inter-process communication (IPC).
2. All components are part of a single process consisting of one thread per component where information exchange happends via shared memory and locking.
3. A single process contains all components which are run in a round-robin fashion.

Examples include CMU's Carmen (Montemerlo et al., 2003a) and MIT's LCM (Huang et al., 2009) for the first category, Player/Stage (Collett et al., 2005) for a combination of the second and third, and The MOOS (Newman, 2009) for just the third.

The thrid paradigm relies on some form of scheduler as part of the main process. This task is best left to the existing scheduler of the operating system in use. As for the other two, the author believes it is advantageous to have one process per functional component such that individual components run in separate address spaces and cannot take each other down. This is crucial in the diagnosis of rarely occuring software glitches where one relies on a post-mortem analysis of log data which might not get written to disk if a crash happens in a single process architecture. Similarly, in the development of the SAMSMobil software stack it turned out to be extremely helpful if only a subset of the components can be run in a debugger while the rest runs at regular speed and built with different compiler option (e.g., `-O3` rather than `-O -ggdb`).

Thus, the focus was on finding a preferrably dedicated IPC library. The CMU "IPC" package which CARMEN is based on was considered but eventually discarded, primarily because it consists of a very old, very large and hard to navigate codebase (due to support for a wide variety of partly exotic platform via `#ifdefs`) which has turned out to be a problem in a previous project (RescueRobotics Project, 2007). Also, CMU IPC is available as source code but to the best of the author's knowledge lacks any licensing information. MIT's LCM was discarded because it requires multicast support on a local network which is difficult to provide without separate network infrastructure and because its UDP semantics are problematic since messages are not guaranteed to be delivered which in general requires every component to perform its own error checking.

It should be noted that after the evaluation above was performed a new player in the field of robotics frameworks has gained widespread attention: the Robotics Operating System (ROS) [1]. At a glance, its inter-process communication layer appears similar in design to CMU's IPC and the Isil system developed as part of this thesis. For future projects the author recommends investigating possible integration with ROS to leverage synergy effects in driver and component development since the ROS community appears to be fairly active and growing.

5.3 Interprocess Communication with Isil

Based on the publish-subscribe paradigm made popular by CMU's IPC and Carmen, a new interprocess communication system called Isil was developed for use in this thesis.

Like CMU's IPC, Isil employs a centralized distributed system architecture (c.f. Taylor & Harrison (2009)). Individual components talk to a central message bus server called `isild` over TCP/IP. Messages on different topics are each published to a so called *channel* (simply identified by a unique string). Each subscriber component indicates interest in a certain set of channels to `isild` and, once a message is published on any of those channels, all subscribed components receive the message from `isild`.

This mechanism is fairly straight forward to implement given adequate library support.

- C++ is used in a "Modern C++" programming style (Alexandrescu, 2001) with data structures from the C++ Standard Template Library (STL) and Boost[2].
- Networking support is provided by the asynchronous operations from *Boost.Asio* which are based on the proactor design pattern (Kohlhoff, 2008).
- Serialization is handled by Google's *protobuf* ("Protocol Buffers") library and message description compiler[3].

Google uses protobuf internally to pass messages between a variety of networked systems. It fills a niche where XML is too heavy-weight but a standardized serialization mechanism is desired. protobuf comes with a compiler that turns a C-like message description into message specific C++ classes which can then be used to serialize data into a binary bytestream and deserialize the stream on the remote end.

5.3.1 isild Message Bus Server

`isild` listens for incoming connections on TCP port 2009. The protocol on top of TCP is based on a stream of messages each with a fixed size header that includes information on the size of the following meassage body. The header itself and all messages are encoded using *protobuf*. `isild` processes two general types of request:

- Clients announce their intent to publish data on a channel or request to be subscribed to a channel (identified by its unique string). In both of these cases `isild` adjusts its internal data structures and responds with a channel identifier mapping. This numerical identifier is used in all further communication concerning the respective channel.

[1] `http://www.ros.org`
[2] `http://www.boost.org`
[3] `http://code.google.com/p/protobuf/`

- Message requests contain the actual data payload. The server only checks the header and forwards the body unmodified to all subscribers (if any).

At about 500 lines of code, the code behind `isild` is fairly compact and easy to maintain which establishes the necessary confidence to run it under realtime priority in round robin mode (`SCHED_RR`, c.f. section 6.5 on page 42) to ensure minimum latencies.

5.3.2 Isil Client API

To communicate over the message bus, Isil clients use a header-only client library which handles all interaction with `isild` internally. The API is shown in Listing 5.1 on the next page. Apart from the library clients only need to know the exact format of messages they want to publish or receive. The format is defined by means of `protobuf .proto` files which are compiled to C++ headers. By including the generated header a client gains access to a class representing a message of the respective type.

Publishers must first announce their intent to publish messages by calling `isil_client::async_register ()` and then simply send messages using `isil_client::async_publish()`. Subscribers can directly call `isil_client::async_subscribe()` to register a callback functor which will be called every time new data arrives on the channel. Clients can be both publishers and subscribers at the same time.

5.3.3 Logging and Replay

The logging component `isil-logger` sends a special request to `isild` asking it to forward any and all messages to it which are passed over the message bus. `isild` first responds with possibly existing channel identifier mappings and the latest data in those channels. Then, any newly published messages are forwarded to `isil-logger` which writes the complete stream received from `isild` to disc. The data stream is self descriptive.

`isil-replayer` is the corresponding replay component. It takes a log file generated by `isil-logger` and sends the data contained therein to `isild`. Through optional command line arguments one can influence how fast a log is replayed as a factor that is applied to the interval between (the time stamped) messages in the log, provide a whitelist of channels to be replayed as a regular expression, a blacklist of channels that should be excluded, and finally one can instruct the first k seconds of the log to be skipped.

The command line parameters are typically used to replay logs faster than in real time, include only channels that contain sensor data (as opposed to products such as the state estimate), and skip an initial phase in the log during which the robot is still being prepared for a test run and does not yet move.

5.4 Google Earth as a Geo-Spatial Data Editor

Google Earth supports marking points on the WGS84 ellipsoid in a convenient aerial view. Polylines and polygons can be constructed from individual points and exported using a file format called KML [4]. Based on tutorial code samples by (McCallum, 2005) a custom parser for a subset of this XML format was developed using the Xerces-C++ library[5].

[4]http://www.opengeospatial.org/standards/kml/

[5]http://xml.apache.org/xerces-c/

Listing 5.1: Isil Client API

```cpp
class isil_client
{
public:
  typedef uint32_t channel_id;

  // Constructor. Connects to the server on the given host.
  //
  // The ErrorHandler is called on any (networking) errors and must
  // have the signature:
  //
  //    void (const boost::system::error_code& error)
  //
  template<typename ErrorHandler>
  isil_client(boost::asio::io_service& io_service,
              const std::string &server,
              ErrorHandler h);

  // Subscribe to a channel.
  //
  // Message must be a protobuf type inheriting google::protobuf::Message.
  // Received messages are passed to the MessageHandler functor with the
  // signature:
  //
  //    void (const isil_client::channel_id &channel_id,
  //          const uint64_t &timestamp,
  //          const Message &m)
  //
  template<typename Message,
           typename MessageHandler>
  void async_subscribe(const std::string &channel,
                       const bool &send_latest_datum,
                       MessageHandler handler);

  // Register new channel.
  //
  // The channel_id mapping is passed to the RegisterHandler and can
  // afterwards be used to publish messages.
  //
  //    void (const isil_client::channel_id &channel_id)
  //
  template<typename RegisterHandler>
  void async_register(const std::string &channel,
                      RegisterHandler handler);

  // Publish a protobuf message.
  template<typename Message>
  void async_publish(const channel_id channel,
                     const Message &msg,
                     uint64_t timestamp);

  // Close the connection to isild
  void close();
};
```

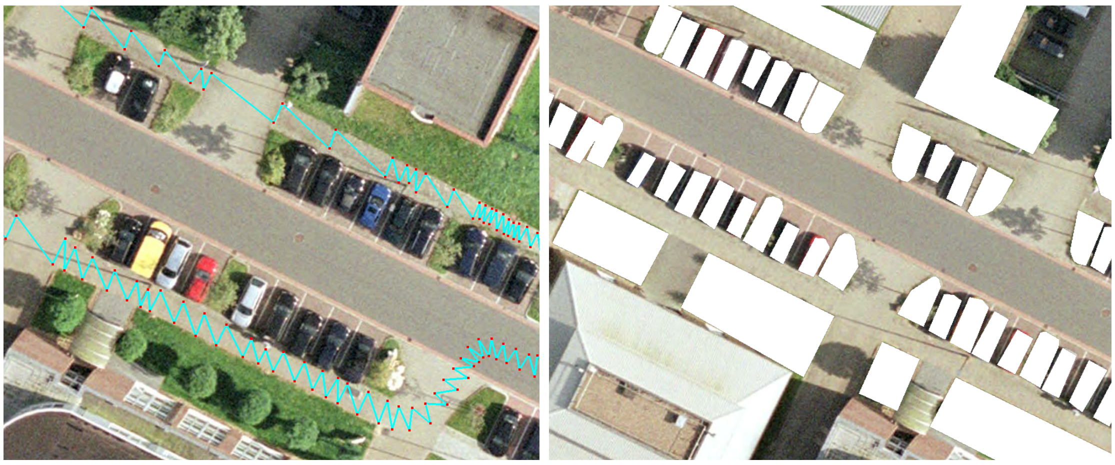


Figure 5.1: The zig-zag-line representing the goal path (left) and obstacle polygons for the simulation (right).

We specify a variety of resources this way:

- The goal path is defined as a zig-zag line (Figure 5.1, left) to encode both the path trajectory and the path width.
- For the simulation polygons define obstacles such as buildings, parked cars, patches of grass and traffic islands (Figure 5.1, right).
- The path boundary map is specified as a set of line segments as will be discussed in section 9.8.10 on page 87.

Similarly, data files have sometimes been generated in the KML format to display geo-spatial data (primarily GPS data and robot trajectories) in Google Earth.

5.5 Visualization

The visualization component serves as a tool to analyze log data or data computed from replayed log data and as a tool to monitor the various hardware and software components on the robot during indoor or outdoor experiments.

It was implemented using the OpenSceneGraph library[6] which provides an abstraction layer on top of the well known OpenGL. Since documentation on OpenSceneGraph is somewhat sparse the examples from the OpenSceneGraph Subversion repository[7] have been used extensively to make this possible. Apart from a 3D model of the SAMSMobil which was created using Blender, the visualization receives all of its input from the Isil message bus.

The basic design of the interface (Figure 5.2 on page 28) is similar to that of many computer games and is based around a main 3D scene and an overlay that mimics a head-up display (HUD). The 3D scene consists of a scene graph comprised of

- the ground plane,

[6] `http://www.openscenegraph.org`
[7] `http://www.openscenegraph.org/projects/osg/browser/OpenSceneGraph/trunk/examples`

- a 3D model of the robot in the estimated pose,
- range scans as a set of red line segments of the respective length,
- the robot trajectory as a polyline,
- the raw GPS position and 2D orientation data indicated by yellow cones and
- height information from the local obstacle map.

Aerial imagery can optionally be overlayed on the ground plane to provide visual ground truth information (Figure 5.3 on page 29). The map tile data is expected to be available in a local directory using the Mercator projection based addressing scheme of, e.g., OpenStreetMap. The visualization environment then computes the address of a map tile with the help of code derived from samples in the OpenStreetMap wiki (OpenStreetMap Project, 2009) and loads the tile image as a texture for the ground plane.

The visualization of the local obstacle map displays for each cell, depending on the mapping algorithm, either the maximum vertical distance of range endpoints (Probabilistic Terrain Analysis) or the absolute maximum z-value (Ground Expectation Mapping).

The HUD visualizes the 3D orientation of the robot on the left hand side of the screen by means of a wire-frame view of the robot projected into the y-z-, x-z- and x-y-planes respectively. The monitoring data displayed on the right hand side of the screen includes (from top to bottom):

- the velocity and distance travelled according to odometry,
- the state estimate (3D position and 2D orientation),
- the latest GPS datum (3D position, HDOP/VDOP, 2D orientation)
- the latest magnetometer readings projected into the plane (2D orientation),
- the first few diagonal entries of the UKF covariance matrix (corresponding to orientation and position uncertainty),
- the distance to closest obstacle (determined by the *guard* component) and finally
- the time since the first datum received.

5.6 Simulation

Simulation environments are an instrumental tool in the development of today's large robotics software systems. Lamon et al. (2006) used a full 3D physics-engine based simulation environment in the development of a complete outdoor navigation system consisting of components for state estimation, mapping, planning and control. We will employ a slightly different approach here. The synthetic IMU and GPS sensor data generated by a simulation environment typically does not exhibit the same error characteristics as real sensor data. Thus, the SAMSMobil state estimation and mapping components were developed based on log data. When it came to the development of the path controller and planner, however, the need for a simulation environment arose as a way to test initial versions of these without putting the robot at risk.

The simulation environment consists of two components:

- A simple planar robot motion model which assumes constant translational and rotational velocity during an update interval is run at 100Hz to determine the robot position and orientation in the plane.

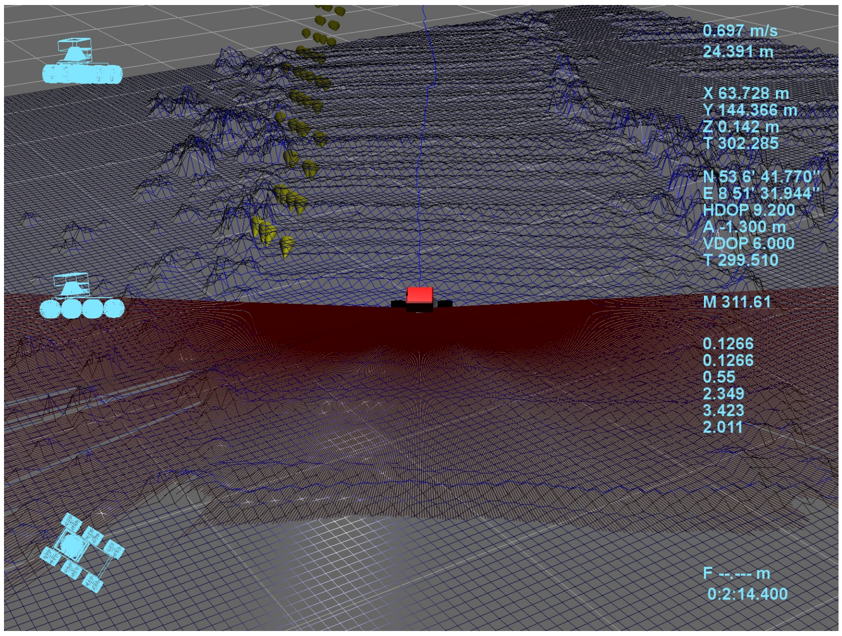


Figure 5.2: Visualization of the state estimation and PTA mapping results from an early test run.

- Based on the simulated robot pose and the pose of the laser range finder relative to it the simulator computes the range at which laser beams would hit an objects to determine simulated range scans. Objects are the ground plane and a set of obstacles specified as polygons in a KML file (c.f. section 5.4 on page 24) and a corresponding height value for each type of obstacle (cars, buildings, etc.). A simple ray tracer then determines the closest range at which a laser beam intersects any of the polyhedra surfaces or the ground plane.

The robot pose and the simulated range scans are published via Isil such that the simulation replaces the STM32 driver, the ROD4plus driver and the pose estimator in the processing pipeline (section 8.1 on page 55). The simulation environment does not produce any visual output on its. Instead, the regular visualization component fulfills this purpose as illustrated in Figure 5.3 on the next page.

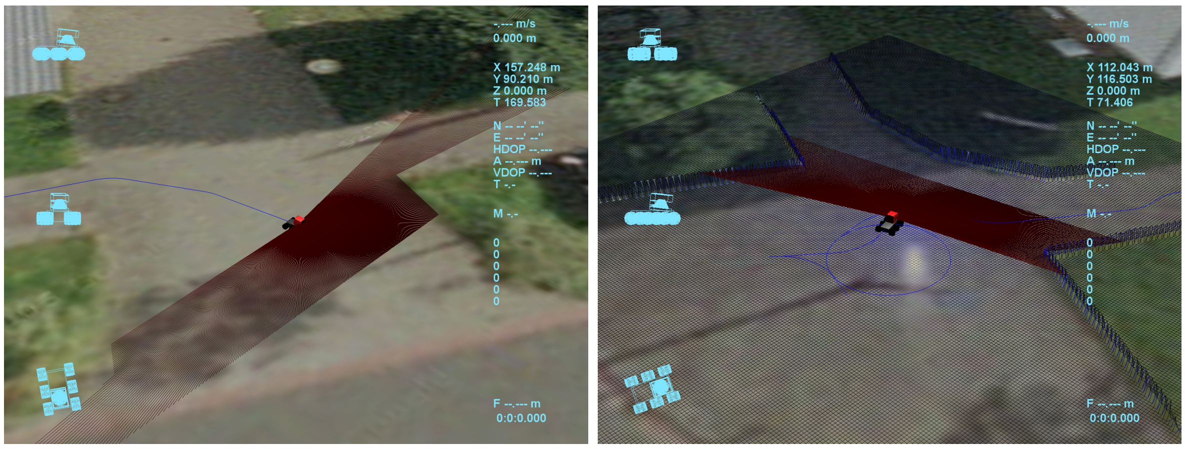


Figure 5.3: Visualization of the simulated robot pose and laser range scans (left) and additionally with the local map generated by the mapper (right).

Chapter 6

Sensors

This chapter introduces the sensors available on the SAMSMobil and how they work. GPS is given a special treatment as knowledge of its inner workings will be instrumental in the analysis of the state estimation performance in section 12.2.2 on page 120. We will close this chapter with an overview of the different update rates of the various sensors and a discussion of the important topic of synchronizing the respective data streams.

6.1 XSens Inertial Measurement Unit (IMU)

The XSens MTi-G combines the following components in a single, fairly compact device:

- accelerometers in 3 axes
- gyroscopes in 3 axes
- magnetometers in 3 axes
- a consumer-grade L1 GPS receiver
- a microcontroller running vendor provided (proprietary) firmware

The GPS receiver capabilities will be discussed in section 6.3.13 on page 40. Here, we will focus on the other sensors and the internal Kalman filter implemented by the XSens firmware.

The exact make/model of the included accelerometer, gyroscope and magnetometer sensors is not publicly documented. The MTi-G manual (XSens Technologies B.V., 2008b) only discloses that the sensors are attached to a 16bit analog to digital converter (ADC) which makes sensor readings available as 16bit unsigned integers. Thus, the sensors clearly output measurements through an analog signal. Together with the relatively low price of the unit this suggests that the sensors are inexpensive analog MEMS sensors (micro-electro-mechanical systems). MEMS technology uses semiconductor manufacturing techniques (etching, etc.) to produce mechanical "machines" at scales of only a fraction of a millimeter (Grewal et al., 2007).

6.1.1 Accelerometers

Accelerometers measure acceleration along a certain body fixed axis. There are several ways to implement MEMS accelerometers. To illustrate the general idea we will use an example from (Grewal et al., 2007, Sec. 9.1.7) who describe a variant where a so called proof mass is attached to a cantilever beam which in turn is attached to the body frame. Acceleration of the body can be sensed by, e.g., a piezoelectric capacitor which changes capacitance as the beam is bent due to the inertia of the proof mass.

The XSens MTi-G provides accelerometer data as one acceleration value in $\frac{m}{s^2}$ for each of the three axes. Although the MTi-G comes calibrated from the factory one can still observe that the accelerometer data is clearly affected by a bias and a significant scaling factor. This can easily be tested since accelerometers do not only measure dynamic acceleration of the body but also the static acceleration due to the Earth's gravitational field. With the sensor at rest and the y-axis pointing straight up or down we expect to measure a static acceleration of around $9.81\frac{m}{s^2}$ or $-9.81\frac{m}{s^2}$ respectively. Instead, one test brought up a mean computed from 30000 samples in each orientation at 100Hz of $9.750233\frac{m}{s^2}$ and $-9.745427\frac{m}{s^2}$ respectively.

The exact values differ for each axis and are not constant across different tests, i.e., it appears that changes in the environmental temperature and supply voltage lead to different bias terms and scaling factors. Both effects may be caused by the MEMS components themselves or by the A/D conversion process which by design relies on a well-known and stable reference voltage.

6.1.2 Gyroscopes

Gyroscopes measure the angular rate about a certain axis of a body. Interestingly, MEMS gyroscopes are actually based on sensing acceleration. This is made possible by the Coriolis effect which results in an oscillating mass fixed to a body being accelerated in correlation to the angular rate of the body. Again, different implementation variants exist; see (Grewal et al., 2007, Sec. 9.1.7) and (Analog Devices, 2004) for examples. For our purposes it is primarily important to note that although sensor manufacturers aim to minimize the problem by measuring acceleration in two orthogonal directions (see (Analog Devices, 2004) for details) acceleration of MEMS gyroscopes, e.g., caused by vibration or other environmental effects, leads to measurement errors due to this very measurement principle.

The MTi-G reports gyroscope measurements as one angular rate value in $\frac{rad}{s}$ per axis. They exhibit a bias in the order of $0.05\frac{rad}{s}$ which can, however, be calibrated on power-up.

6.1.3 Magnetometers

Magnetometers measure the intensity of the magnetic field they are exposed to in the direction of a body fixed axis. MEMS magnetometers deduce this from the *"...Lorenz force generated by a current passing through a magnetic field..."*(King, 2005) and thus also rely on sensing acceleration.

The MTi-G reports the magnetometer readings as a three-dimensional unit-less vector which points in the direction of the surrounding magnetic field. Calibration is essential since the sensed magnetic field can be distorted if there are any nearby ferro-magnetic materials. On the SAMSMobil this is the case as illustrated in, e.g., Figure 4.8 on page 18 (right). The MTi-G is mounted closely above the fairly large metallic case of the ROD4plus laser range finder. We will get back to this along with other calibration efforts in section 7.4 on page 48.

6.1.4 Precision

The XSens MTi-G manual (XSens Technologies B.V., 2008b) lists the performance characteristics summarized in Table 6.1 on the following page. Note that these values apply to the calibrated data output with the vendor-provided calibration parameters already taken into account internally by the MTi-G.

	Angular Rate	Acceleration	Magnetic Field
Full Scale	$\pm 300\frac{\circ}{s}$	$\pm 50\frac{m}{s^2}$	±750mGauss
Linearity	0.1% of FS	0.2% of FS	0.2% of FS
Bias Stability (1σ)	$5\frac{\circ}{s}$	$0.02\frac{m}{s^2}$	0.5mGauss
Scale Factor Stability (1σ)	-	0.05%	0.5%
Alignment Error	0.1°	0.1°	0.1°

Table 6.1: MTi-G performance characteristics in "calibrated data" mode (XSens Technologies B.V., 2008b).

Experience from a previous project (RescueRobotics Project, 2007), which involved ADXL203 MEMS accelerometers (Analog Devices, 2006) and ADXRS150 MEMS gyroscopes (Analog Devices, 2004) both by Analog Devices, suggests that vibration or other sources of large or high-frequency acceleration significantly degrade the actual performance of the sensors. For best performance the IMU should therefore be wrapped in some form of damping material such as rubber foam to decouple the sensor from high-frequency/amplitude acceleration.

On the SAMSMobil this was unfortunately not possible since the stereo-camera/IMU unit had to be attached without modifications as described in section 4.3.2 on page 13. Thus, the MTi-G is rigidly attached to the frame of the upper assembly (c.f. Figure 4.8 on page 18) which has the advantage that its pose relative to the laser range finder does not change due to floating in the damping material but also means that it experiences significant vibration.

6.1.5 Built-in "XSens Kalman Filter"

The MTi-G comes with an internal Kalman filter that estimates the position (in WGS84 coordinates) and orientation (depending on the choice of the user in the form of a rotation matrix, a quaternion or Euler angle representation) of the sensor. Several application scenarios can be selected which enable the fusion of different sensor data sources. Angular rate and acceleration as well as GPS are always taken into account. Optional data source are the magnetometers and a barometric pressure sensor (XSens Technologies B.V., 2008b).

To the best of the author's knowledge the exact filter algorithm is not publicly documented although it appears very likely that an EKF variant is used. Tests with the filter in different configurations quickly showed that the estimated orientation regularly takes 15 minutes or more to stabilize. In combination with the fact that the filter cannot use a vehicle specific process model (in particular to incorporate odometry information) this lead to the decision to discard the idea of using the internal Kalman filter for pose estimation of the SAMSMobil.

6.2 Wheel Encoders

The purpose of wheel encoders is to provide odometry information on the ego-motion of the robot. The SAMSMobil is a differential drive vehicle – each side of the drivetrain consists of a DC motor attached to a gearbox which in turn drives the wheels through a set of toothed transmission belts. The translational and rotational velocity of the vehicle can be determined from the velocity of the left hand side and the right hand side wheels (given knowledge of the axle-base).

The same two HEDL-5640#A13 encoders selected for the original version of the SAMSMobil by Plasswich (2008) are still used to measure those velocities. These optical incremental encoders are based on a code wheel which in the SAMSMobil is attached to a shaft of the transmission belt gearbox which turns at the same velocity as the wheels (ignoring belt slackness, etc.). A combination of LEDs and photo-diodes senses evenly spread spaces in the rotation wheel and thereby turns the wheel rotation into a quadrature-amplitude-modulated (QAM) signal. For the exact theory of operation see the datasheet (Agilent Technologies, 2002) or the thesis by Plasswich (2008).

For the purposes of this thesis it is sufficient to say that the QAM signal is an incremental signal which is decoded by the STM32 microcontroller and accumulated in a hardware counter. The optical encoders have a resolution of 2000 ticks per wheel revolution on the SAMSMobil. Forward and reverse motion can be distinguished in the QAM signal. Forward rotation by $\frac{1}{2000}$ of a wheel revolution increments the hardware counter by one, reverse rotation by the same amount decrements it by one.

The counter value is thus proportional to the distance travelled by a wheel[1]. For this thesis the calibration results by Plasswich (2008) have been re-used to transform encoder ticks into the metric distance travelled which in turn can be used to calculate the current velocity given knowledge of the time between two consecutive measurements. The STM32 firmware ensures that the encoder values are queried at 100Hz as discussed in section 4.3.3 on page 15.

6.3 Positioning Based on the Global Positioning System (GPS)

In recent years GPS based devices have not only become ubiquitous in our everyday lives with applications ranging from in-car navigation systems to location aware services on cellular phones GPS has also established itself as a popular source of positioning information in outdoor robotics systems.

In the robotics literature GPS is typically covered from an application point of view as a black box component “magically” providing absolute position information to a mobile robot. Receivers used range from inexpensive consumer grade devices such as a Garmin handheld receiver (Thrun et al., 2003) to mid range survey grade receivers (Thrun et al., 2006b). The quality of the data they provide is rarely discussed[2].

Regardless of the receiver hardware in use it is, however, important to understand several of the principles behind GPS in order to assess what type of information it can and cannot provide to a robotics system and what must be taken into account if one wants to ensure proper operation of the overall system. The following is primarily due to Xu (2007) who derives the fundamental algorithms from mathematical models of the underlying physical phenomena and presents applications with a bias towards geographic surveying. A comprehensive discussion of GPS within the context of combined GPS and inertial navigation systems (INS) can be found in Grewal et al. (2007).

The following will provide a general introduction to GPS and related systems, illustrate how the position of a GPS receiver is determined, highlight various error sources and strategies to reduce their effect, and finally present the GPS receiver hardware used on the SAMSMobil.

[1]Note, however, that wraparounds in the 16bit counter value need to be taken into account.

[2]A notable exception to this black box approach is research performed in the context of the Stanford Autonomous Helicopter project (`http://heli.stanford.edu/`) where a custom (L1-only) GPS baseband logger has been built (Quigley et al., 2007).

6.3.1 Global Navigation Satellite Systems

GPS is one of several global navigation satellite systems (GNSS). The simplified principle behind any GNSS is that a set of satellites orbiting the Earth broadcast range information which a ground, air, or sea based receiver can use to determine its position using knowledge of the current position of the satellites. The exact procedure is more complex and varies slightly depending on the GNSS used.

GPS was originally conceived by the US Department of Defense under the official name NAVSTAR-GPS to provide positioning and navigation to the US military forces and later opened for civilian use. It is the earliest GNSS with the first satellite being launched in 1978 (Xu, 2007, chap. 1). Russia operates a similar system called GLONASS. There are efforts in the European Union (Galileo), China (COMPASS), and India (IRNSS) to develop competing systems. All systems mentioned differ in the number and orbital position of the satellites, the frequency bands and encodings used for transmission, and the coordinate and time systems employed.

At the time of this writing (2009) affordable receivers are only available for GPS. Some high end ("surveying quality") receivers also support GLONASS in addition to GPS which is advantageous due to an increase in the number of visible satellites at any given point in time and the fact that the distribution of satellites is in a way complementary with GLONASS servicing primarily the northern hemisphere. There are plans to keep Galileo partly compatible to GPS so that some of the more recent consumer grade GPS receivers will be able to process Galileo signals after a firmware upgrade.

6.3.2 GPS Overview

The current GPS satellite constellation is comprised of 24 satellites split over six orbital planes consisting of four satellites each. The satellites orbit the Earth at an altitude of about 26.578km (Xu, 2007, chap. 1). It should be noted that the GPS orbit is not geo-stationary so that the constellation of visible satellites changes over time as previously visible satellites move "below" the horizon and previously invisible satellites become available. This is important because the GPS radio signal is comparably weak making a clear line of sight necessary for reception.

As stated above, GPS assumes that the satellite positions are known so that a receiver can compute its (previously unknown) position. To make this possible a number of ground based control stations constantly monitor the satellite orbits. A complete set of ephemerides (where each satellite is at any given point in time) is referred to as an almanac and transfered to the satellites which broadcast it along with the GPS signal (Xu, 2007, chap. 1).

GPS receivers decode the GPS radio signals and usually determine their position in real time based on either an internal almanac or the one received from a satellite. For some applications the received data is also logged (as raw data) for later post processing (on more powerful hardware) to achieve higher quality position solutions (Xu, 2007, chap. 1).

6.3.3 Time Not Distance

The above statement that GPS is based on range information is not strictly correct since the distance from a satellite to a receiver is not directly observable. Instead, the time of flight of a signal from satellite s to the receiver r can be measured and translated into a geometric distance with the following equation as also illustrated in Figure 6.1 on the facing page.

$$\Delta s = (t_r - t_s)c \tag{6.1}$$

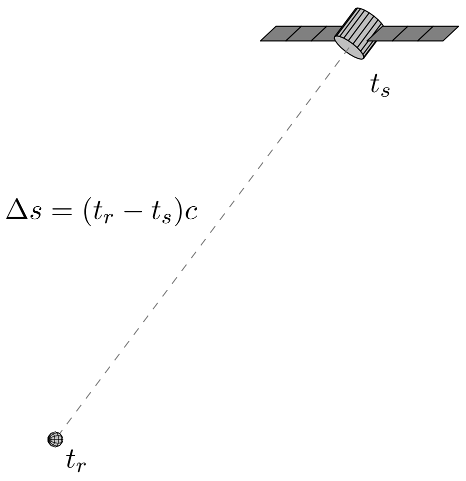


Figure 6.1: Determining range from time of flight in GPS.

t_s and t_r represent the points in time when the signal was transmitted from satellite s and received by receiver r respectively. c is the speed of light in a vacuum. It should be noted that (6.1) is only correct for transmission through a vacuum and does not take errors into account. Consequently, GPS works with so called pseudoranges R_r^s (Xu, 2007, chap. 4).

$$R_r^s = (t_r - t_s)c + \delta_{total} \tag{6.2}$$

(6.2) models errors in the combined error term δ_{total}. We will investigate several error sources in section 6.3.6 on page 37.

6.3.4 GPS Signals

As one would expect with the above in mind, the primary payload of GPS signals consists of the time (in UTC) when the signal was sent by the satellite. GPS currently uses the frequency bands L1 (1575.42 MHz) and L2 (1227.60 MHz) (Grewal et al., 2007, sec. 1.2.1). A third frequency band L5 (1176.45 MHz) is being introduced with the ongoing GPS modernization (Grewal et al., 2007, sec. 3.7).

GPS employs a code division multiple access (CDMA) scheme to enable all satellites to transmit at the same frequency. This is made possible by assigning a unique pseudorandom noise (PRN) code to each satellite. The PRN is a bit sequence which the satellite uses to XOR the actual signal with – for each 0-bit in the data stream it sends the PRN sequence, for each 1-bit the one's complement of the PRN. A receiver can recover the individual signals by XORing the combined signal with each satellite's PRN and applying statistics to distinguish 0-bits, 1-bits from the data stream as well as undecided cases. A detailed discussion of CDMA with examples can be found in (Tanenbaum, 2003, sec. 2.6).

Although GPS is used in a wide variety of civilian applications today, its military origins still show. Only the so called C/A code is available for civilian use (on the L1 band). Use of the encrypted P code requires military receivers with the corresponding decryption keys. The P code is modulated on both the L1 and L2 bands and primarily serves as an anti spoofing measure. Additionally, the so called selective availability (SA) feature adds random errors to the broadcast timestamps to make the C/A code largely unusable to enemy forces. It was disabled in 2000 (Xu, 2007, chap. 5.7) but can be turned back on at any time.

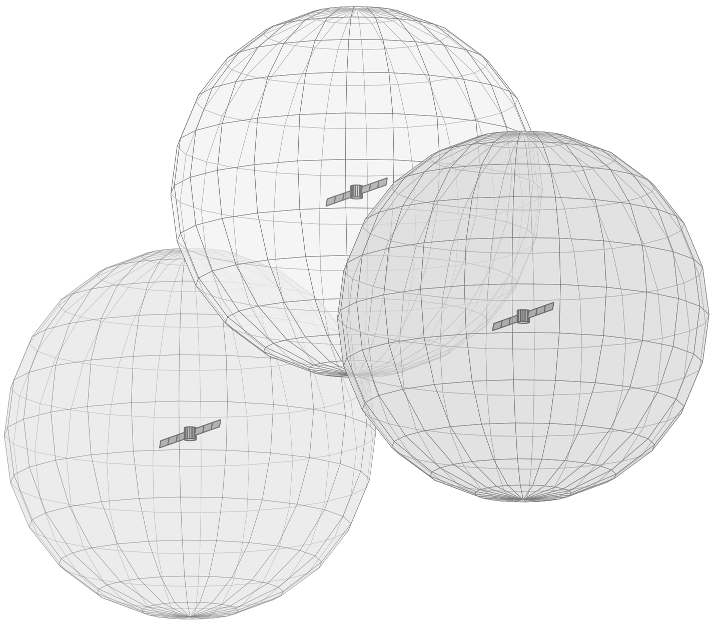


Figure 6.2: Positioning with pseudoranges from three GPS satellites. The pseudoranges determine the radius of each sphere. The positioning solution lies (approximately) where the three spheres intersect.

6.3.5 Determining Position

Given the pseudorange R_r^s to a certain satellite the receiver only knows it is near to the surface of a sphere centered around that satellite with a radius of R_r^s. The reception of signals from multiple satellites allows the receiver to narrow this down further. With the second pseudorange the position solution must be near the circle formed by the intersection of the two corresponding spheres. A third pseudorange narrows the solution down to two points one of which can usually be ruled out taking into account that the receiver is located somewhere in the Earth's atmosphere. An illustration of the three-satellites scenario is depicted in Figure 6.2.

Actual GPS receivers, however, require the simultaneous reception of pseudoranges from at least four satellites which given the above may seem a bit surprising. One reason is that this avoids the need for heuristics to encode prior knowledge of the receiver position. The other is that the redundancy gained from signals of the fourth and all further satellites helps determining receiver internal parameters that model various error sources. In particular, the clock in the receiver is usually not precise to a μs as would be needed for $t_r - t_s$ in (6.1) and the clock bias is estimated along with the position (Xu, 2007, chap. 5.5).

Pseudoranges from additional satellites further improve the position solution. It is, however, not just the number of signals received that influences precision. The geometric configuration of the satellites has a significant effect. In general one can say that the farther apart the satellites used in the positioning calculations are the better the precision of the resulting position estimate will be. GPS receivers pass the expected quality of the geometric configuration on as the dilution of precision (DOP) value – usually a horizontal (HDOP) and a vertical one (VDOP).

Considering that most GPS receivers use Kalman filtering internally (section 6.3.11 on page 39) one might expect that they make information about the uncertainty of the calculated estimate (from the corresponding covariance matrix) available. If this is done at all it is unrelated to the (NMEA) DOP values and consumer grade receivers typically report geometric configuration derived DOP only.

6.3.6 Error Sources

Clock errors have a significant effect on GPS positioning. While the satellites carry atomic clocks the receiver clocks are comparably imprecise and require continuous synchronization. This is one of the reasons why receivers cannot instantly output a position after being turned on.

A second obvious source of errors is the precision of the ephemerides in both the receiver internal almanac and the almanac broadcast by the GPS satellites. The receiver internal alamanac is usually of limited precision unless the receiver supports AGPS (see section 6.3.10 on page 39) and only used to acquire a signal from the first few satellites. The broadcast almanac contains forecast approximations of the satellite orbits and is normally used in all further calculations performed by the receiver. For applications where post-processing is an option the International GNSS Service (previously International GPS Service) provides the so called IGS precise ephemerides which offer significantly better quality and allow for centimeter rather than meter pseudorange precision (Kouba, 2003).

Ionospheric effects also influence GPS. This is because in contrast to the assumption in (6.1) the radio signals do not travel through a vacuum. The resulting error in the pseudorange depends on the exact atmospheric conditions and can, at times, exceed 20 meters (Xu, 2007, chap. 5.1). The atmosphere below the ionosphere (in the GPS literature historically referred to as troposphere) induces additional errors which are at up to a few meters, however, less dramatic (Xu, 2007, chap. 5.2). Both ionospheric and tropospheric effects can be assumed to be reasonably constant in a locally constrained area over a short period of time.

By far the most problematic GPS errors are caused by multipath effects. Much like light is reflected by a mirror, the GPS radio signals are reflected by surfaces of buildings or any other obstacles. Thus there is no longer just one path on which a signal from a satellite can reach a receiver – there are multiple paths as illustrated in Figure 6.3 on the next page. This has two main consequences: If the reflected (slightly delayed, possibly phase shifted) signals reach the receiver they cause electromagnetic interference with the original radio signal and with each other, i.e., it may no longer be possible to decode the signal. Secondly, if there is an additional obstacle in the direct path to the receiver (as is often the case in "urban canyons") a reflected signal may be mistaken for the original one – the best thing the receiver can do is pick the first signal received as the "correct" one. Multipath effects change as the receiver, the satellites, or obstacles move their position and may according to (Xu, 2007, chap. 5.6) result in errors of theoretically up to 150m (15m if the non-civilian P code signal is used).

Other sources of errors in GPS include relativistic effects, antenna design, and instrumental biases, which need to be taken into account in the receiver design but have little practical influence, as well as Earth tide and ocean loading tide effects which are only relevant in applications that require extreme precision (Xu, 2007, chap. 5).

6.3.7 Improving Precision through Carrier-Phase Detection

While consumer grade GPS receivers base all calculations on code pseudoranges only, higher end devices also detect carrier phases which can be measured up to a precision of 1% of the carrier wavelength (Xu, 2007, chap. 4.2).

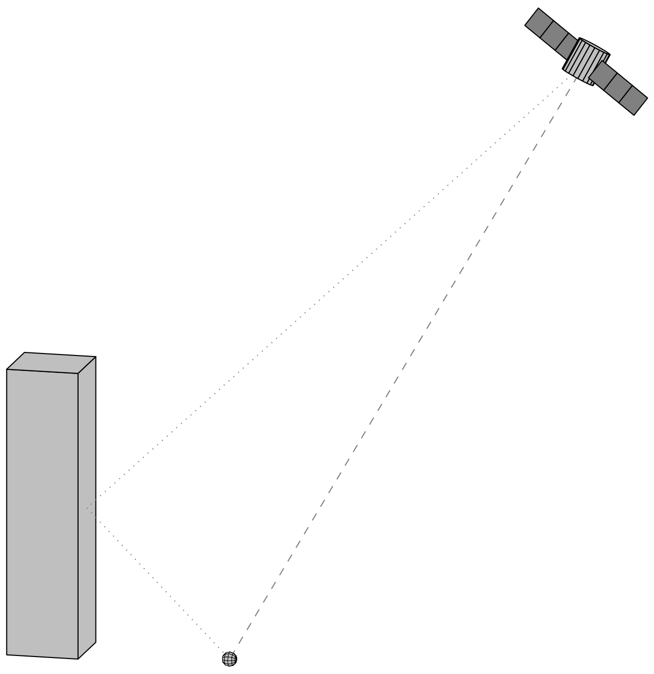


Figure 6.3: GPS multipath effects

Another feature provided by high end receivers exploits the fact that as a dispersive medium the ionosphere affects radio signals travelling through it differently depending on their frequency. The simultaneous use of multiple frequency bands thus allows for an indirect measurement of ionospheric effects. If receiver hardware for two frequency bands is available (usually L1 and L2; the content on L2 is encrypted, but the signal is available) ionospheric effects can be eliminated up to a first order approximation. With three different frequency bands up to the second order (Xu, 2007, chap. 5.1).

6.3.8 Doppler Observables

Like any other wave, GPS radio signals are subject to the Doppler effect which causes a frequency shift if sender and receiver are in motion relative to each other (Xu, 2007, chap. 4.3). This is the same effect that pedestrians notice every day: an oncoming car sounds different from one that moves away from the observer. Applied to a signal from a single satellite the Doppler observable is a measure of the change of the range to that satellite. Given multiple measurements and knowledge of the satellite motion the velocity of the receiver can be deduced.

Doppler observables are a by-product of carrier phase detection and thus unfortunately not available in consumer grade GPS receivers. An interesting property of the Doppler effect is that it is unaffected by ionospheric and other low frequency effects (Xu, 2007, chap. 4.3). However, it is only precise at receiver velocities above $2\frac{m}{s}$.

6.3.9 Differential GPS (DGPS)

The general idea behind Differential GPS (DGPS) is that in addition to a mobile receiver one or more nearby reference stations also monitor GPS signals. These reference stations know their position allowing them to compare the expected GPS observables with those actually received which can then be transformed into a correction signal that is transferred to the mobile receiver. The primary use case is to compensate for regionally fairly constant errors such as atmospheric effects (Grewal et al., 2007, sec. 1.3).

There are several DGPS systems in use which primarily differ in the number of reference stations and the way the correction signal is distributed. The most commonly used DGPS variants are the satellite based augmentation systems (SBAS) WAAS (Wide Area Augmentation System) in the United States and EGNOS (European Geostationary Navigation Overlay Service) in Europe (Grewal et al., 2007, sec. 1.4). Both systems use geo-stationary satellites to broadcast a correction signal which is processed by virtually all GPS receivers starting with the current generation of consumer grade devices.

In the EGNOS system the correction signal contains information on satellite clock and ephemeris errors as well as ionospheric errors (ESA, 2005a). EGNOS uses a grid model of the ionosphere covering Europe and parts of North Africa (ESA, 2005b). For each grid point the expected ionospheric delay and delay rate are transmitted as part of the correction signal. After applying the satellite clock error and ephemeris corrections EGNOS-enabled GPS receivers interpolate these data points and adjust the GPS pseudoranges accordingly (ESA, 2005a).

6.3.10 Assisted GPS (AGPS)

The primary goal behind AGPS is to improve the so called time to first fix (TTFF) – the time it takes from power-up until the receiver outputs the first positioning solution. This is achieved by providing receivers with a precise almanac independently from the GPS signal thereby reducing the ambiguity involved in the initial positioning solution. This is particularly useful in situations where the default receiver internal almanac is too imprecise to lock onto any GPS signal at all which is typically the case in so called "urban canyons" (few satellites visible, significant multipath effects).

AGPS data is usually either manually programmed into receiver internal memory while preparing a series of measurements or downloaded on-demand from vendor provided servers on the internet in applications where internet connectivity is available (cellular phones, WiFi cabable PDAs).

6.3.11 Receiver Internal Filters

GPS receivers typically use Kalman filtering to calculate the position solution (Grewal et al., 2007). Some receivers also stack additional Kalman filters on top of this to incorporate a motion model (pedestrian, automotive, aircraft). The effects can be dramatic if the receiver keeps sending motion model based position estimates instead of proper GPS derived positioning solutions[3]. Unfortunately, the exact inner workings of a GPS receiver are rarely documented by the receiver vendor and usually need be evaluated (to the limited extent possible) through experimentation.

[3]This happened during the course "Autonome Navigation im Außenraum" in the winter term 2007/2008.

6.3.12 GPS Output

To conclude the discussion of GPS fundamentals we still need to cover what type of output GPS receivers produce. Consumer grade receivers typically report at least GPS time, the 2D positioning solution (latitude, longitude) in WGS84 coordinates (section 7.1 on page 46), optionally also the altitude if a 3D solution was possible, the horizontal and vertical dilution of precision (DOP), information on the satellites used in the calculations (space vehicles, SV) and their positions.

To make this data available virtually all commercial GPS receivers support the so called NMEA 0183 protocol (short NMEA; controlled by the National Marine Electronics Association [US based but not a government body]) over RS232C or other serial links (USB, Bluetooth). The official documentation of this text based protocol is not freely available. The most comprehensive internet resource on NMEA the author is aware of is a list of NMEA sentences collected by Raymond (2009). In addition to NMEA many receivers also support vendor specific binary protocols which offer a more efficient way to get at the same information and sometimes also allow access to advanced features.

Although parsing NMEA is fairly straightforward the author decided to leave this up to the GPSD daemon. GPSD supports communication with GPS receivers through NMEA and a number of vendor specific binary protocols and offers a documented client library `libgps` which hides the protocol details from the user (c.f. section 8.2 on page 55).

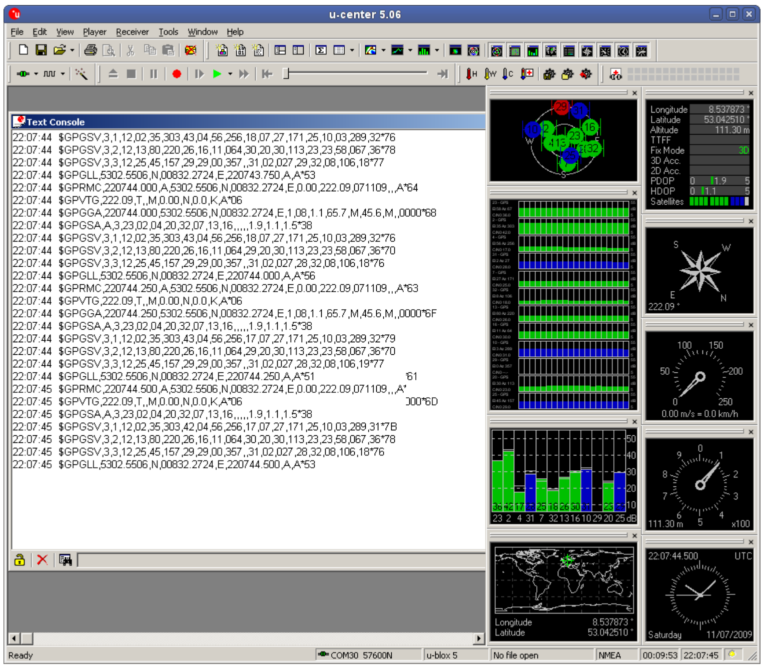
6.3.13 GPS Receivers for the SAMSMobil

The MTi-G comes with a built-in GPS receiver and an external antenna. Theoretically, this would be the ideal setup in the sense of providing GPS data synchronized with the IMU data stream. In practice, however, the MTi-G communication protocol (XSens Technologies B.V., 2008b) does not allow access to unprocessed GPS data in combination with calibrated IMU data. The device has to be switched to “RAW Data Output” mode to get at GPS data which also means that accelerometer, gyroscope and magnetometer readings are only available as raw 16-bit ADC measurements. This bypasses the essential application of the calibration and temperature compensation parameters. There is no documentation on how exactly this procedure is implemented on the MTi-G and judging from earlier experience with the subject (RescueRobotics Project, 2007) it would require substantial effort to replicate this step.

Instead, a replacement receiver was sought ideally with identical or better performance. The MTi-G manual does not specify the exact model of the built-in receiver but describes it as “GPS receiver; 16ch L1; SBAS” (XSens Technologies B.V., 2008b, Sec. 2.1) which essentially means that it is a standard consumer grade receiver (L1 frequency band only) with 16 channels and SBAS support. Use of EGNOS is enabled (XSens Technologies B.V., 2008b, Sec. 2.2.1) irrespective of the test mode bit. Furthermore, DGPS based on a custom reference station is supported by the receiver via RTCM but not accessible through the XSens protocol. The GPS update rate appears to be 1Hz (XSens Technologies B.V., 2008b, Sec. 4.7.2).

In addition to the above a replacement receiver also had to be available immediately and at a reasonable price. The Navilock NL-402U unit which includes a *u-blox 5* chipset with a patch antenna and a USB interface in a compact housing satisfied these constraints. The *u-blox 5* chipset is a state of the art consumer grade receiver typically used in cell phone[4] or automotive navigation applications. Noteworthy are its comparably high update rate of up to 4Hz and the fact that it speaks both NMEA and a documented binary protocol which is supported by GPSD thereby ensuring convenient access to its output data.

[4] `http://wiki.openmoko.org/wiki/GTA02_GPS`



```
22:07:44  $GPGSV,3,1,12,02,35,303,43,04,56,256,18,07,27,171,25,10,03,289,32*76
22:07:44  $GPGSV,3,2,12,13,80,220,26,16,11,064,30,20,30,113,23,23,58,067,36*78
22:07:44  $GPGSV,3,3,12,25,45,157,29,29,00,357,,31,02,027,29,32,08,106,18*77
22:07:44  $GPGLL,5302.5506,N,00832.2724,E,220743.750,A,A*53
22:07:44  $GPRMC,220744.000,A,5302.5506,N,00832.2724,E,0.00,222.09,071109,,,A*64
22:07:44  $GPVTG,222.09,T,,M,0.00,N,0.0,K,A*06
22:07:44  $GPGGA,220744.000,5302.5506,N,00832.2724,E,1,08,1.1,65.7,M,45.6,M,,0000*68
22:07:44  $GPGSA,A,3,23,02,04,20,32,07,13,16,,,,,1.9,1.1,1.5*38
22:07:44  $GPGSV,3,1,12,02,35,303,43,04,56,256,18,07,27,171,25,10,03,289,32*76
22:07:44  $GPGSV,3,2,12,13,80,220,26,16,11,064,29,20,30,113,23,23,58,067,36*70
22:07:44  $GPGSV,3,3,12,25,45,157,29,29,00,357,,31,02,027,28,32,08,106,18*76
22:07:44  $GPGLL,5302.5506,N,00832.2724,E,220744.000,A,A*56
22:07:44  $GPRMC,220744.250,A,5302.5506,N,00832.2724,E,0.00,222.09,071109,,,A*63
22:07:44  $GPVTG,222.09,T,,M,0.00,N,0.0,K,A*06
22:07:44  $GPGGA,220744.250,5302.5506,N,00832.2724,E,1,08,1.1,65.7,M,45.6,M,,0000*6F
22:07:44  $GPGSA,A,3,23,02,04,20,32,07,13,16,,,,,1.9,1.1,1.5*38
22:07:44  $GPGSV,3,1,12,02,35,303,43,04,56,256,17,07,27,171,25,10,03,289,32*79
22:07:44  $GPGSV,3,2,12,13,80,220,26,16,11,064,29,20,30,113,23,23,58,067,36*70
22:07:44  $GPGSV,3,3,12,25,45,157,29,29,00,357,,31,02,027,28,32,08,106,19*77
22:07:44  $GPGLL,5302.5506,N,00832.2724,E,220744.250,A,A*51                   '61
22:07:45  $GPRMC,220744.500,A,5302.5506,N,00832.2724,E,0.00,222.09,071109,,,A'
22:07:45  $GPVTG,222.09,T,,M,0.00,N,0.0,K,A*06                                 )00*6D
22:07:45  $GPGSA,A,3,23,02,04,20,32,07,13,16,,,,,1.9,1.1,1.5*38
22:07:45  $GPGSV,3,1,12,02,35,303,42,04,56,256,17,07,27,171,25,10,03,289,31*7B
22:07:45  $GPGSV,3,2,12,13,80,220,26,16,11,064,30,20,30,113,23,23,58,067,36*78
22:07:45  $GPGSV,3,3,12,25,45,157,29,29,00,357,,31,02,027,28,32,08,106,18*76
22:07:45  $GPGLL,5302.5506,N,00832.2724,E,220744.500,A,A*53
```

Figure 6.4: Screenshot of *u-center* showing NMEA output along with the visualized GPS data.

The exact output format as well as a number of settings affecting the way the receiver operates are highly configurable via the vendor provided *u-center* tool (Figure 6.4) which also supports uploading AGPS data to the receiver. AGPS data covering the next 1-14 days is provided by u-blox free of charge. Experience with the receiver shows that AGPS data significantly reduces the TTFF from several minutes to well below a minute.

In configuring the receiver a guide by the Paparazzi Project (Paparazzi Project, 2009) provided useful advice particularly because the *u-center* user interface is somewhat hard to navigate. Initially, the receiver was operated in NMEA mode as summarized in Table 6.2. NMEA turned out to be problematic because the positioning solution is split across several GPS sentences which confused GPSD due to conflicting information in each of these (2D vs. 3D fix) and because NMEA truncates (the textual representation of) numeric fields after a few decimal digits leading to discretization artefacts (c.f. Figure 12.8 on page 124). In response to this it was switched to the UBX binary protocol when EGNOS was enabled after the system had become officially operational on October 1, 2009 (ESA, 2009). The

Protocol	NMEA v2.3
Enabled messages	RMC, VTG, GGA, GSA, GSV, GLL
SBAS	disabled
Update rate	4Hz
Motion model	“Automotive” or “Airborne 6g”
AGPS data	5 days

Table 6.2: Initial *u-blox 5* configuration.

Protocol	UBX
Enabled messages	NAV_DOP, NAV_SOL, NAV_TIMEGPS, NAV_SVINFO
SBAS	EGNOS
Update rate	4Hz
Motion model	"Automotive"
AGPS data	1 day

Table 6.3: Revised *u-blox 5* configuration.

relevant parts of the new configuration can be found in Table 6.3.

6.4 Leuze rotoScan ROD4plus Laser Range Finder

The Leuze rotoScan ROD4plus (short: ROD4plus) is a planar laser range finder. It is based on time of flight measurements of pulsed laser beams: A pulsed laser beam is emitted from a laser diode, hits a rotating mirror from where it leaves the case of the sensor and is reflected from an object (if any). The reflected beam again hits the mirror and is sensed by a photo diode sensitive to the respective wavelength. The time of flight is then measured and can be transformed into a range given knowledge of the speed of light and the internal optical geometry of the sensor.

The frequency at which the mirror rotates determines the interval between two consecutive range scans. The ROD4plus has a nominal scan rate of 25Hz. Each scan covers a sweep of 190° and consists of 529 range measurements with an angular resolution of 0.36°. According to the manufacturer the ROD4plus achieves a distance measurement resolution of 5 mm and is able to reproduce scans to an accuracy of ±15 mm (Leuze electronic, 2008). In tests performed for this thesis one can see noise of sometimes greater than ±30 mm and additional low frequency fluctuations of about ±60mm. A sample data set of 40 consecutive scans that should show the straight line arising from the intersection of the scan plan with the floor is depicted in Figure 6.5 on the facing page. Similar effects occur with other flat surfaces such as walls although the exact result depends on the intersection angle and the surface material, i.e., its reflectivity properties.

6.5 Update Rates and Data Synchronization

A list of used sensors along with their respective connection types and update rates is given in Table 6.4.

XSens IMU	USB	100Hz
Wheel Encoders (STM32)	USB	100Hz
Navilock GPS	USB	4Hz
Leuze rotoScan ROD4plus laser range finder	Ethernet	∼ 25Hz

Table 6.4: Sensors, connection types and update rates.

IMU and wheel encoder measurements are hardware synchronized with the IMU clock as the master clock as discussed in section 4.3.3 on page 15. The other sensors have their own independent clocks.

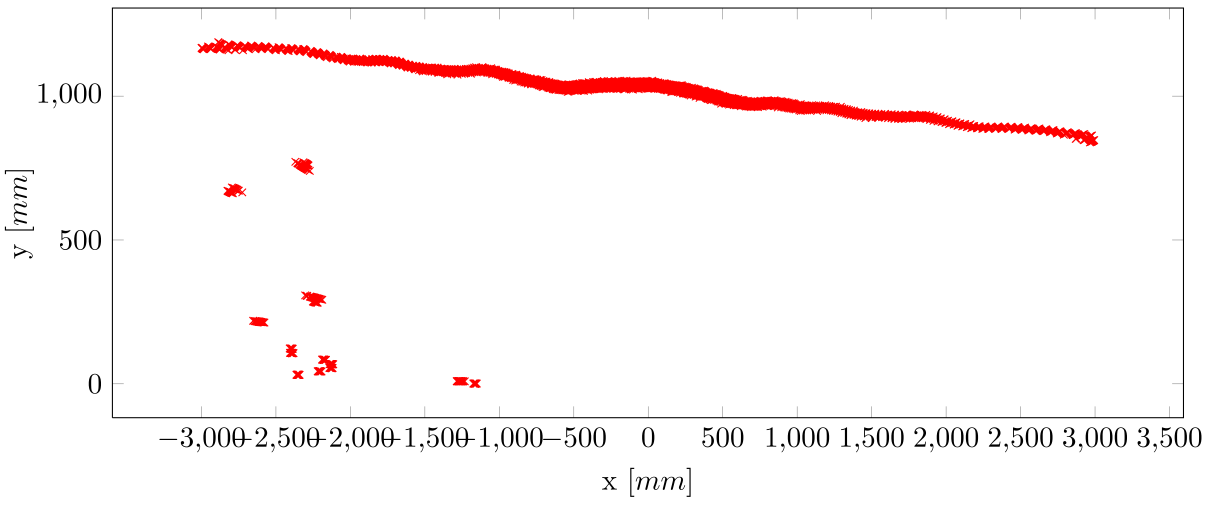


Figure 6.5: 40 consecutive raw ROD4plus scans showing the intersection with the ground plane in an indoor setting. Note the different scale on the two axes. The outliers are caused by table and chair legs.

The GPS receiver clock is synchronized to GPS time and used as a clock source by `ntpd` to keep the system clock synchronized (c.f. 5.1.1 on page 21).

Correct fusion of the sensor data from these different data streams requires proper synchronization between all of them. To understand this consider the data flow diagram in Figure 6.6. IMU, odometry and GPS data contribute to the pose estimate. The IMU and odometry data streams are already synchronized, but the temporal relation of GPS data to IMU/odometry data must also be known for this to work. Similarly, the pose estimate and lidar data contribute to the map – a lidar scan can only be correctly transformed into world coordinates if the pose of the robot at the time the scan was taken is known.

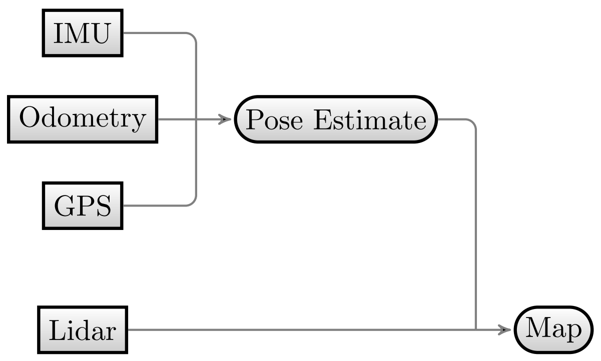


Figure 6.6: Sensor data sources (rectangular) and some of the products derived from them (rounded).

The solution to this problem turns out to be more difficult than one might expect. Initially, data was simply processed as it came in naively assuming that this always gives us the "latest" data items so that the pose estimate and map products always reflect the "latest" state of the system. This assumption does not hold in practice: I/O buffering may lead to different delays for different data sources and the scheduler will in general decide to grant the CPU to a single software module for longer than the time needed to process a single datum. The result is most visible if the mapper uses an outdated pose estimate to register more recent laser scans in the wrong place in the map.

The standard approach to avoid this is to use timestamps in system time to ensure that data is processed in the correct order. The idea behind this is that each driver module queries the value of the system clock in microseconds via `gettimeofday(2)` at the earliest time possible after the arrival of a new data item and uses this to tag the item with a timestamp. E.g., the ROD4plus driver calls `gettimeofday(2)` as soon as it has detected the start of a new scan data packet (i.e., the byte sequence indicating a new header) coming in over its TCP/IP socket. GPS data already contains timestamps in the form of GPS time as the system clock is synchronized to GPS time (c.f. 5.1.1 on page 21).

While this solves problems caused by modules that calculate products such as the pose estimate it does not work well when it comes to the accuracy of the source data timestamps since the `gettimeofday(2)` system call may be delayed for the same reasons as above. The first attempt as part of this thesis to work around this was based on the sequence numbers that IMU/odometry and lidar data packets provide. This allows us to calculate the timestamp t of a datum by multiplying the difference between the first and the current sequence number (seq_0 and seq) with the nominal update interval Δt and adding the time of the first datum t_0.

$$t = t_0 + (seq - seq_0) \cdot \Delta t \tag{6.3}$$

This generates jitter free timestamps for a single data source but, unfortunately, the nominal update intervals do not match the actual update intervals in practice so that synchronization between different data streams is lost quickly. E.g., the nominal update rate of the Leuze rotoScan ROD4plus laser range finder is 25Hz. Experiments have shown that the true update rate is slightly higher with an interval of $39.90ms$ instead of the nominal $40ms$.

This leads us to the final solution: During an "interval calibration" phase we have the driver modules measure timestamps using `gettimeofday(2)` as outlined above over a period of several seconds and otherwise drop the corresponding data items. For each data source we then compute the mean interval and the corrected start timestamp using linear regression. All later data items are passed on over IPC with their timestamps calculated using the calibrated start time $t_{0,calib}$ and interval Δt_{calib}.

$$t = t_{0,calib} + (seq - seq_0) \cdot \Delta t_{calib} \tag{6.4}$$

One could additionally keep monitoring the interval over the complete lifetime of the system and thereby adjust the interval to compensate for potential clock drift but in practice this has never been necessary especially given the limited battery life available on the SAMSMobil (max. 3.5 hours).

Note that this approach requires drivers to be able to receive sensor input data as soon as possible after the data has arrived on the respective (physical) I/O port. This is made possible by assigning the Linux `SCHED_RR` (round robin) scheduling policy and a real-time priority of 25 to all driver modules (c.f. section 8.2 on page 55). Now, as soon as data arrives for a driver module, i.e., the corresponding process becomes runnable, the Linux kernel will interrupt any other process (with lower priority) and allow the driver to process the incoming data. The round-robin policy furthermore ensures that all drivers run in the well-known round-robin fashion, i.e., a single driver module cannot run indefinitely. There is one catch in this scenario: Drivers must be able to pass sensor data on to the Isil message bus or jams in their socket buffers may occur. Thus, the Isil daemon `isild` must also run under the `SCHED_RR` policy but with a higher priority of 30.

We still have to discuss how the timestamps help with the fusion of the different data sources and how timestamps of products are derived. We will get back to this in chapter 8 on page 55.

Chapter 7

Coordinate Systems and Sensor Calibration

This chapter introduces the various coordinate systems used in the software components developed for this thesis and discusses the steps performed in the sensor calibration and cross-calibration. We will start with a brief discussion of the World Geodetic System 1984 (WGS84) coordinate system because its use by GPS receivers strongly influences later design decisions.

7.1 WGS84 (GPS) Coordinates

The WGS84 coordinate system is defined by a US Department of Defense standard (National Imagery and Mapping Agency, 2000) and uses a reference ellipsoid to approximate the Earth's surface at the mean sea level. According to Grewal et al. the *"... rotation axis [of the reference ellipsoid is] coincendent with the rotation axis of the earth, its center at the center of mass of the earth, and its prime meridian [goes] through Greenwich."*(Grewal et al., 2007, p. 349). The standard defines the reference ellipsoid by its semi-major axis a (radius at the equator) and reciprocal flattening $\frac{1}{f}$:

$$a = 6378137.0m \tag{7.1}$$

$$\frac{1}{f} = 298.257223563 \tag{7.2}$$

The semi-minor axis b (polar radius) is related to a and f as follows (Vincenty, 1975):

$$f = \frac{a-b}{a} \tag{7.3}$$

Thus, b can be computed as

$$b = a - f \cdot a \approx 6356752.314245m \tag{7.4}$$

As discussed in section 6.3.12 on page 40 GPS receivers typically (if they use the NMEA protocol) report positioning solution in geodetic longitude and latitude in the WGS84 ellipsoid model. The same format is used by GPSD and thus defines our input format of GPS measurements.

While it is possible to have a state estimation filter directly estimate the position in WGS84 coordinates (Grewal et al., 2007, Sec. 9.4.3.5) we will employ a simpler approach which is based on the observation that the operating range of the SAMSMobil is very small compared to the curvature of the WGS84 ellipsoid. The SAMSMobil uses a local Cartesian coordinate system with the x-y-plane coincendent with the tangential plane through a reference point on the WGS84 ellipsoid. The x-axis points east, the y-axis north and the z-axis into space.

Points in WGS84 coordinates are translated into Cartesian coordinates by computing their distance and azimuth to the reference point with the help of Vincenty's "inverse method" (Vincenty, 1975). The reverse operation is provided by Vincenty's "direct method" (Vincenty, 1975) but only needed to convert the state estimate back into WGS84 coordinates for visualization purposes (e.g., in Google Earth; see section 5.4 on page 24).

The C++ implementation of the inverse method was written from scratch based on the formulae in the paper. When the need for an implementation of the direct method arose Udo Frese suggested using an existing implementation originally developed for the course "Autonome Außenraumnavigation" at University of Bremen (winter term 2007/2008) which in turn is based on LGPL code from the open source "NMEA library"[1]. This implementation was re-used virtually unmodified and turns out to employ the exact formulae of Vincenty's direct method.

7.2 Coordinate Systems

Now that the **world** coordinate system has been introduced we will look at the definition of the remaining sensor coordinates systems.

- The **robot** coordinate system has its origin on the ground right in the middle of a line between the center pair of wheels of the SAMSMobil. It is fixed to the body of the robot which at any point in time "stands on" the x-y-plane in robot coordinates with the x-axis pointing to the right, the y-axis to the front of the robot and the z-axis pointing upwards. Odometry is measured in this coordinate system.
- The **lidar** coordinate system is fixed to the ROD4plus laser range finder with its origin being the optical center. The x-y-plane in lidar coordinates coincides with the scan plane with the x-axis pointing right and the z-axis pointing upwards.
- The **IMU** coordinate system is fixed to the MTi-G IMU with its origin being the origin of the sensor as defined in the manual (XSens Technologies B.V., 2008b) and its x-y-plane parallel to the base plate of the sensor where the x-axis points to the right of the robot and the z-axis points up[2].

The relationships between world, robot, lidar and IMU coordinates are defined by $\mathbb{R}^{4\times4}$ transformation matrices in homogeneous coordinates. Throughout the thesis we will use the naming scheme A2B to designate the transformation from coordinates in coordinate system A to those in coordinate system B, e.g., ROBOT2WORLD defines the pose of the robot in world coordinates and can be used to translate robot coordinates into world coordinates.

7.3 Laser Range Finder Calibration

As the Leuze rotoScan ROD4plus laser range finder can be operated as a stand-alone device (e.g. to turn off an appliance if someone steps into the scan plane) one would expect it to come readily calibrated from the manufacturer so that range readings received from the range finder can be used without any preprocessing. However, ROD4plus units virtually identical to the one on the SAMSMobil have previously been used by the Bremen Autonomous Wheelchair project[3] for the third generation

[1] `http://nmea.sourceforge.net/`

[2] Note that these axes are not identical to those defined by the markings on the sensor.

[3] `http://www.informatik.uni-bremen.de/rolland/index_e.htm`

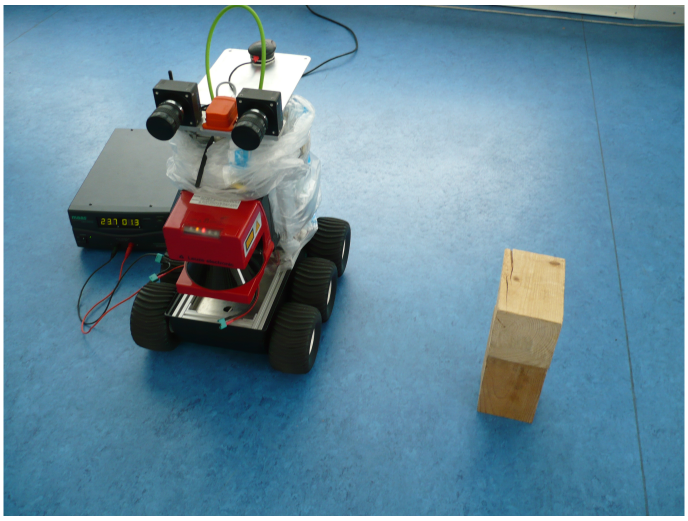


Figure 7.1: Laser range finder calibration setup

"Rolland" wheelchair (Mandel et al., 2005) and it turns out that an additional calibration is benefitial especially if range scans from multiple range finder units are to be combined (Gersdorf, 2009).

The idea behind the calibration procedure is to find a function that translates a range measurement received from the laser range finder into the "correct" range. The experimental setup of the calibration procedure replicates the one successfully used by the Bremen Autonomous Wheelchair project: a piece of a wooden beam serves as a calibration object which is placed at a known distance (measured manually up to millimeter precision) from the laser range finder as depicted in Figure 7.1.

The range scan data is inherently noisy, i.e., consecutive scans of the same object at the same distance will generally yield slightly different results. The effect is illustrated by the plot in Figure 7.2 on the next page which shows endpoints corresponding to the surface of the calibration object as reported by the ROD4plus laser range finder collected from 100 consecutive scans. Commonly used sensor models of laser range finders assume endpoints to be distributed according to a normal distribution in the vicinity of the sensed obstacle (Thrun et al., 2005a, sec. 6.3). We can thus recover the actual distance from the range finder data by taking the mean distance of the cluster of endpoints corresponding to the calibration object. Note that we ignore the washed out corners in doing so.

Varying the distance of the calibration object results in the dataset plotted in Figure 7.3 on the facing page. The dataset deviates from the model $f(d) = d$ we would expect after calibration. However, we also see that a linear model should be able to adequately represent the dataset. Linear regression yields a slope of 1.0441 and a y-intercept of -27.24 and in the ROD4plus driver (section 8.2 on page 55) this model is applied before publishing range scans.

7.4 IMU Calibration

The MTi-G IMU unit includes several types of sensors which as discussed in section 6.1 on page 30 should ideally be calibrated before they can be used.

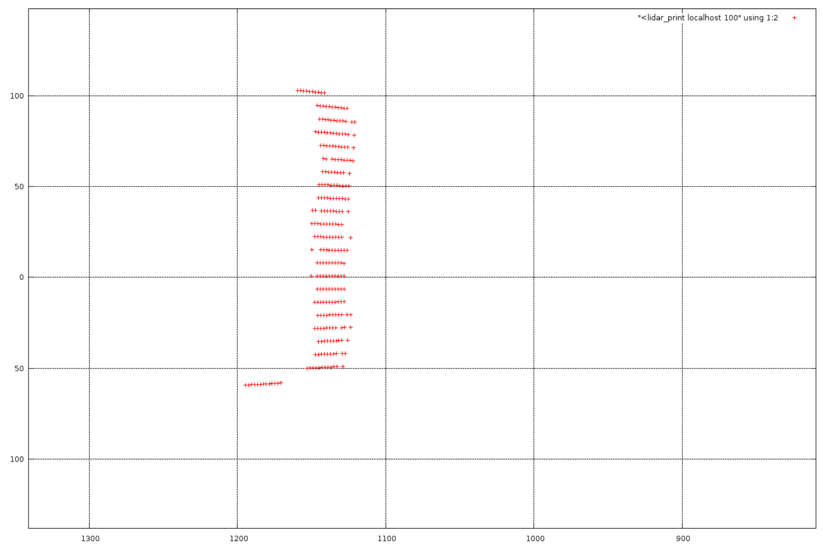


Figure 7.2: The surface of the calibration object as seen by the laser range finder (100 consecutive scans)

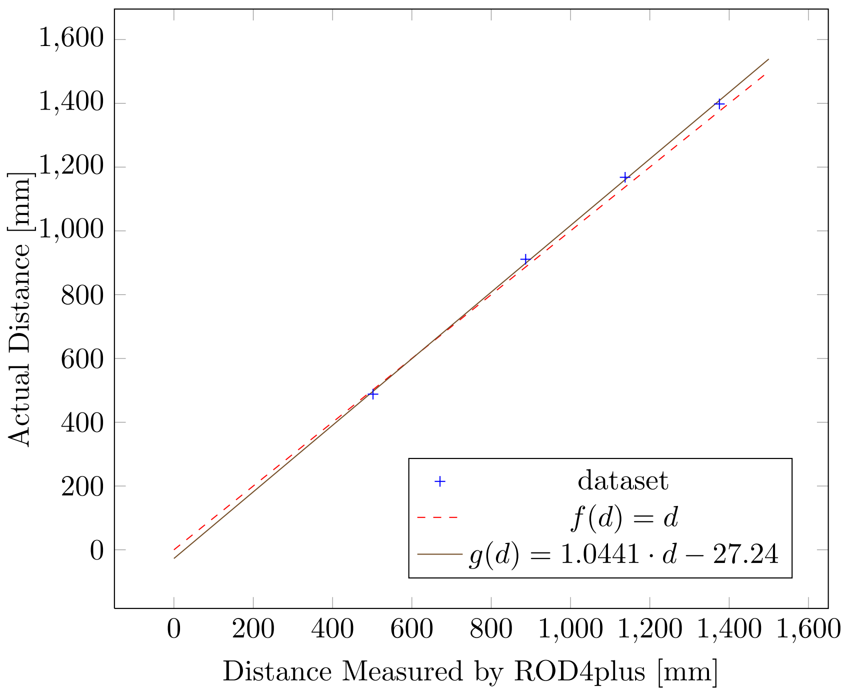


Figure 7.3: Laser range finder calibration dataset

7.4.1 Accelerometer Calibration

In each measured axis the accelerometer readings exhibit a different bias and scale factor. The Earth's gravity vector can serve as a calibration reference. For each axis this would require acquiring a sufficiently large set of measurements while the respective axis points exactly in the same direction as the gravity vector and another set for the exact opposite direction. Since the exact values depend on time varying external factors (c.f. section 6.1 on page 30) this procedure would need to be performed every time the robot is turned on.

This is obviously not a practical solution on the SAMSMobil so that the accelerometer measurements are used unmodified. As will be discussed later in section 9.8.2 on page 79 we compensate for this by including the bias terms in the state estimate. In our application the scale factor is not a problem because we only rely on the direction of the measured gravity vector and not its length.

7.4.2 Gyroscope Calibration

Before the state estimation filter uses any gyroscope measurements it collects a set of measurements while the robot is at rest to determine the inital gyroscope bias terms (c.f. section 9.8.11 on page 89) which are later estimated as part of the filter state.

Calibration of the scale factor would require rotating the IMU at a precisely known rate which is equally difficult to achieve as the exact orientation requirement with the accelerometers above. A visual comparison of the orientation of the sensor with the state estimate computed purely based on gyroscope in the visualization environment (c.f. section 5.5 on page 26) suggests that the factory provided calibration of the scale factor is, however, reasonably accurate, i.e., a 90° rotation of the sensor leads to a 90° rotation of the visualized state estimate.

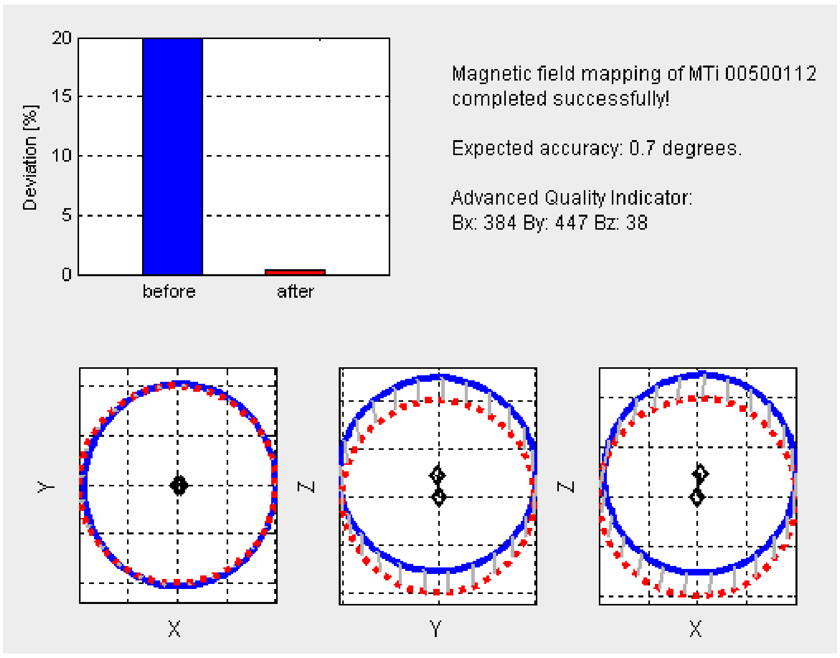
The author has experimented with adding the scale factor to the estimated state variables but this only introduces a degree of freedom which overlaps with the bias and is of no use to the filter.

7.4.3 Magnetometer Calibration

XSens provides a tool called *Magnetic Field Mapper* to compensate for distortions of the magnetic field by (ferro-)magnetic objects which are rigidly attached to the sensor. On the SAMSMobil the main source of distortions is the metallic case of the ROD4plus laser range finder. The calibration procedure relies on accelerometer and gyroscope measurements to estimate the orientation of the sensor and attempts to find a least squared errors mapping such that the 3D vector of magnetometer measurements has unit length and points in a consistent direction at all times (XSens Technologies B.V., 2008a).

The *Magnetic Field Mapper* implements a 2D and a 3D variant of this procedure. The 3D variant requires the complete robot to be slowly rotated around the origin of the IMU coordinate system for several minutes without exposing the IMU to any linear acceleration. Before the tool computes a calibration result it checks whether these constraints are met. The author has tried this procedure several times but eventually gave up since the generated calibration data sets were always rejected.

The 2D calibration procedure is less problematic as it only requires the robot to be rotated several times around the world z-axis. As illustrated by the calibration report in Figure 7.4 on the next page the distortion is significant which is in line with the data sets which were captured before calibration (Figure 12.7 on page 123). Unfortunately, the resulting calibration only leads to usable measurements



Magnetic field mapping of MTi 00500112 completed successfully!

Expected accuracy: 0.7 degrees.

Advanced Quality Indicator:
Bx: 384 By: 447 Bz: 38

Figure 7.4: Screenshot of the 2D calibration report produced by the XSens Magnetic Field Mapper.

if the robot has the same x- and y-orientation as when the calibration data set was acquired. Minor tilting by a few degrees already renders the magnetometer measurements useless.

The author had originally planned to perform an outdoor test run with the calibration obtained from the 2D procedure anyway but before this was possible the calibration was lost due to the MTi-G unit entering a "funny state" while it was in use by another user of the sensor, Oliver Birbach. In this state several internal variables were corrupted. After consulting XSens support Oliver Birbach eventually managed to get the sensor back into an operational state by performing a full factory reset. Due to time constraints[4] it was not possible to re-do the calibration. Also, the author did not want to risk further problems with the sensor which he suspects to be a side effect of the calibration procedure writing to internal memory that is otherwise never touched.

7.5 IMU and Laser Range Finder Cross-Calibration

The original plan was to avoid the need for cross calibration by mounting the IMU and laser range finder in such a way that the transformation LIDAR2IMU between the two coordinate systems would be a pure translation which could be easily determined based on the measured distance between the two sensors and on the position of the origin specified in the respective manuals. Both sensors are attached

[4]These were intensified by the fact that the calibration procedure has to be performed outdoors and the *Magnetic Field Mapper* could not be made to work on the author's Windows Vista laptop due to obscure errors with DLLs of the Visual C++ runtime despite trying various suggestions from the XSens support personnel and developers. The calibration procedure thus had to be performed with a borrowed XP laptop which was not always available.

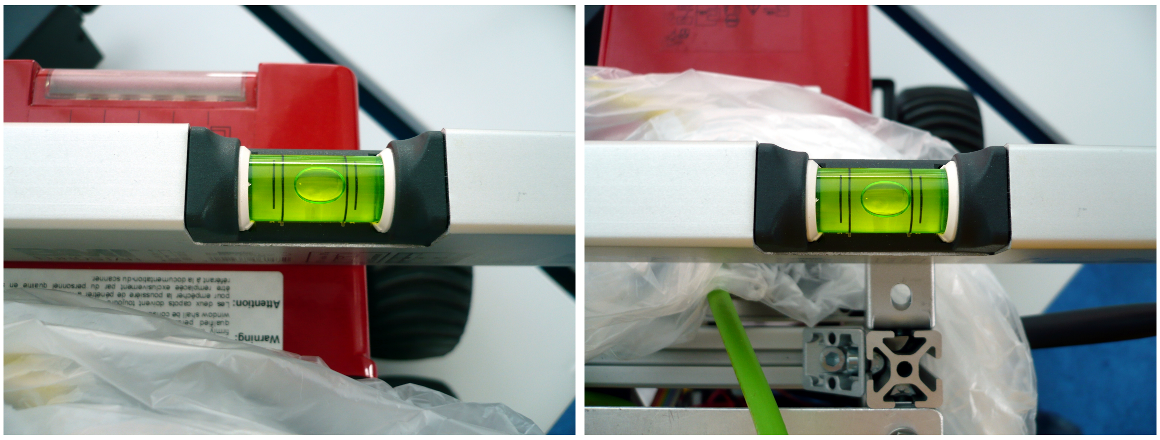


Figure 7.5: A level on top of the laser range finder case (left) and on the IMU/stereocamera head mounts (right). Up to the precision of the level the y-orientation of the two should be identical.

in a right angle to the aluminium frame (same x-orientation) and as can be seen in Figure 7.5 up to the precision of a level have the same y-orientation.

In practice, however, this leads to a significant mismatch between measured range scans and range scans simulated based on the intersection of the scan plane and the ground plane as illustrated in Figure 7.6. Thus, we need to find a way to calibrate the full LIDAR2IMU transform (including rotation).

Initially, the use of a calibration object similar to the pyramid-like object proposed by Antone &

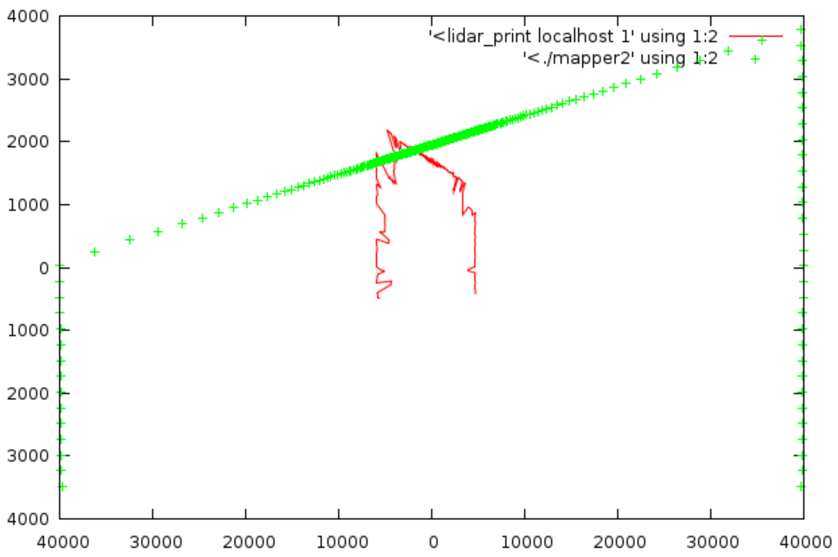


Figure 7.6: Assuming a pure translation for LIDAR2IMU the simulated range scan (green crosses) does not match the actual range scan (red line). The plot shows the scan plane (both axes in mm).

Friedman (2007) was considered. This would require the precise fabrication of the calibration object and the implementation of a calibration routine which, in particular, involves a non-trivial segmentation of lidar range scans. Instead, a simpler method was employed which simply uses the ground plane as a calibration object and works as follows:

- Based on a single range scan resulting from the intersection of the scan plane with the ground plane one can deduce that the range finder must be on the surface of a cone with a rotation axis identical to the line formed by the intersection.
- The orientation of the IMU relative to the ground plane is determined by the pose estimator. Its position is measured up to millimeter precision.
- The translation between the IMU and lidar coordinate systems is also measured up to millimeter precision.
- Observing the ground plane in range scans acquired from the same position relative to the ground plane but with different orientations eventually pinpoints the complete rotational part of LIDAR2IMU.

The latter is realized by solving a least squares optimization problem. The underlying assumption is that LIDAR2IMU can be understood as a rotation by α about the y-axis, followed by a rotation by β about the x-axis and a final translation as measured (see above). We then aim to find the vector $x^* = (\alpha \ \ \beta)^T$ which minimizes the function

$$F(x) = \sum_i (f_i(x))^2, \tag{7.5}$$

where $f_i(x)$ is defined as follows:

$$f_{k\cdot m+p}(x) = r_{e,k,p}(x) - r_{a,k,p} \qquad \text{for } p = 1,\dots,m \text{ and } k = 1,\dots,n, \tag{7.6}$$

where n is the number of range scans in the training set, m is the number of endpoints in each scan, $r_{e,k,p}(x)$ is the range expected for endpoint p in scan k determined by a simulation (c.f. section 10.7.2 on page 104) based on a LIDAR2WORLD = IMU2WORLD · LIDAR2IMU with $\alpha = x_1$ and $\beta = x_2$, and finally $r_{a,k,p}$ is the corresponding actual (measured) range. The training set has been picked manually from log data to cover various orientations and the range scans are preprocessed such that only endpoints within an x-range of $[-2m; 2m]$ are used because the rest corresponds to features other than the ground plane.

The calibration procedure uses the Levenberg-Marquardt algorithm (Madsen et al., 2004) in the form of MINPACK's LMDIF[5] (with a custom Eigen based C++ wrapper) to find x^*. We do not require a special (manifold based) Levenberg-Marquardt formulation here (c.f. (Birbach, 2008)) since the optimization starts with a good initial guess (based on manual experimentation) of

$$x = \begin{pmatrix} \mathrm{rad}(0.98°) \\ \mathrm{rad}(-0.9°) \end{pmatrix} \tag{7.7}$$

and the final rotation angles are comparably small:

$$\alpha^* = 0.9557807° \tag{7.8}$$
$$\beta^* = -0.7024297° \tag{7.9}$$
$$\tag{7.10}$$

The resulting LIDAR2IMU transform was tested on a test data set. A few sample plots comparing the actual and simulated ranges are depicted in Figure 7.7 on the following page.

[5] http://www.netlib.org/minpack/lmdif.f

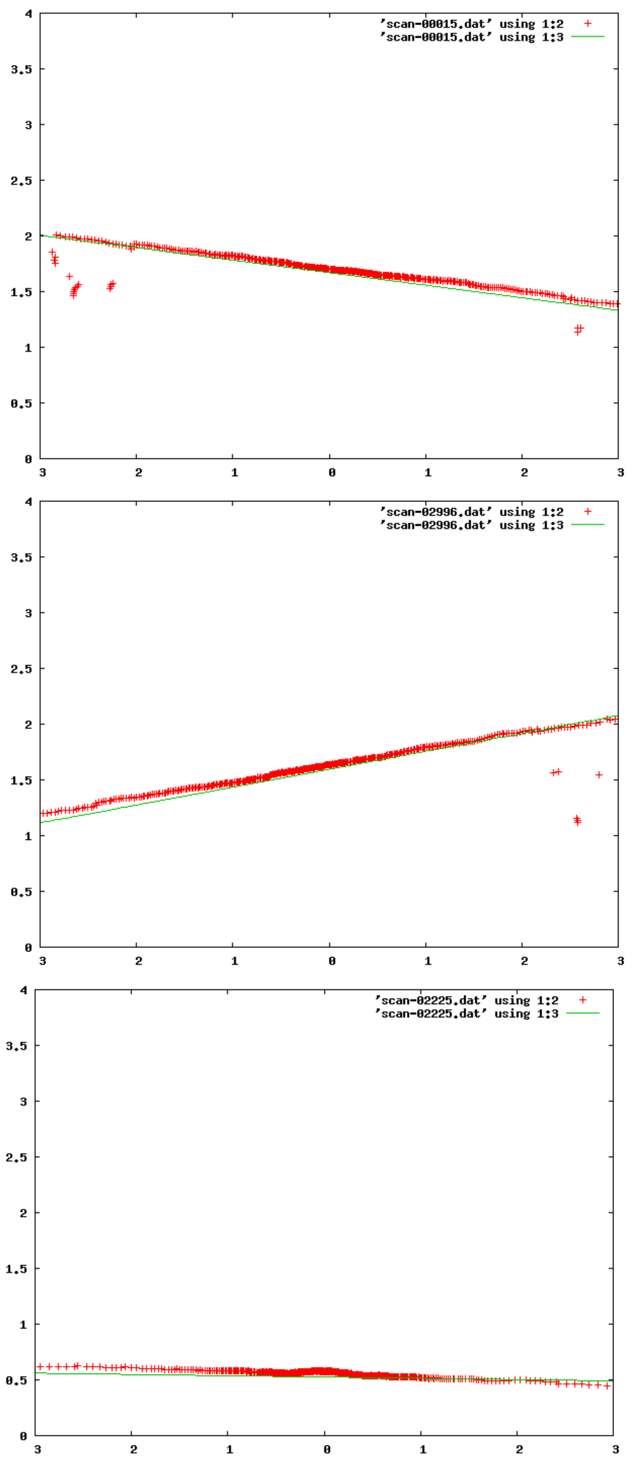


Figure 7.7: Actual and simulated range scans for a few samples from a test data set after calibration of LIDAR2IMU. The plot shows the scan plane (all axes in m).

Chapter 8

Software Modules and Control Flow

This chapter introduces the various on-board software modules that power the SAMSMobil in its outdoor navigation configuration. We will start with a flow chart of the processing pipeline and provide a high-level idea of which tasks the individual modules fullfil along with links to later chapters containing the algorithmic details behind the core modules.

8.1 Overview

The flowchart in Figure 8.1 on the next page illustrates the outdoor navigation processing pipeline. Sensor data is acquired by a set of driver modules, passed on to the pose estimator and mapper which generate a state estimate (pose, velocity, smooth pose) and a local obstacle map respectively, which are then used by the planner and path controller modules to derive control commands to be sent to the low-level vehicle controller. All sensor and control data is exchanged though messages that are distributed over the Isil IPC message bus (section 5.3 on page 23). Two additional inputs are the path boundary map section 9.8.10 on page 87 and the goal path both of which are read as KML files (c.f. section 5.4 on page 24) directly from disk by the modules requiring the respective information.

8.2 Drivers

A common property of all drivers is that they talk to the `isild` message bus daemon (c.f. section 5.3 on page 23). Since the Isil client library uses *Boost.Asio* for proactor style asynchronous I/O and network communication this means that all drivers need to be integrated with the *Boost.Asio* mainloop. While it is possible to integrate third party libraries which use standard Unix system calls directly this is generally inconvenient and error prone so that third party libraries have either been patched or the required I/O and network communication code been reimplemented. Note that this problem is not specific to Isil and occurs with any mainloop-based library (Qt, Gtk+, Carmen IPC, etc.).

The **STM32 driver** receives odometry from and sends control commands to the STM32 microcontroller. Since it essentially forms a bridge between the Isil message bus and the STM32 communication link it is called `stm32bridge`. It communicates with the STM32 through a Boost.Asio based reimplementation of the USB protocol originally developed as part of the SAMS project. Additionally, the `stm32bridge` sanitizes control inputs received via Isil to ensure that the vehicle is capable of executing the commands. In particular, ground friction causes the motors in the SAMSMobil to draw excessive amounts of current so that the rotational velocity needs to be limited to prevent the motor circuit fuse from blowing.

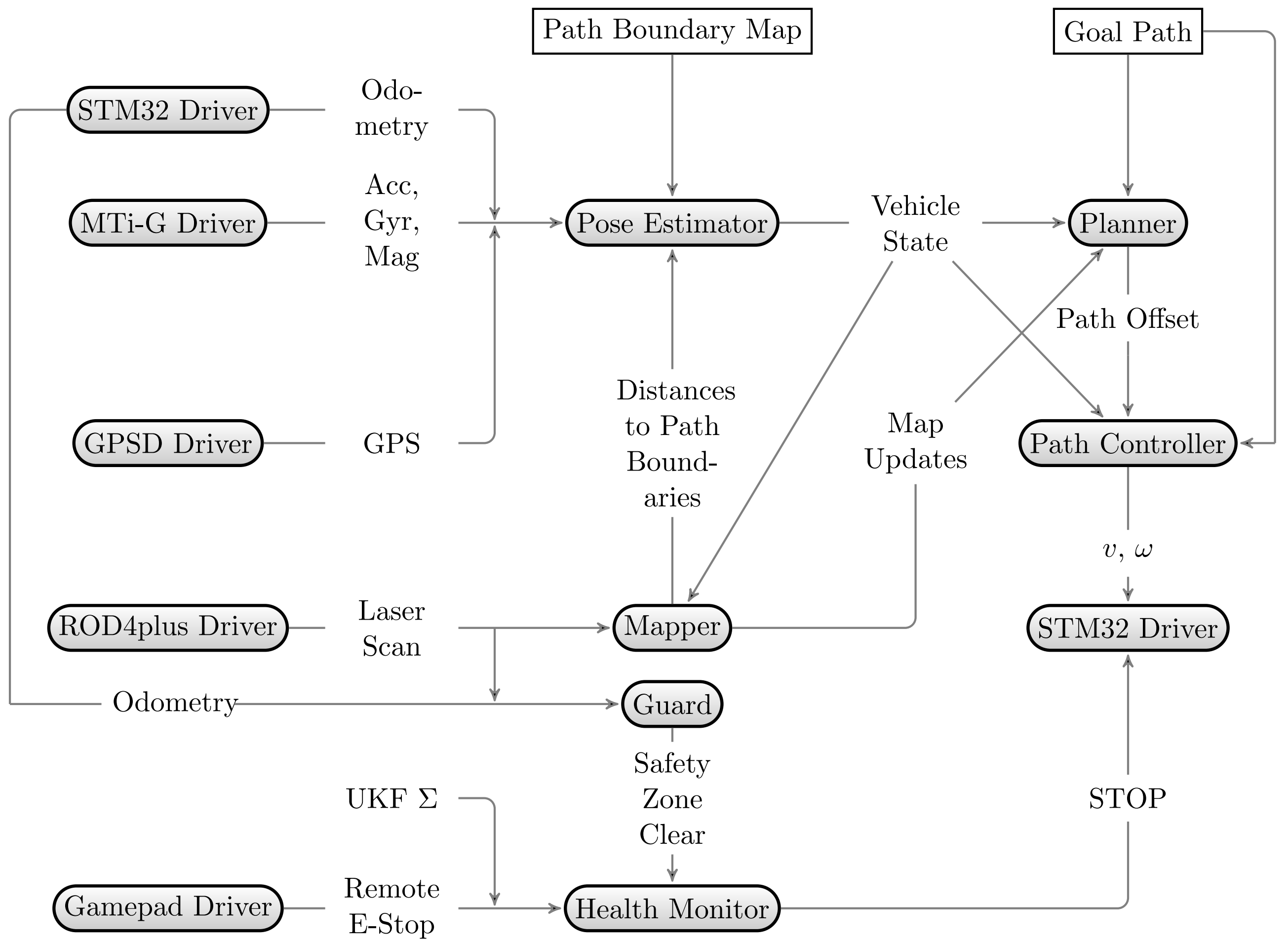


Figure 8.1: The SAMSMobil outdoor navigation processing pipeline. Note that the STM32 driver appears twice and that UKF Σ is a pose estimator output drawn separately for clarity.

The **MTi-G driver** `mti_g_bridge` configures the MTi-G on startup and then makes the IMU messages it receives available via Isil. In the final configuration this includes timestamped accelerometer, gyroscope and magnetometer readings in the “calibrated data” variant (see XSens Technologies B.V. (2008b) for details). The communication protocol of the MTi-G sensor transmits packets through a bi-directional bytestream over an RS232-C link which in our case sits behind a vendor provided USB to serial converter. To parse and generate these so called MTComm packet format parts of a library originally developed by Jörg Kurlbaum (Kurlbaum, 2007) are used. On top of this existing low-level library a set of C++ adaptor classes have been newly developed for this thesis which represent concrete messages and provide convenient constructors and accessor methods. Furthermore, a set of classes that provide a Boost.Asio based API for communication with the MTi-G over serial links has been implemented. Finally, it should be noted that the MTi-G supports different ways to encode individual numerical values. The SAMSMobil initially used the 32bit floating point format but this later switched to the 16/32bit fixed point format because it reduces the (quantization) noise in the data stream.

The **GPSD driver** `gpsd_bridge` receives GPS data from the GPSD Linux daemon and passes this on over the Isil message bus. To make this possible a *Boost.Asio* wrapper was implemented around the `libgps` library which comes with GPSD and hides the TCP/IP based GPSD communcation protocol behind a convenience library and makes measurements available through C structures. This required modifications of `libgps` to separate its parsing capabilities from the network communication parts since we need to use *Boost.Asio* for the latter.

For the **ROD4plus driver** the ROD4 communication protocol was implemented from scratch as per the vendor provided documentation using *Boost.Asio*. Holger Täubig of the SAMS project provided valuable hints as to inconsistencies between the documentation and the actual behaviour of the ROD4plus laser range finder.

The **gamepad driver** uses the new Linux event device based API (as opposed to the old joystick API) to talk to joystick or gamepad devices. Its implementation primarily consists of a port of code developed for the SAMS project to *Boost.Asio*. There are two main modes of operation: In race mode (Figure 8.1 on the facing page) the gamepad is used as a remote emergeny stop (e-stop) button. In manual test mode the robot receives control commands from the gamepad.

8.3 Pose Estimator

The pose estimator is a wrapper around Unscented Kalman Filter (UKF) implementation that will be discussed in section 9.8 on page 77. Here we will focus on how it interacts with the other modules.

The pose estimator receives sensor input from the STM32 driver (odometry), the MTi-G driver (IMU data: accelerometer, gyroscope and magnetometer values) and the GPSD driver (GPS). Before any processing can take place temporally related measurements need to be identified. For odometry and IMU data this is easy since corresponding IMU and odometry data items carry the same sequence numbers. The corresponding GPS data must be identified by timestamp. The pose estimator fuses each of the IMU/odometry pairs with the latest available GPS data item that carries a timestamp earlier than the IMU timetamp and tags the resulting pose estimate with the IMU timestamp. The exact reasoning behind the way GPS data is fused is discussed in section 9.8.7 on page 84.

The output of the pose estimator is the current (estimated) vehicle state consisting of the UKF mean pose and velocity and an additional pose estimate in “smooth coordinates” (see section 10.2 on page 91 for a description of the latter).

Optionally, the pose estimator can also fuse path boundary measurements received from the mapper. The idea is that the mapper implements a virtual sensor that detects the distance from the robot to the path boundary to the left and to the right of the vehicle. This narrows the position estimate down to places which have about this distance to road boundaries in a predefined path boundary map which is read from disk. The exact algorithm is described in section 9.8.10 on page 87. As far as the control flow is concerned, the pose estimator always fuses the latest available path boundary measurement (like with GPS above).

8.4 Mapper

The mapper implements the mapping algorithms discussed in chapter 10 on page 91 and operates slightly differently when it comes to combining different data sources. It combines the current laser scan with the interpolated pose derived from the pose estimate directly before and after the timestamp of the scan. The exact interpolation procedure is discussed in section 10.3 on page 92.

There are two main modes: The first generates the map using a re-implementation of the Probabilistic Terrain Analysis algorithm (PTA, see section 10.6 on page 95), the other uses a newly developed mapping algorithm called Ground Expectation Mapping (GEM, see section 10.7 on page 101). The result produced by the mapper is a local obstacle map which is needed by the planner to plan a path around obstacles. Instead of relying on shared memory, the planner replicates the obstacle map based on map updates received by the planner. To make this possible the mapper determines which map cells have changed after the integration of a complete laser scan and publishes information on these via Isil.

8.5 Planner

The job of the planner is to plan a path that generally follows a predefined goal path which it reads from disk but avoids obstacles. To this end the planner receives the current vehicle state from the pose estimator and map updates from the mapper. Every time a map update comes in it updates its replicated obstacle map and in combination with the current pose adjusts the path as needed and publishes the modification in the form of a so called path offset via Isil. The exact planning procedure will be presented in section 11.3 on page 111.

8.6 Path Controller

The path controller executes a given path trajectory. It starts with the goal path that it reads from disk and incorporates path offsets to keep it up to date with what the planner deemed necessary. We will get back to the exact algorithm in section 11.4 on page 114.

8.7 Health Monitor and Guard

All of the above modules only produce output when they receive input, i.e., the respective processes sleep until new data arrives on the Isil channels they are subscribed. By design, this means that if any of them crash the processing pipeline is interrupted and controller running on the STM32 board will eventually run into a timeout and perform an emergency stop (i.e., turn off both motors).

In some situtations, however, we want the robot to stop in response to an error condition that is not a software error. This is the job of the **health monitor**. It sends a stop command to the STM32 if either of the following conditions is true:

- The UKF covariance matrix (Σ) indicates there is a problem with the state estimate (c.f. section 9.8 on page 77).
- The **guard** has determined that an object in the raw laser scan is too close for the vehicle to keep driving at the current velocity as determined from odometry.
- The user initiated an emergency stop from the gamepad.
- Any of the modules sending the above information has not sent any new data for more than a certain timeout.

Effectively, the guard and the health monitor function as a safety net that prevents damage from being caused to the robot.

Chapter 9

Localization

As outlined above the approach chosen for the SAMSMobil treats localization and mapping as largely separate tasks. In particular, the SAMSMobil does not employ simultaneous localization and mapping (SLAM). This chapter will first differentiate the related problems of state estimation, localization and SLAM, discuss which approach appears appropriate for the SAMSMobil, then move on to the family of Bayes filters and, more specifically the (unscented) Kalman filter which lays the algorithmic basis of state estimation and localization on the SAMSMobil. Finally, the concrete design and implementation of the filter used on the SAMSMobil will be discussed.

9.1 State Estimation vs. Localization vs. SLAM

Before working with the terms state estimation, localization and SLAM we will first examine their respective definitions within the probabilistic framework by Thrun et al. (2005a) which can be seen as the de-facto standard in the current robotics literature. The key complexity involved in all three approaches is the inherent uncertainty in perception, control, and the environment (unmodelled state). Probabilistic methods model these error sources explicitely and aim to yield estimates that optimally incorporate knowledge from all available sources (sensors).

State estimation refers to the task of estimating the current state x_t of a system given all measurements $z_{1:t-1}$ and controls $u_{1:t}$. In probabilistic terms this is modelled as the following probability distribution:

$$p(x_t|z_{1:t-1}, u_{1:t}) \tag{9.1}$$

The exact characteristics of a *state* x_t are application specific. Typically, x_t contains a set of variables to provide sufficient information in order to execute a certain application specific task. For autonomous navigation x_t would at least describe the robot pose (position plus orientation). Control of a manipulator arm to move objects would require knowledge of its kinematic configuration and the location of each object (i.e., parts of its environment) at time t.

A *measurement* z_t describes the result of a perceptual interaction of the robot with its environment (Thrun et al., 2005a, Sec. 2.3.2). For our purposes this should be an observation that reveals clues about the relevant state of the robot and its environment. Perception generally works with the help of sensors such as cameras, laser range finders, or, more trivially, a switch that indicates whether a mobile robot's bumper has collided with an object. It should be noted that although we often refer to a "measurement z_t" as a single entity it can be a vector and all algorithms in this chapter are Bayes filter variants which can be extended to handle multiple, altogether different types of measurements from different sensors received at the same time step t (Thrun et al., 2005a, Sec. 2.3.2) by multiple invocation of the respective part of the algorithm (the Bayes filter correction step).

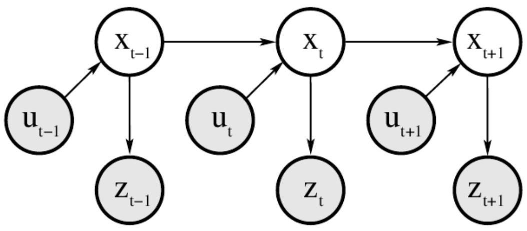


Figure 9.1: Bayesian network representing the state estimation problem (Image source: Thrun et al. (2005a)).

Controls are a little less intuitive. The name implies that a control u_t refers to the control commands to be executed by the robot to get from state x_{t-1} to state x_t (Thrun et al., 2005a, Sec. 2.3.2). The key point within the probabilistic framework, however, is that controls contain information on the change in state rather than on the state itself. In practice, there is a class of sensors that happen to measure data providing clues about this very state transition so that these sensor inputs are used in lieu of control commands. Examples of such sensors include wheel encoders (section 6.2 on page 32) and inertial sensors (section 6.1 on page 30).

In the following we will additionally require each state x_t to be complete as per the *Markov assumption* (Thrun et al., 2005a, Sec. 2.3.3), i.e., knowledge of x_t is sufficient to predict the outcome of measurement z_t:

$$p(z_t|x_{0:t}, z_{1:t-1}, u_{1:t}) = p(z_t|x_t) \tag{9.2}$$

and given knowledge of x_{t-1} we only need to know the control u_t to deduce the next state x_t:

$$p(x_t|x_{0:t-1}, z_{1:t-1}, u_{1:t}) = p(x_t|x_{t-1}, u_t) \tag{9.3}$$

Equations (9.2) and (9.3) express conditional independence which is graphically illustrated in the form of a *Bayesian network* in Figure 9.1[1]. Note that $p(z_t|x_t)$ is often referred to as the *measurement model* and $p(x_t|x_{t-1}, u_t)$ as the *motion* or *process model.*

It should be noted that the Markov assumption poses a fairly strict requirement on the *completeness* of a state (Thrun et al., 2005a, Sec. 2.4) and in practice it is generally impossible to fully satisfy it in the sense that a fully complete model of the world state covering all eventualities – regardless of how unlikely they are – cannot be achieved. Again, we pragmatically handle this by modelling those parts of the state that are required to yield a sufficiently precise (within the application context) approximation of the respective probability distributions.

To extend the above examples this means that although the navigation application only cares about the current pose it may be necessary to add the current velocity to the state variables in order to satisfy the Markov assumption to an extent that allows the state estimation algorithm to produce useful estimates. However, we will not attempt to model possible causes of unlikely events such as natural disasters as part of the state representation.

Localization adds another concept into the mix – that of a *map.* The robot is given a global *map* containing a suitable description of its environment such as a floor plan for a robot that operates in

[1]Nodes in a Bayesian network represent random variables. A directed arc from node A to B signifies that A is a parent of B. The local semantics of Bayesian networks say that a node is conditionally independent of all other nodes given its parents (Russell & Norvig, 2002, Sec. 14.2)

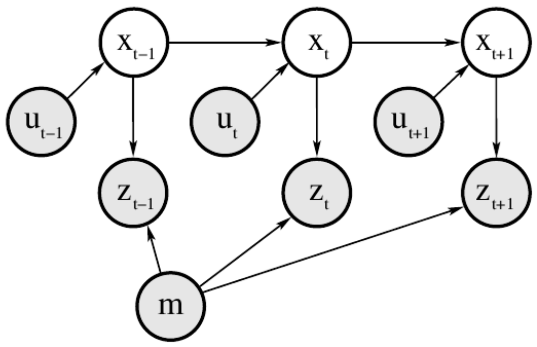


Figure 9.2: Bayesian network representing the localization problem (Image source: Thrun et al. (2005a)).

an office environment. The localization task now consists of figuring out where in the map the robot currently is. More specifically, the robot's current pose expressed in the global (map) coordinate system is to be estimated (Thrun et al., 2005a, Chap. 7). The corresponding Bayes network is depicted in Figure 9.2.

Within the context of localization the measurement model now becomes:

$$p(z_t|x_t, m) \tag{9.4}$$

and the motion model is sometimes also extended to:

$$p(x_t|x_{t-1}, u_t, m) \tag{9.5}$$

Furthermore, Thrun et al. (2005a) distinguish *position tracking* and *global localization*. Position tracking refers to the special case where the initial pose of the robot is known and estimation only needs to take local uncertainty into account while in global localization no prior knowledge of the initial pose is available and the estimation starts with a uniform distribution over the entire map (Thrun et al., 2005a, Sec. 7.1).

Finally, SLAM combines the task of localization with that of mapping[2] in a possibly surprising way. The idea is that initially neither the robot pose nor a map are known resulting in the "chicken and egg" type problem of simultaneously building a map and localizing itself within that map. Thrun et al. (2005a) differentiate between *online SLAM* which aims to estimate the map together with the current pose:

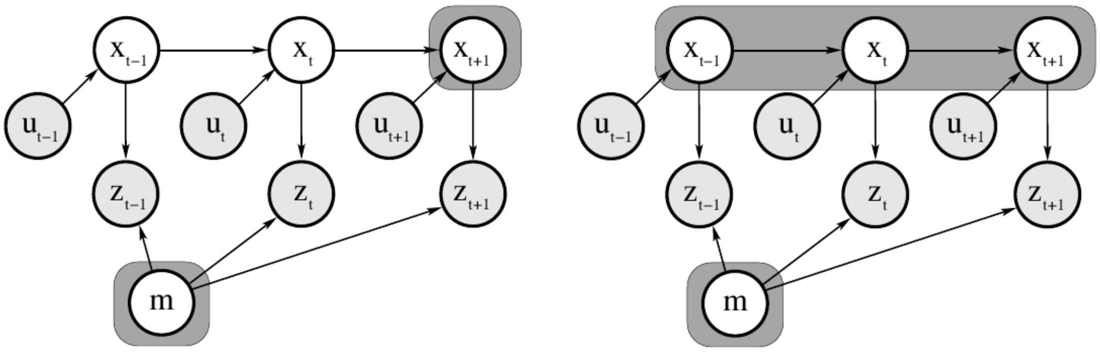
$$p(x_t, m|z_{1:t}, u_{1:t}) \tag{9.6}$$

and *full SLAM* where the joint probability of all poses is to be estimated along with the map:

$$p(x_{1:t}, m|z_{1:t}, u_{1:t}) \tag{9.7}$$

Figure 9.3 on the next page illustrates the corresponding Bayes networks. It should be noted that as the name implies autonomous robots typically employ online SLAM since full SLAM is often not computationally feasible (Thrun et al., 2005a, Sec. 10.1).

[2]See chapter 10 on page 91 for the mapping approach used on the SAMSMobil.



(a) Online SLAM (Image source: Thrun et al. (2005a)). (b) Full SLAM (Image source: Thrun et al. (2005a)).

Figure 9.3: Bayesian networks representing the online and full SLAM problems. The shaded areas are to be estimated.

9.2 Which Approach on the SAMSMobil?

Autonomous navigation relies on knowledge of both the robot pose and a map of the environment. Given the above one might argue that since SLAM estimates both it should generally be the approach of choice and thus also implemented on the SAMSMobil. This was indeed considered but discarded for the following reasoning.

For the sake of the argument we will borrow the graph representation of the SLAM problem used by GraphSLAM (Thrun et al., 2005a, Chap. 11). GraphSLAM graphs consist of state nodes (poses) x_t and map feature nodes m_k. State transitions are treated as constraints affecting a pair of state nodes each. Measurements are considered as constraints on a map feature and one or more state nodes. Both constraint types are stored as arcs in the graph. An example constraint graph is depicted in Figure 9.4.

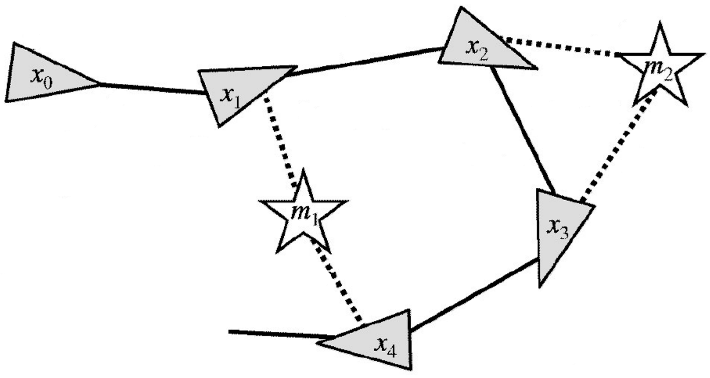


Figure 9.4: Constraint graph used by GraphSLAM (Image source: Thrun et al. (2005a); original annotations removed).

Note that, in contrast to the Bayes networks above, here the map is not modelled as a single entity. Instead, individual map features are treated separately reflecting the fact that in most applications only parts of the map are observed at each point in time. This gives rise to the notion that the same map feature m_i must be observed from at least two states x_{t_1} and $x_{t_2 \neq t_1}$ for the constraint to propagate through the graph to the path formed by the sequence of states. In Figure 9.4 on the previous page m_1 is observed from the states x_1 and x_4, m_2 is seen from x_2 and x_3.

If we now consider the situation on the SAMSMobil we note that given its particular forward tilted laser range finder setup pure forward motion (i.e., the orientation does not change) leads to no overlap in the two “slices” of the environment sensed through the laser range finder. Changes in the z-axis angle (turning) lead to an overlap in at most a single point.

In practice the robot is exposed to terrain induced orientation changes since hardly any outdoor surface is perfectly smooth (leading to high frequency orientation changes) or plain (leading low frequency, high amplitude orientation changes). Thus there is some additional overlap in features measured from different poses but the effect is not controlled and its impact in the sense of forming constraints is neglegibly small particularly if the precision of the laser range finder is taken into account.

One might try to artificially create overlap through projection of 3D laser endpoints into the 2D plane. This would cause significant errors: Unlike in many indoor environments slices of an outdoor environment at different height levels will generally not be identical. Also, this approach cannot yield an estimate of 3D orientation since the 2D projection loses the information required for that.

However, even if we ignore those errors the overlap is spatially limited since the intersection of the scan plane with the ground plane effectively reduces the range covered by the laser range finder. On average we would expect to see overlaps of around twice the distance from the laser range finder to the intersection in front of the vehicle.

Thus it is doubtful whether the estimated trajectory is going to be better than that produced by a pure state estimation or localization approach. Also note that we do not benefit from the map and trajectory correction which usually occurs in SLAM upon loop closures during (potentially long) periods of time between loop closures. I.e., we need the current pose to guide online planning and control and we need a high quality estimate at any given point in time. As for the map, we are not necessarily interested in a global map since obstacle avoidance is an inherently local task. On the other hand SLAM algorithms tend to be complex both in terms of high implementation effort and computational costs so that the risk involved in pursuing a SLAM approach most likely outweighs the expected benefit.

Instead, we will employ a state estimation approach with the assumption that the initial pose is known. In our race track style scenario this is obviously a given and in the case of a future automotive application it is equivalent to the assumption that a car is intermittently parked in known locations such as the driveway or garage at home. Later in this chapter we will extend this to a localization (position tracking) approach that is aided by a predefined global map.

9.3 Bayes Filters

In recent years the family of Bayes filters has seen successful use in a wide variety of state estimation applications on mobile robots. Bayes filters maintain a *belief* over the current state x_t and use controls in the *prediction step* (9.8) and measurements in the *measurement update* or *correction step* (9.9) to recursively update the belief (Thrun et al., 2005a, Sec. 2.4).

$$\overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1})bel(x_{t-1})\, dx_{t-1} \tag{9.8}$$

$$bel(x_t) = \eta\, p(z_t|x_t)\overline{bel}(x_t) \tag{9.9}$$

There are several ways in which a Bayes filter can be implemented. The most popular ones come in the form of particle filters and various Kalman filter variants. Particle filters represent the belief through a set of Monte Carlo style samples ("particles") and are particularly able to maintain multiple hypotheses at a time. They have a reputation of being comparably easy to implement but fairly resource hungry since an adequate approximation of the belief distribution may require a large number of particles. Kalman filters on the other hand represent the belief as a (multivariate) Gaussian distribution which is obviously best suited to maintain a single hypothesis at a time[3]. Kalman filter implementations have a reputation of being more involved but also computationally cheaper.

For the SAMSMobil an unscented Kalman filter (UKF) was chosen to track its pose. The primary reason behind this decision is that we need a software stack that operates in real time on the available hardware which in contrast to, e.g., Stanley (Thrun et al., 2006b) cannot arbitrarily be extended with additional computing nodes. Based on previous experience with Kalman filtering (RescueRobotics Project, 2007) the author confident this would be possible with a UKF. Also, in a pose tracking setting multiple hypotheses primarily arise whenever the robot takes turns and a particle filter "tries out" a number of different trajectories to weed out unlikely ones later on. Although this may be an advantage in estimating the overall trajectory the effect comes too late since in our application we need the pose estimate for navigation purposes already while the robot is still turning. We do not benefit from significant later improvements (as would result from a completely different hypothesis becoming more likely) since this means the robot has long ended up stuck off the path.

9.4 The Kalman Filter

Before we discuss the UKF we will introduce its ancestor, the Kalman filter (Kalman, 1960) first which is an instance of the Bayes filter for continuous state and discrete time state estimation of linear Gaussian systems. The primary underlying idea is that the probability distribution of the multivariate random variable to be estimated is not represented up to arbitrary precision but only such that the first two moments (mean, covariance) are preserved (Julier & Uhlmann, 1996) in the belief. This moments parameterization fully describes a Gaussian distribution since all of its other moments are zero (Thrun et al., 2005a, Sec. 3.1).

The Kalman filter represents the current belief $bel(x_t)$ as a multivariate Gaussian distribution with mean $\mu_t \in \mathbb{R}^n$ and covariance $\Sigma_t \in \mathbb{R}^{n\times n}$ and is based on a few assumptions (Thrun et al., 2005a, Sec. 3.2):

- The initial belief $bel(x_0)$ must be a Gaussian distribution, i.e.,

$$bel(x_0) = \mathcal{N}(\mu_0, \Sigma_0)\,. \tag{9.10}$$

[3] Note that this does not rule out the option of running multiple Kalman filter instances as is often done in multi-hypothesis tracking.

- The process model must be a linear Gaussian. It must be linear in its arguments subject to additive white Gaussian noise, i.e,

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t, \tag{9.11}$$

  where

$$\epsilon_t \sim \mathcal{N}(0,\, R_t)\,. \tag{9.12}$$

- Similary, the measurement model must be a linear Gaussian and thus satisfy

$$z_t = C_t x_t + \delta_t, \tag{9.13}$$

  where

$$\delta_t \sim \mathcal{N}(0,\, Q_t)\,. \tag{9.14}$$

Based on these assumptions the Kalman filter can be derived (Thrun et al., 2005a, Sec. 3.2.4) from the Bayes filter update rule ((9.8) and (9.9)) and it can be shown that it is optimal in the sense of calculating a minimum mean squared error (MMSE) estimate (Julier & Uhlmann, 1997). In the following we will present the resulting formulae comprising the Kalman filter algorithm.

Like the Bayes filter, the Kalman filter algorithm can be split into a prediction and a correction step. The prediction step takes as input the previous belief represented by the mean μ_{t-1} and covariance Σ_{t-1} of the respective normal distribution as well as a control u_t. Both, mean and covariance are propagated through the linear process model to yield the predicted estimate described by $\bar{\mu}_t$ and $\bar{\Sigma}_t$ as follows:

$$\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t \tag{9.15}$$
$$\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t \tag{9.16}$$

Note how the measurement noise (R_t) is added to the covariance matrix in (9.16).

The correction step is a little more involved. It takes the prediction and a measurement z_t as input and adjusts the state estimate as follows:

$$K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1} \tag{9.17}$$
$$\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t) \tag{9.18}$$
$$\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t \tag{9.19}$$

Two concepts are essential here: The so called *innovation* $z_t - C_t\bar{\mu}_t$ says how much the actual measurement z_t differs from the measurement that is expected in state $\bar{\mu}_t$ according to the measurement model. The Kalman gain K_t determines how much the innovation is used to correct the state estimate. It can be thought of as a measure of how much the filter trusts the measurement in relation to the state estimate (and vice versa) or how the two need to be balanced to optimally fuse the information contained in both.

9.5 The Unscented Kalman Filter (UKF)

Many real world applications involve non-linear systems for which the regular Kalman filter is not applicable. Up until recently the most popular extension to the Kalman filter to address this has been the extended Kalman filter (EKF) which works around the problem by linearizing process and measurement models via first order Taylor series expansion.

Julier & Uhlmann (1997) point out two main issues with the EKF approach:

- The linearization requires knowledge of the Jacobians which can be difficult to derive for complex process or measurment models.
- It assumes that all higher order Taylor series terms are neglegible which, in practice, is often not the case.

The first issue is often worked around through numerical approximation of the Jacobians. The second point is more severe. Crude approximation of the respective functions can lead to significant errors and, worse yet, may cause the filter to overestimate the accuracy of its state estimate (i.e., the covariance is lower than the mean squared error). Inconsistent and biased prediction of the state or measurement distributions are the result and bear the potential of filter divergence (Julier & Uhlmann, 1997).

This is often worked around via artificially increased noise terms. The tuning of these, however, can be difficult: too large noise terms make the filter inefficient, too small noise terms can lead to (depending on the application potentially dangerous) filter divergence (Julier & Uhlmann, 1997).

The UKF replaces the problematic linearization through first order Taylor series expansion with the so called *unscented transform* which is based

> *"...on the intuition that it is easier to approximate a Gaussian distribution than it is to approximate an arbitrary nonlinear function or transformation"* (Julier & Uhlmann, 1997).

Consequently, the idea is to parameterize the belief distribution by a set of deterministically picked samples (called *sigma points*), apply the original non-linear process or measurement model to transform each sample, and finally recover the statistics of the transformed distribution from the transformed samples. Jacobians of the process or measurement model are not needed – the UKF is therefore often called a *derivatice free* filter (Thrun et al., 2005a). The discrete (sample based) approximations of the respective distributions should have the property that at least the first and second moments should be preserved (Julier & Uhlmann, 1996).

To the best of the author's knowledge this makes the UKF the "best of its class" algorithm in handling non-linearities. We will now look at the exact operation of the unscented transform and how it is used to formulate the UKF algorithm.

9.5.1 Algorithm

The unscented transform and the UKF have undergone a fairly long evolution from early papers on the unscented transform (Quine et al., 1995) to the work by Julier and Uhlmann (Julier & Uhlmann, 1996, 1997) and by van der Merwe et al. (van der Merwe & Wan, 2003; van der Merwe et al., 2004). The following is based on the UKF formulation by Thrun et al. (2005a) which, given an appropriate choice of parameters, is equivalent to the earlier variants.

Non-Linear Process and Measurement Models

In comparison to the regular Kalman filter, the UKF drops the linearity requirement on the process and measurement models but keeps the requirement that both be subject to additive white Gaussian noise, i.e.,

$$x_t = g(u_t, x_{t-1}) + \epsilon_t \tag{9.20}$$
$$z_t = h(x_t) + \delta_t \tag{9.21}$$

where $g : \mathbb{R}^p \times \mathbb{R}^n \to \mathbb{R}^n$ and $h : \mathbb{R}^n \to \mathbb{R}^m$ are arbitrary (but sufficiently nice) functions, $\epsilon_t \sim \mathcal{N}(0, R_t)$, $\delta_t \sim \mathcal{N}(0, Q_t)$, and all ϵ_t and δ_t are independent.

Sigma Points and the Unscented Transform

The set of $2n+1$ *sigma points* of an n-dimensional Gaussian distribution with mean μ and covariance Σ is computed as follows (Thrun et al., 2005a, Sec. 3.4.1):

$$\begin{aligned}
\mathcal{X}^{[0]} &= \mu \\
\mathcal{X}^{[i]} &= \mu + \left(\sqrt{(n+\lambda)\Sigma}\right)_{\bullet i} \quad \text{for } i = 1, \ldots, n \\
\mathcal{X}^{[i+n]} &= \mu - \left(\sqrt{(n+\lambda)\Sigma}\right)_{\bullet i} \quad \text{for } i = 1, \ldots, n
\end{aligned} \tag{9.22}$$

where $\sqrt{(n+\lambda)\Sigma}$ denotes the matrix square root of $(n+\lambda)\Sigma$, and $\lambda = \alpha^2(n+\kappa) - n$. The scaling parameters α and κ will be discussed later.

The unscented transform now propagates the distribution represented by its set of sigma points through a function f:

$$\mathcal{Y}^{[i]} = f(\mathcal{X}^{[i]}) \tag{9.23}$$

and uses a set of weights

$$\begin{aligned}
w_m^{[0]} &= \frac{\lambda}{n+\lambda} \\
w_c^{[0]} &= \frac{\lambda}{n+\lambda} + (1 - \alpha^2 + \beta) \\
w_m^{[i]} = w_c^{[i]} &= \frac{1}{2(n+\lambda)} \quad \text{for } i = 1, \ldots, 2n
\end{aligned} \tag{9.24}$$

when recovering the mean and covariance of the transformed distribution as follows:

$$\mu' = \sum_{i=0}^{2n} w_m^{[i]} \mathcal{Y}^{[i]} \tag{9.25}$$

$$\Sigma' = \sum_{i=0}^{2n} w_c^{[i]} (\mathcal{Y}^{[i]} - \mu')(\mathcal{Y}^{[i]} - \mu')^T \tag{9.26}$$

It can be shown (Julier & Uhlmann, 1997) that the mean and covariance of X as well as those of Y are preserved up to the second order, and further (Julier & Uhlmann, 1996) that higher orders are partially captured as well.

The UKF Algorithm

The prediction step of the UKF algorithm is now fairly straightforward. It takes the previous belief represented by its mean μ_{t-1} and covariance Σ_{t-1} and a control u_t as input, calculates the set of sigma points, applies the unscented transform to the state belief with $f = g$ and adds the process noise to the covariance matrix as follows (Thrun et al., 2005a, Sec. 3.4.2):

$$\mathcal{X}_{t-1} = (\mu_{t-1} \qquad \mu_{t-1} + \sqrt{(n+\lambda)\Sigma_{t-1}} \qquad \mu_{t-1} - \sqrt{(n+\lambda)\Sigma_{t-1}}) \tag{9.27}$$

$$\bar{\mathcal{X}}_t^* = g(u_t, \mathcal{X}_{t-1}) \tag{9.28}$$

$$\bar{\mu}_t = \sum_{i=0}^{2n} w_m^{[i]} \bar{\mathcal{X}}_t^{*[i]} \tag{9.29}$$

$$\bar{\Sigma}_t = \sum_{i=0}^{2n} w_m^{[i]} (\bar{\mathcal{X}}_t^{*[i]} - \bar{\mu}_t)(\bar{\mathcal{X}}_t^{*[i]} - \bar{\mu}_t)^T + R_t \tag{9.30}$$

The UKF correction step first calculates the new set of sigma points, propagates it through the measurement model to gain the sigma points corresponding to the expected measurement distribution, and recovers its mean $\hat{z}_t$ and covariance S_t (including measurement noise). Similarly, the cross-covariance $\bar{\Sigma}_t^{x,z}$ between state and expected measurement is calculated.

$$\begin{aligned}
\bar{\mathcal{X}}_t &= (\bar{\mu}_t \qquad \bar{\mu}_t + \sqrt{(n+\lambda)\bar{\Sigma}_t} \qquad \bar{\mu}_t - \sqrt{(n+\lambda)\bar{\Sigma}_t}) && (9.31)\\
\bar{\mathcal{Z}}_t &= h(\bar{\mathcal{X}}_t) && (9.32)\\
\hat{z}_t &= \sum_{i=0}^{2n} w_m^{[i]} \bar{\mathcal{Z}}_t^{[i]} && (9.33)\\
S_t &= \sum_{i=0}^{2n} w_c^{[i]} (\bar{\mathcal{Z}}_t^{[i]} - \hat{z}_t)(\bar{\mathcal{Z}}_t^{[i]} - \hat{z}_t)^T + Q_t && (9.34)\\
\bar{\Sigma}_t^{x,z} &= \sum_{i=0}^{2n} w_c^{[i]} (\bar{\mathcal{X}}_t^{[i]} - \bar{\mu}_t)(\bar{\mathcal{Z}}_t^{[i]} - \hat{z}_t)^T && (9.35)
\end{aligned}$$

The latter two are then used to compute the Kalman gain which determines how the innovation is to be used to update the mean and how much uncertainty can be removed from the covariance matrix to reflect the information gained from the measurement.

$$\begin{aligned}
K_t &= \bar{\Sigma}_t^{x,z} S_t^{-1} && (9.36)\\
\mu_t &= \bar{\mu}_t + K_t(z_t - \hat{z}_t) && (9.37)\\
\Sigma_t &= \bar{\Sigma}_t - K_t S_t K_t^T && (9.38)
\end{aligned}$$

9.5.2 Parameter Choices and Stability

First of all we have yet to clarify the **matrix square root** $\sqrt{A}$ which is used to compute the sigma points. Interestingly, Thrun et al. (2005a) do no specify it at all. Julier & Uhlmann (1996) show that any matrix square root of the form

$$\sqrt{A}\sqrt{A}^T = A \tag{9.39}$$

can be used[4] but specifically suggest the Cholesky decomposition for its stability and efficiency.

Experience gained while working on this thesis strongly suggests that the way the Cholesky decomposition implementation deals with numerically problematic cases is crucial for the filter to function properly (in the sense of providing good estimation results) and for its numerical stability. The author originally used the respective implementation from Eigen[5] 2.0 which often lead to poor filter performance apparently due to a degraded covariance matrix and occasional decomposition failures where the library claimed the covariance matrix was not positive definite although inspection of the eigenvalues suggested otherwise. The exact cause has not been investigated since replacing the problematic library calls with a custom wrapper around LAPACK's DPOTRF Cholesky decomposition routine[6] immediately solved the issue. Correspondence on the Eigen mailing list[7] indicates an implementation error in Eigen 2.0 as the source of the problem.

[4]Julier & Uhlmann (1996) also note that a square root of the form $\sqrt{A}^T\sqrt{A} = A$ can be used with minor modifications to the algorithm.

[5]`http://eigen.tuxfamily.org`

[6]`http://www.netlib.org/lapack/double/dpotrf.f`

[7]See the thread "Precision in Cholesky Decomposition" starting at `http://listengine.tuxfamily.org/lists.tuxfamily.org/eigen/2009/06/msg00007.html`. Last checked: 2009-11-27.

In this context it is worth noting that factorized square-root UKF variants exist (van der Merwe & Wan, 2001a,b) which avoid the need for decomposition at each time step. Use of these alternative formulations was considered when the problem described above arose but this idea was discarded after the switch to LAPACK's DPOTRF routine since the factorized UKF variants are less well understood than the standard UKF and we do not rely on the performance gain they bring about.

As for the **scaling parameters** α, β and κ, these allow the sigma point propagation to be tuned for higher order properties of the represented distribution and van der Merwe (2004) suggests that their optimal values are problem specific and no universially optimal parameter set exists. For the purposes of this thesis the author aims for a safe but not necessarily optimal scaling parameter set. α and β were introduced with the *scaled unscented transform* by Julier (2002). Choosing $\alpha = 1$ and $\beta = 0$ yields the same formulae as in the original *unscented transform* und thus the same performance as the original UKF.

The remaining parameter κ is crucial for the stability of the filter. Setting $\alpha = 1$ and $\beta = 0$ gives:

$$\lambda = \alpha^2(n+\kappa) - n = \kappa \tag{9.40}$$

and thus the following weights

$$\begin{aligned} w_c^{[0]} &= \frac{\lambda}{n+\lambda} + (1-\alpha^2+\beta) && = \frac{\kappa}{n+\kappa} \\ w_c^{[i]} &= \frac{1}{2(n+\lambda)} && = \frac{1}{2(n+\kappa)} \text{ for } i = 1,\dots,2n. \end{aligned} \tag{9.41}$$

i.e., if $\kappa < 0$ there is at least one $w_c^{[i]} < 0$. Now recall that these weights are used to recover the covariance matrix in the unscented transform:

$$\Sigma' = \sum_{i=0}^{2n} w_c^{[i]}(\mathcal{Y}^{[i]} - \mu')(\mathcal{Y}^{[i]} - \mu')^T, \tag{9.42}$$

i.e., if there is any $w_c^{[i]} < 0$ it is no longer guaranteed that Σ' is positive semidefinite. Thus, κ must not be negative or the attempt to apply the Cholesky decomposition (to the non-positive semidefinite covariance matrix) in the next timestep will fail. A workaround for this exists (Julier & Uhlmann, 1997) in the form of an alternative method of computing the covariance matrix but this has other drawbacks.

For an exact Gaussian state belief Julier & Uhlmann (1997) suggest choosing κ based on the heuristic $n + \kappa = 3$. In our case n is 18 so that we cannot use this heuristic without κ being negative. Thus, for the purposes of this thesis we follow the same strategy as Kurlbaum (2007) and Frese & Schröder (2008) by setting $\kappa = \frac{1}{2}$ when it comes to $w_m^{[\bullet]}$ and using an altogether different $w_c^{[\bullet]}$ such that:

$$w_m^{[i]} = \frac{1}{2n+1} \quad \text{for } i = 0,\dots,2n \tag{9.43}$$

$$w_c^{[i]} = \frac{1}{2} \quad \text{for } i = 0,\dots,2n \tag{9.44}$$

This has the effect that all sigma points have the same weight in the computation of the mean and that the covariance is not scaled dependent on the size n of the state vector.

As a final note on filter stability, it is worth pointing out that the correction step formulation by Thrun et al. (2005a) was chosen since although mathematically equivalent it turned out to be numerically more stable than the one presented by Frese & Schröder (2008). In particular, it appears to be advantageous how K_t is multiplied to the left and to the right of S_t in (9.38).

9.6 Extension of the UKF to Manifolds

Julier & Uhlmann (1997) claim that since *"[...] [t]he mean and covariance are calculated using standard vector and matrix operations [...] the [UKF] algorithm is suitable for any choice of process model"*. While the process model is indeed not a problem, this statement misses an important point: The operations that calculate the mean (9.25) and covariance (9.42) operate on *states* and, since they are implemented by means of standard vector arithmetic, states must be elements of the Euclidean space $\mathbb{R}^n$ for the results to be meaningful.

Let us illustrate this with an example. Suppose we want to estimate the orientation of a robot operating in a planar environment, i.e. an orientation in two dimensions or more formally an element of the special orthogonal group $SO(2)$. For the above UKF formulation to be applicable we would need to find a state representation such that among other requirements the mean of a set of states $x_0, \dots, x_k$ where $x_i \in \mathbb{R}^n$ for $i = 0, \dots, k$ can be calculated as follows:

$$\hat{x} = \frac{1}{k} \sum_{i=0}^{k} x_i \tag{9.45}$$

Intuitively, it is clear that this will fail: If we choose a representation where $x_i \in \mathbb{R}^1$ then each orientation is represented by a single real value, an angle φ. However, the filter does not know about the periodicity of this representation, i.e., that $\varphi \in [-\pi, \pi)$ with the respective "wraparounds" that may occur when a state is changed and the extra care needed to calculate differences between states. For higher-dimensional representations (e.g., $x_i = \binom{\cos\varphi}{\sin\varphi}$) additional constraints must be met (e.g., $|x_i| = 1$) to ensure normality and the filter does not know about these either.

With orientations in the three-dimensional (3D) space, or more formally elements of $SO(3)$, the situation is similar. In fact, it can be shown (Frese & Schröder, 2008) that a singularity-free representation of $SO(3)$ with just 3 parameters does not exist. Additional parameters would introduce constraints that need to be respected since a 3D orientation has only three degrees of freedom. Sound representations include rotation matrices (9 parameters) and unit quaternions (4 parameters), both of which are constrained by orthonormality or unit constraints respectively.

While it is possible to work around these issues by means of re-normalization after each time step or other application specific modifications to the filter algorithm (see, e.g., Abbeel (2008)) this is a cumbersome and (depending on the type of robot controlled based on the state estimate) potentially dangerous undertaking. Clearly, it would be desirable to derive a generic UKF formulation that is able to work with sound representations of states involving $SO(2)$ or $SO(3)$. Kraft (2003) suggests to use the concept of (Riemannian) manifolds from topology and differential geometry to allow a UKF to operate on quaternions. Kurlbaum (2007) and Frese & Schröder (2008) generalize this approach to a UKF that operates on arbitrary manifolds and is thus able to handle more complex topological spaces including but not limited to $SO(2)$ and $SO(3)$.

9.6.1 Manifold Operators

We will focus on the general idea behind manifolds and how they are used in the UKF formulation which is the basis of the filter implementation used on the SAMSMobil here. For a more formal discussion of manifold state representations within the context of least squares state estimation see the thesis by Hertzberg (2008).

We have seen above that the original UKF operates on the Eucledian space $\mathbb{R}^n$. An analysis of the UKF formulae also reveals that all operations are inherently local, i.e., they consider differences and apply changes in a local environment around the mean. This is where manifolds come into play. An n-manifold is locally homeomorphic to $\mathbb{R}^n$, i.e., informally speaking[8] we can establish a bi-directional mapping from a local neighborhood (a so called *local chart*) in a n-manifold to $\mathbb{R}^n$. Now, the idea is to use a manifold to represent a state while the UKF only "sees" a mapped part of the manifold in $\mathbb{R}^n$ at any point in time.

Kurlbaum (2007) and Frese & Schröder (2008) suggest to implement this mapping by means of two operators

$$\begin{aligned} \boxplus &: \mathcal{M} \times \mathbb{R}^n \to \mathcal{M} \qquad \text{and} && (9.46)\\ \boxminus &: \mathcal{M} \times \mathcal{M} \to \mathbb{R}^n, && (9.47) \end{aligned}$$

where $\mathcal{M}$ is an n-manifold and $\boxplus$ and $\boxminus$ must satisfy:

$$m_1 \boxplus (m_2 \boxminus m_1) = m_2, \tag{9.48}$$

where $m_1, m_2 \in \mathcal{M}$.

In unpublished work, Hertzberg et al. (2009) also demand that

$$(m_1 \boxplus \delta) \boxminus \delta = m_1 \tag{9.49}$$

and

$$m \boxplus 0 = m, \tag{9.50}$$

where $m, m_1, m_2 \in \mathcal{M}$. This is in line with our findings in section 9.8.3 on page 79.

Also, the second argument $\delta \in \mathbb{R}^n$ to $\boxplus$ must be sufficiently small to avoid "dropping off" the *local chart* and the effect of $\boxplus$ should grow with δ such that a small δ leads to a small change in the manifold and a larger δ leads to a larger change respectively. Similarly, $\boxminus$ should not be passed arguments that are too distant from each other, and its return value should also smoothly grow with the distance of its arguments in the manifold. Both of these requirements can be summarized by saying that the mapping established by $\boxplus$ and $\boxminus$ is only locally well-defined and must be locally non-singular.

$SO(2)$ and $SO(3)$ can both be treated as manifolds and appropriate operators can be defined. This also applies to $\mathbb{R}^n$ where the operators are, trivially, the well-known vector operators.

9.6.2 The Unscented Transform on Manifolds

To finally arrive at the generic UKF we need a formulation of the unscented transform that operates on manifold distributions. A formal derivation would require the definition of several concepts from probability theory for use with manifold random variables which is given by Frese & Schröder (2008). For our purposes it will suffice to say that we represent a Gaussian distribution of an n-manifold random variable as a mean $\mu \in \mathcal{M}$, where $\mathcal{M}$ is the n-manifold, and a covariance matrix $\Sigma \in \mathbb{R}^{n \times n}$ expressed relative to the mean.

With this in mind the manifold sigma points can be computed as follows:

$$\begin{aligned} \mathcal{X}^{[0]} &= \mu \\ \mathcal{X}^{[i]} &= \mu \boxplus \left(\sqrt{\Sigma}\right)_{\bullet i} \quad \text{for } i = 1, \ldots, n \\ \mathcal{X}^{[i+n]} &= \mu \boxplus -\left(\sqrt{\Sigma}\right)_{\bullet i} \quad \text{for } i = 1, \ldots, n \end{aligned} \tag{9.51}$$

[8]See (Lee, 2003) for a comprehensive formal definition.

These are propagated through the process or measurement model f:

$$\mathcal{Y}^{[i]} = f(\mathcal{X}^{[i]}) \text{ for } i = 0, \ldots, 2n, \tag{9.52}$$

where $f : \mathcal{M}_1 \to \mathcal{M}_2$ operates on an n-manifold argument and returns an element of an m-manifold. Note that in the case of the process model $m = n$ and $\mathcal{M}_1$ and $\mathcal{M}_2$ are identical.

The mean μ' of the transformed distribution can no longer be computed by pure vector arithmetic. Instead, we use the following recursive definition suggested by Frese & Schröder (2008):

$$\mu'_0 = \mathcal{Y}^{[0]} \tag{9.53}$$

$$\mu'_{k+1} = \mu'_k \boxplus \frac{1}{2n+1} \sum_{i=0}^{2n} \mathcal{Y}^{[i]} \boxminus \mu'_k \tag{9.54}$$

$$\mu' = \lim_{k\to\infty} \mu'_k \tag{9.55}$$

The underlying idea is that μ' should minimize the sum of the distances (squared errors in the one-dimensional case) to all $\mathcal{Y}^{[i]}$, i.e.,

$$\mu' = \underset{m\in\mathcal{M}_2}{\operatorname{argmin}} \sum_{i=0}^{2n} \left\| \mathcal{Y}^{[i]} \boxminus m \right\| \tag{9.56}$$

Note that in practice the limit in (9.55) can be computed by an iterative loop that is terminated if the norm of the most recent summed error vector is below a certain threshold. In the following we will use $\textsc{MeanOfSigmaPoints}(\mathcal{Y}, \boxplus, \boxminus)$ to denote this approximation of (9.55).

The covariance Σ' of the transformed distribution is obtained by replacing the vector operator $-$ with $\boxminus$ such that:

$$\Sigma' = \frac{1}{2} \sum_{i=0}^{2n} (\mathcal{Y}^{[i]} \boxminus \mu')(\mathcal{Y}^{[i]} \boxminus \mu')^T \tag{9.57}$$

Note that the formulae for the manifold variants of μ' and Σ' in this section already include weights which in the UKF formulation by Thrun et al. (2005a) would be equivalent to $w_m^{[i]} = \frac{1}{2n+1}$ and $w_c^{[i]} = \frac{1}{2}$ for $i = 0, \ldots, 2n$ (c.f. section 9.5.2 on page 69). Furthermore, we dropped the scaling of the covariance matrix square root. Both modifications are due to Frese & Schröder (2008).

9.6.3 The Manifold-UKF

For our formulation of the manifold-UKF we use the overall algorithmic framework as presented by Thrun et al. (2005a). States are represented by manifolds as suggested by Frese & Schröder (2008) such that the UKF is only allowed to access a state through the manifold operators $\boxplus$ and $\boxminus$ and otherwise treats it as a black box. A further modification specific to our formulation is that we also represent measurements by manifolds. This will allow us to use orientations as measurements which would otherwise cause the same type of problems as outlined above for states.

Thus, the complete manifold-UKF algorithm takes as input the previous belief represented by its mean $\mu_{t-1} \in \mathcal{M}_1$ and covariance $\Sigma_{t-1} \in \mathbb{R}^{n\times n}$, a control[9] $u_t \in \mathbb{R}^n$ and a measurement $z_t \in \mathcal{M}_2$, where

[9]In practice there are no restrictions on the type of the control u_t. The algorithm does not access it directly – it simply needs to be compatible with the process model g.

$\mathcal{M}_1$ is an n-manifold with operators $\boxplus_1$ and $\boxminus_1$ and $\mathcal{M}_2$ is an m-manifold with operators $\boxplus_2$ and $\boxminus_2$. The algorithm further requires a process model $g : \mathbb{R}^n \times \mathcal{M}_1 \to \mathcal{M}_1$ with associated process noise represented by its covariance $R_t \in \mathbb{R}^{n\times n}$ and a measurement model $h : \mathcal{M}_1 \to \mathcal{M}_2$ with associated measurement noise covariance $Q_t \in \mathbb{R}^{m\times m}$.

The prediction step calculates the predicted mean $\bar{\mu}_t \in \mathcal{M}_1$ and covariance $\bar{\Sigma}_t \in \mathbb{R}^{n\times n}$ as follows:

$$\begin{aligned}
\mathcal{X}_{t-1} &= (\mu_{t-1} \qquad \mu_{t-1} \boxplus_1 \sqrt{\Sigma_{t-1}} \qquad \mu_{t-1} \boxplus_1 -\sqrt{\Sigma_{t-1}}) && (9.58)\\
\bar{\mathcal{X}}_t^* &= g(u_t, \mathcal{X}_{t-1}) && (9.59)\\
\bar{\mu}_t &= \textsc{MeanOfSigmaPoints}(\bar{\mathcal{X}}_t^*, \boxplus_1, \boxminus_1) && (9.60)\\
\bar{\Sigma}_t &= \frac{1}{2}\sum_{i=0}^{2n} (\bar{\mathcal{X}}_t^{*[i]} \boxminus_1 \bar{\mu}_t)(\bar{\mathcal{X}}_t^{*[i]} \boxminus_1 \bar{\mu}_t)^T + R_t && (9.61)
\end{aligned}$$

The correction step to compute the new mean $\mu_t \in \mathcal{M}_1$ and covariance $\Sigma_t \in \mathbb{R}^{n\times n}$ is then:

$$\begin{aligned}
\bar{\mathcal{X}}_t &= (\bar{\mu}_t \qquad \bar{\mu}_t \boxplus_1 \sqrt{\bar{\Sigma}_t} \qquad \bar{\mu}_t \boxplus_1 -\sqrt{\bar{\Sigma}_t}) && (9.62)\\
\bar{\mathcal{Z}}_t &= h(\bar{\mathcal{X}}_t) && (9.63)\\
\hat{z}_t &= \textsc{MeanOfSigmaPoints}(\bar{\mathcal{Z}}_t, \boxplus_2, \boxminus_2) && (9.64)\\
S_t &= \frac{1}{2}\sum_{i=0}^{2n} (\bar{\mathcal{Z}}_t^{[i]} \boxminus_2 \hat{z}_t)(\bar{\mathcal{Z}}_t^{[i]} \boxminus_2 \hat{z}_t)^T + Q_t && (9.65)\\
\bar{\Sigma}_t^{x,z} &= \frac{1}{2}\sum_{i=0}^{2n} (\bar{\mathcal{X}}_t^{[i]} \boxminus_1 \bar{\mu}_t)(\bar{\mathcal{Z}}_t^{[i]} \boxminus_2 \hat{z}_t)^T && (9.66)\\
K_t &= \bar{\Sigma}_t^{x,z} S_t^{-1} && (9.67)\\
\mu_t &= \bar{\mu}_t \boxplus_1 K_t(z_t \boxminus_2 \hat{z}_t) && (9.68)\\
\Sigma_t &= \bar{\Sigma}_t - K_t S_t K_t^T && (9.69)
\end{aligned}$$

9.7 Generic Manifold-UKF Implementation

The author implemented a generic manifold-UKF as a C++ template class with the help of the Eigen 2.0 matrix library and as noted above the Cholesky decomposition routine DPOTRF from LAPACK. Eigen provides overloaded operators based on expression templates so that the code is a fairly verbatim transcription of the formulae from section 9.6.3 on the preceding page into C++. In the following we will introduce the API and the requirements on the state representation.

9.7.1 API

The generic manifold-UKF implementation comes in the form of a header-only library with the API shown in Listing 9.1 on the next page. The basic idea is that one `ukf::ukf` instance is used per estimated state (μ, Σ) where states can be of an arbitrary manifold type that is passed as a template parameter.

Manifold types must provide a few members for the template instantiation to work as illustrated in Listing 9.2 on page 76. Specifically, the generic UKF code needs to know the type of scalars (::

Listing 9.1: Manifold-UKF API

```cpp
namespace ukf {

static bool accept_any_mahalanobis_distance(const double &mahalanobis2)
{
  return true;
}

template <typename state>
class ukf {
  enum {
    n = state::DOF
  };

public:
  typedef typename state::vectorized_type vectorized_state;
  typedef Matrix<typename state::scalar_type,
                 state::DOF,
                 state::DOF> cov;
  typedef std::vector<state,
                      Eigen::aligned_allocator<state> > state_vector;

  // constructor
  ukf(const state &mu,
      const cov &sigma);

  // prediction step
  template<typename ProcessModel,
           typename ProcessNoiseCovariance>
  void predict(ProcessModel g, ProcessNoiseCovariance R);

  // manifold measurement variant of measurement update step
  template<std::size_t measurement_rows,
           typename Measurement,
           typename MeasurementModel,
           typename MeasurementNoiseCovariance>
  void update(const Measurement &z,
              MeasurementModel h,
              MeasurementNoiseCovariance Q);

  // vector measurement variant
  template<std::size_t measurement_rows,
           typename MeasurementModel,
           typename MeasurementNoiseCovariance,
           typename MahalanobisTest>
  void update(const Matrix<typename state::scalar_type, measurement_rows, 1> &z,
              MeasurementModel h,
              MeasurementNoiseCovariance Q,
              MahalanobisTest mt);

  // accessors
  const state &mu() const;
  const cov &sigma() const;
};

} // namespace ukf
```

Listing 9.2: Minimum members to be provided by a manifold C++ type.

```
class manifold
{
public:
  typedef scalar scalar_type;

  enum {
    DOF = n
  };

  typedef Matrix<scalar_type, DOF, 1> vectorized_type;

  // manifold operator [+]
  manifold& operator+=(const Matrix<scalar,DOF,1> &delta);

  manifold operator+(const Matrix<scalar,DOF,1> &delta) const;

  // manifold operator [-]
  vectorized_type operator-(const manifold &other) const;
};
```

`scalar_type`) the dimension n of the mapped Euclidean space $\mathbb{R}^n$ (`::DOF`[10]) and the type used to represent $\mathbb{R}^n$ itself (`::vectorized_type`). Furthermore, the manifold operators must be provided in the form of `operator+=`, `operator+` and `operator-`. Note that `operator+=` can be implemented in terms of `operator+` or vice versa.

The prediction step is implemented by `ukf::ukf<state>::predict()` which takes two templated functors as arguments, a `ProcessModel g` and a `ProcessNoiseCovariance R`. A `ProcessModel` must be callable taking a `state` manifold as the only argument and return the new state after application of the process model. This may seem surprising given the fact that in (9.59) g also takes a control u_t as an argument. Note, however, that the u_t passed to g is always the same and passed to the UKF from the outside. In the C++ implementation we simply require that u_t be managed by the `ProcessModel` functor internally which has the advantage that controls of arbitrary type can be used.

The correction or measurement update step is implemented by `ukf::ukf<state>::update()` which comes in two overloaded variants: one, more general one, for manifold measurements and a second one contraining an optimized code path for vector measurements. The parameters are handled in a fashion similar to `predict()` with the important exception that this time the generic UKF code needs access to the measurement directly. Furthermore, the vector variant of `::update()` takes an additional functor argument which can be used to reject measurements z_t based on their squared Mahalanobis distance (Mahalanobis, 1936) which is computed as follows which is passed as an argument to the functor:

$$d^2 = (z_t \boxminus_2 \hat{z}_t) S_t^{-1} (z_t \boxminus_2 \hat{z}_t)^T, \tag{9.70}$$

where $z_t \in \mathcal{M}_2$ with corresponding manifold operator $\boxminus_2$ and the expected measurement $\hat{z}_t$ and the measurement covariance matrix S_t have been defined in equations (9.64) and (9.65) respectively.

[10]Historically, manifolds were only used to represent states so that n correponds to the degrees of freedom (DOF) visible to the filter. This was kept even though measurements are now also allowed to be represented by manifolds.

A default implementation that accepts any Mahalanobis distance is provided in the form of `ukf::accept_any_mahalanobis_distance()`.

Finally, use of the generic manifold-UKF works as follows: The user calls `ukf::ukf<state>::predict()` once for each prediction step, with the control u_t embedded in the process model functor which can be different at different time steps. Similarly, `ukf::ukf<state>::update()` is called every time a measurement is to be fused into the combined state estimate. Multiple invocations of `ukf::ukf<state>::update()` at each time step with different measurement types and models are possible (and required to fuse measurements from different sensors as we will see later). It is recommended that the generic manifold-UKF is used in conjunction with *boost::bind* to create anonymous functors on the fly.

9.7.2 Implementation Notes

As far as the implementation behind the above API is concerned, the author would like to acknowledge the inspiration drawn from Java code of a reference implementation provided for use in exercises of the course by Frese & Schröder (2008) which we used in a previous project (RescueRobotics Project, 2007) as the basis of a different C/C++ implementation of a UAV pose tracking filter.

Although the general structure proposed by Frese & Schröder (2008) was kept, the new implementation behind the above API started from a clean slate since implementation errors had been suspected behind problems which turned out to be caused by numerical instabilities (c.f. section 9.5.2 on page 69) and issues related to the deeper structure of the manifold state representation (c.f. section 9.6 on page 71 and, specifically, section 9.8.3 on page 79). Unique to the new implementation is the clean API made possible by the application of modern C++ concepts (templated functors, *boost::bind*) and the expression templates based highly efficient Eigen matrix library, the generalization to arbitrary state and measurement manifolds via heavy use of C++ templates, the use of a Cholesky decomposition based on LAPACK's DPOTRF (c.f. section 9.5.2 on page 69)), and the Mahalanobis distance acceptance checks.

9.8 Manifold-UKF-Based Localization for the SAMSMobil

Now that we have the generic manifold-UKF as a state estimation framework it is time to discuss how it is used on the SAMSMobil. We will start with the design of the overall filter architecture and then move on to the state representation and the process and measurement models required to implement it. Finally, the calibration and initialization phases and the C++ implementation will be described.

9.8.1 Filter Design

The mapping and planner components require an estimate of the vehicle pose in six degrees of freedom (6DOF). An analysis of what type of information the available sensors can contribute to this reveals the following[11]:

- The wheel encoders can fairly accurately capture the momentary translational velocity of the vehicle while due to wheel slip they only poorly describe the rotational velocity. Note that this information is inherently two-dimensional, i.e., the robot motion is described relative to the ground surface.

[11]The theory of operation of the respective sensors is covered in detail in chapter 6 on page 30.

- The accelerometers measure both static and dynamic acceleration, i.e., on the robot they will measure the sum of acceleration due to the Earth's gravitational field and due to ego-motion of the robot.

  The position can be deduced from the second integral of the three-dimensional (3D) acceleration vector with respect to time given the starting position and velocity. To get the position in world coordinates, precise knowledge of the orientation of the robot in world coordinates is also required. Note that for this to yield a position estimate which is useful over a longer period of time the accelerometer measurements must be very accurate which in general is not the case with (inexpensive) MEMS accelerometers.

  Conversely, while the robot is at rest accelerometers will measure acceleration pointing in the opposite direction of the Earth's gravity vector. This provides clues on orientation in world coordinates but does not cover orientation about an Earth-fixed z-axis.
- Integration with respect to time of the angular rate measured by a three-axis gyroscope yields the 3D orientation (given knowledge of the initial orientation). Errors accumulate, too, although the effect is less dramatic since it is a single integration.
- Magnetometers can be used to deduce the orientation of the robot with respect to the Earth's magnetic field which is, however, distorted due to the influence of nearby (ferro-)magnetic materials.
- GPS receivers actually measure the time of flight of a radio signal from a set of low-Earth-orbit satellites to the receiver. However, we only have access to the output of a receiver internal Kalman filter which can theoretically provide a 3D position estimate and, while in motion, a 3D velocity and orientation estimate (Doppler effect).

The latter has by far the most important effect on the filter design: It forces us to operate the overall filter in a loosely coupled setup, i.e., we use the output of the GPS receiver-internal filter (which is beyond our control) in a stacked second filter which additional incorporates data from the other sensors. The stacked filter has no influence on the performance of the GPS receiver-internal filter while in a tightly coupled setup the information gained from additional sensory input could be used to improve the GPS performance.

There are still several ways in which the available sensor data can be fused into a state belief in the loosely coupled setup. For the SAMSMobil the author has decided for the following:

- The process model uses the angular rates from the gyroscopes to adjust the orientation and the average of the left and right wheel encoder ticks to adjust the position of the state.
- The robot is assumed to undergo only minor acceleration such that the measured acceleration vector in world coordinates should approximately match the reversed gravity vector.
- For the remaining rotational degree of freedom around the world z-axis we can use the magnetometer readings assuming they indicate a vector approximately pointing to magnetic north.
- GPS provides measurements of the 2D/3D position and the 2D velocity and orientation in the tangential plane.

Apart from fusing the above sensory information, another measurement model based on virtual range sensors is used which determine the distance to the left and right path boundary from the local obstacle map. Given a predefined map of path boundaries in the environment the robot is deployed in, this extends our filter from a pose tracking to a localization algorithm.

9.8.2 State Representation

As discussed in section 7.1 on page 46 we use a local cartesian coordinate system relative to a reference point on the WGS84 ellipsoid for the world coordinate system and translate the WGS84 coordinates of GPS positioning solutions into this local coordinate system. Given the limited range of the SAMSMobil this is an acceptable approximation which greatly simplifies the filter implementation. Thus, we represent the state as follows:

- A transformation matrix $B2W \in \mathbb{R}^{4\times 4}$ represents the pose of the IMU in metric, cartesian world coordinates.
- A vector $v_w \in \mathbb{R}^3$ contains the 3D velocity vector of the IMU in world coordinates.
- A vector $b_a \in \mathbb{R}^3$ contains the gyroscope bias for each of the three respective axes.
- Similarly, a vector $b_g \in \mathbb{R}^3$ contains the accelerometer bias for each of the three respective axes.
- A vector $o_{gps} \in \mathbb{R}^3$ describes the 3D correction offset representing the expected difference of GPS positioning solutions and the actual robot position.

Note that $B2W$ was chosen to be Imu2World as this greatly simplifies the process model and the output of the pose estimator can be simply be computed as

$$\textsc{Robot2World} = \textsc{Imu2World} \cdot \textsc{Robot2Imu}. \tag{9.71}$$

One could additionally add an odometry scaling factor to the state to model changes in the wheel diameter due to, e.g., different tire pressure. This is useful if high precision GPS data is available but with a consumer-grade GPS receiver it gives the filter the freedom to incorrectly adjust the scaling factor due to poor GPS positioning solutions so that this state parameter is omitted in the SAMSMobil UKF.

9.8.3 State Manifold Operators

The manifold operators establish the mapping from the state manifold $\mathcal{S}$ into $\mathbb{R}^{18}$. For local orientation changes we will use a parameterization by Frese & Schröder (2008) which is based on small rotations in matrix exponential representation in world coordinates. In a matrix exponential representation a rotation is defined by a "scaled axis" vector where the direction of the vector determines the rotation axis and its length the angle to rotate by.

In our implementation the manifold-UKF thus sees local changes $\delta \in \mathbb{R}^{18}$ defined as follows:

$$\delta = \begin{pmatrix} \Delta\omega \\ \Delta p \\ \Delta v \\ \Delta b_a \\ \Delta b_g \\ \Delta o_{gps} \end{pmatrix}, \tag{9.72}$$

where $\Delta\omega \in \mathbb{R}^3$ is a rotation in its angle-axis representation, $\Delta p \in \mathbb{R}^3$ a translation, $\Delta v \in \mathbb{R}^3$ a velocity change, $\Delta b_a, \Delta b_g \in \mathbb{R}^3$ a change in the accelerometer and gyroscope bias respectively, and finally $\Delta o_{gps} \in \mathbb{R}^3$ a change in the GPS correction offset.

The manifold operator $\boxplus_{\mathcal{S}} : \mathcal{S} \times \mathbb{R}^{18} \to \mathcal{S}$ incorporates a $\delta \in \mathbb{R}^{18}$ into a state $s \in \mathcal{S}$ to get a new state $s' \in \mathcal{S}$ as follows:

$$\begin{aligned} B2W' &= \text{trans}(\Delta p) \cdot B2W \cdot \text{Rot}(B2W^{-1} \cdot (\tbinom{\Delta\omega}{0})) && (9.73)\\ v'_w &= v_w + \Delta v && (9.74)\\ b'_a &= b_a + \Delta b_a && (9.75)\\ b'_g &= b_g + \Delta b_g && (9.76)\\ o'_{gps} &= o_{gps} + \Delta o_{gps}, && (9.77) \end{aligned}$$

where $\text{Rot} : \mathbb{R}^3 \to \mathbb{R}^{4\times4}$ implements the matrix exponential by means of the Rodriguez formula (Frese & Schröder, 2008), and $\text{trans} : \mathbb{R}^3 \to \mathbb{R}^{4\times4}$ turns a translation vector into a transformation matrix as follows:

$$\text{trans}(t) = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{9.78}$$

The manifold operator $\boxminus_{\mathcal{S}} : \mathcal{S} \times \mathcal{S} \to \mathbb{R}^{18}$ computes the difference $\delta = s_1 \boxminus s_2$ between two states $s_1 \in \mathcal{S}$ and $s_2 \in \mathcal{S}$ mapped into $\mathbb{R}^{18}$ as follows:

$$\begin{aligned} \Delta\omega &= B2W_2 \cdot \text{aRot}(B2W_2^{-1} \cdot B2W_1) && (9.79)\\ \Delta p &= \begin{pmatrix} (B2W_1)_{1,4} - (B2W_2)_{1,4} \\ (B2W_1)_{2,4} - (B2W_2)_{2,4} \\ (B2W_1)_{3,4} - (B2W_2)_{3,4} \end{pmatrix} && (9.80)\\ \Delta v &= v_{w,1} - v_{w,2} && (9.81)\\ \Delta b_a &= b_{a,1} - b_{a,2} && (9.82)\\ \Delta b_g &= b_{g,1} - b_{g,2} && (9.83)\\ \Delta o_{gps} &= o_{gps,1} - o_{gps,2}, && (9.84) \end{aligned}$$

where $\text{aRot} : \mathbb{R}^{4\times4} \to \mathbb{R}^3$ is the inverse operation of Rot, i.e., $\text{aRot}(Q) = v \Leftrightarrow \text{Rot}(v) = Q \wedge |v| \leq \pi$ (Frese & Schröder, 2008).

The implementation of Rot and aRot is based on a transcription of Java code provided for use in exercises for the course by Frese & Schröder (2008) into C++ with Eigen as the matrix library and contains minor modifications to handle rotation angles exceeding π more gracefully in Rot.

This leads to an interesting topic: Recall that the manifold-UKF generates the state sigma points, e.g., as follows in the prediction step:

$$\begin{aligned} \mathcal{X}_{t-1} &= (\mu_{t-1} \qquad \mu_{t-1} \boxplus_1 \sqrt{\Sigma_{t-1}} \qquad \mu_{t-1} \boxplus_1 -\sqrt{\Sigma_{t-1}}) && (9.85)\\ & && (9.86) \end{aligned}$$

i.e., it passes columns of the covariance matrix square root as the δ to the manifold operator $\boxplus_1$. In our case, when this is done with a single sigma point an orientation wraparound occurs if $|\Delta\omega| > \pi$. Now consider what happens if after a previous predicition or correction step a diagonal entry of Σ_{t-1} corresponding to the z-orientation exceeds π^2: The filter will pass at least one δ with $|\Delta\omega| > \pi$ to $\boxplus_S$ such that after propagating the sigma points through the process model the recovered covariance will likely (depending on the control) be erroneously smaller than Σ_{t-1} which can be illustrated by a simplified example. Let $s_1, s_2 \in \mathcal{S}$ be two states and $|\Delta\omega| > \pi$ then

$$s_1 \boxplus \begin{pmatrix} \Delta\omega \\ \vdots \end{pmatrix} = s_2, \text{ but} \tag{9.87}$$

$$s_2 \boxminus s_1 \neq \begin{pmatrix} \Delta\omega \\ \vdots \end{pmatrix}. \tag{9.88}$$

Under this error condition the unscented transform (and thus the entire filter, see Julier & Uhlmann (1997)) will no longer be consistent, i.e., the covariance will be under-estimated. In our experiments this often lead to filter divergence (c.f. 12.2.2 on page 120)

Note that the fact that the covariance decreases is not a problem as such – this can happen normally given certain controls. The problem is that it happens as a result of a violation of the requirements for $\boxplus_{\mathcal{S}}$ and δ and cannot be explained by the dynamics of the system we model.

We postulated in section 9.6.1 on page 71 that δ should be "sufficiently small". In our concrete case we can now say that the rotation component of δ must satisfy $|\Delta\omega| \leq \pi$. In practice $|\Delta\omega|$ must be well below π to avoid numerical instability.

9.8.4 Process Model

The process model employs a standard 3D odometry approach based on the following assumptions[12]:

- The ground surface is locally planar. This is plausible since the robot operates on relatively smooth paths (designed for use by humans) and the process model runs at 100Hz so that at a maximum translational velocity of $1\frac{m}{s}$[13] the robot can move at most 1cm per time step.
- The forward-translation (in robot-y-direction) of the robot in the plane can be approximated as the average of the distance travelled by the left and right wheels (as deduced from the odometry encoder ticks).
- The 3D angular rate of the robot is approximately constant throughout an update interval of 10ms.

This allows us to first apply half of the rotation, then the averaged forward translation pointing along the new y-axis of the robot coordinate system, and finally the other half of the rotation.

Thus, given a control u_t consisting of the measured angular rate vector $\dot{\omega} \in \mathbb{R}^3$ and the distances travelled by the left and right wheels Δs_l and Δs_r, we can compute the new state x_t from a previous

[12]See, e.g., (Lamon & Siegwart, 2003) for a more sophisticated approach which, however, relies on a specific legged/wheeled robot design.

[13]This was later reduced to $0.5\frac{m}{s}$ for autonomous runs.

state x_{t-1} as follows:

$$\begin{aligned}
\Delta s &= 0.5 \cdot (\Delta s_l + \Delta s_r) && (9.89)\\
v &= \frac{\Delta s}{\Delta t} && (9.90)\\
B\bar{2}W_t &= B2W_{t-1} \cdot \mathrm{Rot}(0.5 \cdot (\dot{\omega} + b_g)\Delta t) && (9.91)\\
B\hat{2}W_t &= \mathrm{trans}(\textsc{Robot2Imu} \cdot \begin{pmatrix} 0 \\ \Delta s \\ 0 \\ 1 \end{pmatrix}) \cdot B\bar{2}W_t && (9.92)\\
v_{w,t} &= B\hat{2}W_t \cdot \textsc{Robot2Imu} \cdot \begin{pmatrix} 0 \\ v \\ 0 \\ 0 \end{pmatrix} && (9.93)\\
B2W_t &= B\hat{2}W_t \cdot \mathrm{Rot}(0.5 \cdot (\dot{\omega} + b_g)\Delta t), && (9.94)
\end{aligned}$$

where trans is overloaded to work with homogeneous coordinates analogously to the definition in (9.78) and Δt is the time between x_{t-1} and x_t, i.e., the update interval of in our case 10 ms. The bias and offset terms are copied unchanged.

Note that the use of ROBOT2IMU is required because $B2W$ is in fact IMU2WORLD and that the somewhat unusual looking update of the velocity vector (which throws away the previous estimate) is harmless for the filter performance since the velocity estimate does not otherwise influence the process model and is not used by any of the measurement models.

As far as the process noise covariance matrix is concerned, we assume that the noise of all state variables is uncorrelated (i.e., all off-diagonal entries are 0) and define R_t as follows:

$$\begin{aligned}
\sigma_\omega &= \mathrm{rad}(1) \cdot \Delta t && (9.95)\\
\sigma_s &= 0.1 \cdot |\Delta s| && (9.96)\\
\sigma_v &= \frac{\sigma_s}{\Delta t} && (9.97)\\
\sigma_o &= |\Delta s| && (9.98)\\
R_t &= diag(\sigma_\omega^2, \sigma_\omega^2, \sigma_\omega^2,\quad \sigma_s^2, \sigma_s^2, \sigma_s^2,\quad \sigma_v^2, \sigma_v^2, \sigma_v^2,\quad 0,0,0,\quad 0,0,0,\quad \sigma_o^2, \sigma_o^2, \sigma_o^2), && (9.99)
\end{aligned}$$

where rad converts degrees to radians.

The rotation noise corresponds to a gyroscope drift rate of about one degree per second. This may seem a bit high but reflects the fact that the gyroscopes experience significant vibration on the SAMSMobil(c.f. section 6.1.2 on page 31). Also note that we set the noise terms for both bias vectors to 0. The filter figures out on its own that both are related to the orientation uncertainty through the process and measurement models and it turned out to be easier to specify the overall uncertainty in a single noise term (σ_ω). With this in mind σ_ω should be plausible, particularly given the low bias stability of the gyroscopes (see the specification by the manufacturer in Table 6.1 on page 32).

The position noise is set to 10% of the (averaged) travelled distance. The author believes this is realistic given the wheel geometry and expected wheel slip. The GPS offset noise σ_o may seem a bit surprising at first. The underlying idea is that the uncertainty in the GPS offset correction grows as the robot moves which reflects the fact that the errors in the GPS positioning solution are dominated by multipath errors which change depending on the position relative to, e.g., nearby buildings (c.f. section 6.3.6 on page 37).

9.8.5 Earth's Gravitational Field Measurement Model

Deriving an expected accelerometer measurement from a state x_t with orientation $B2W_t$ and accelerometer bias vector b_a under the assumption that the robot is approximately at rest is straightforward:

$$h_{acc}(x_t) = \text{start}(B2W_t^{-1} \cdot \begin{pmatrix} 0 \\ 0 \\ 9.81 \\ 0 \end{pmatrix}, 3) - a_b, \tag{9.100}$$

where start extracts the first k elements of a vector.

Obviously, the robot is not actually always at rest. We can model this by means of a correspondingly large noise term

$$\sigma_{acc} \quad = \quad \begin{cases} 0.1 & \text{during initialization} \\ 10 & \text{during temporary halts} \\ 100 & \text{otherwise, i.e., when in motion.} \end{cases} \quad . \tag{9.101}$$

The measurement noise is assumed to be approximately uncorrelated such that the measurement noise covariance $Q_{acc,t}$ is given by:

$$Q_{acc,t} = \text{diag}(\sigma_{acc}^2, \sigma_{acc}^2, \sigma_{acc}^2) \tag{9.102}$$

Additionally, we require that for a measurement to be fused into the state belief its Mahalanobis distance (c.f. (9.70)) be less than 0.1. This value was determined by measuring the Mahalanobis distance in an experiment in which the rear end of the robot was lifted and then dropped by a few centimeters several times as illustrated by the plot in Figure 9.5 on the following page. At the same time the state estimate was used to compute the expected laser range scans due the intersection of the scan plane with the ground plane (c.f. section 10.7.2 on page 104) and compared it to the actual range scans. Without the Mahalanobis distance check the state estimate is "jumpy", with the threshold applied the expected and actual range scans nearly match.

9.8.6 Earth's Magnetic Field Measurement Model

Deriving the expected three-dimensional magnetometer measurement vector from the current state requires an adequate model of the Earth's magnetic field. Such models exist, e.g., in the form of the World Magnetic Model 2005 (McLean et al., 2004). However, the author decided to test the use of magnetometers with a simpler model first.

The direction of the magnetometer measurement vector[14] $z_{mag,t} \in \mathbb{R}^3$ is assumed to point to magnetic north. Thus, magnetometer data is preprocessed by projecting it into the tangential plane[15] and computing its 2D orientation:

$$z'_{mag,t} = \text{orientation2d}(\text{atan2}(z_{mag,t,y}, z_{mag,t,x})), \tag{9.103}$$

where orientation2d instantiates a 2D orientation ($SO(2)$) manifold representation (see below).

[14]Given a proper set of calibration parameters and assuming the local magnetic field is not distorted by nearby (ferro-)magnetic materials the MTi-G should report a unit vector $\in S^2$. In practice, this is, however, rarely the case.

[15]Under normal operation (i.e., without toppling over) the robot will never have an orientation where this operation is problematic.

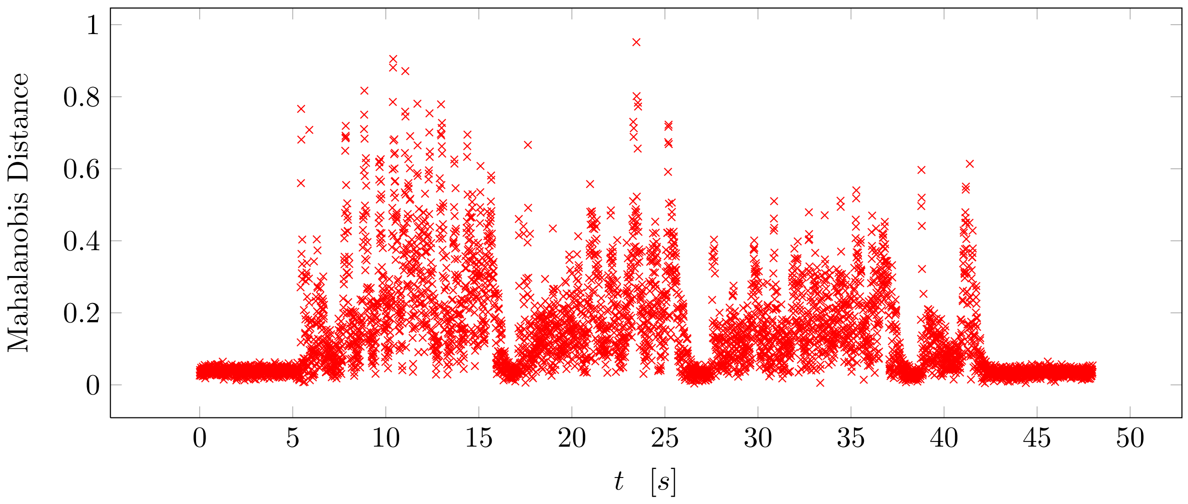


Figure 9.5: Mahalanobis distance of accelerometer measurements. The rear end of the robot was lifted a few centimeters and dropped several times. The resulting large accelerations violate the "at rest" assumption and are to be filtered out.

This has the positive side effect that the magnetometer measurements now cover only the degree of freedom that is not covered by accelerometer measurements such that the influences of the two on the state estimate cannot "battle" each other.

To process these 2D orientation ($SO(2)$) measurements in the manifold-UKF, we use a manifold representation that consists of an orientation angle in radians. The corresponding manifold operator $\boxplus$ adds the rotation contained in a $\delta \in \mathbb{R}^1$ and re-normalizes the orientation to be within $[-\pi; \pi)$. Similarly, $\boxminus$ computes the normalized rotation angle required to get from one 2D orientation to the other.

The measurement model to derive a magnetometer measurement (pre-processed as discussed above) from a state x_t with orientation $B2W_t$ is simply:

$$h_{mag}(x_t) = \text{orientation2d}(\text{atan2}((B2W_t)_{2,2}, (B2W_t)_{1,2})). \tag{9.104}$$

For the measurement noise covariance we use a σ of $15°$ (converted to radians). This is well above the expected precision of the magnetometers but models the fact that (ferro-)magnetic materials (parked cars, etc.) may have an influence on magnetometer measurements.

Since the calibration of the magnetometers turned out to be difficult as discussed in section 7.4 on page 48 the use of a more sophisticated measurement model based on the World Magnetic Model has not been investigated.

9.8.7 GPS Measurement Models

Before we discuss the GPS measurement models we will look at a few specific issues and design decisions related to GPS data.

9.8.8 Fusing Time Delayed Measurements

Strictly speaking, fusion of GPS navigation solutions produced by a receiver-internal Kalman filter in a second, stacked filter requires taking the time delay into account that is introduced by the signal processing and navigation solution calculation process on the receiver. Techniques for the fusion of time delayed measurements exist (van der Merwe et al., 2004) but require non-trivial modifications to the filtering algorithm since the filter has to reconsider past decision akin to backtracking in a search algorithm.

In our case the error introduced by time delayed GPS measurements is neglegible compared to the precision of our consumer-grade GPS receiver. At a maximum velocity of $1 \frac{m}{s}$ even a delay of a full second would only cause an error of a single meter.

9.8.9 Rejecting Erroneous GPS Measurements

Another issue when working with (consumer-grade) GPS receivers relates to errors in the receiver output caused by errors in the GPS signal, particularly due to multipath effects (c.f. section 6.3.6 on page 37). One way to deal with this is to drop GPS measurements which exhibit "unphysical" innovations (Mason et al., 2007) or which differ from the previous GPS measurement by more than a certain threshold (Cremean et al., 2007). This approach was discarded for two reasons:

- The GPS receiver-internal filter uses an internal motion model which effectively low-pass filters the GPS positioning solution, i.e., the errors we see are not large, obviously "unphysical jumps" but gradual although significant changes in the positioning solution (see 12.2.2 on page 121 for an analysis of a dataset illustrating this).
- In lack of usable magnetometers (see calibration issue in section 7.4 on page 48) the orientation and thus position estimate is dominated by GPS which makes it difficult to decide which of these gradual changes are "correct" and which stem from GPS errors.

The filter does, however, ignore GPS measurements with a Mahalanobis distance greater than 2 which is very permissive but ensures that obvious outlier measurements are not fused. The author has experimented with smaller thresholds but this usually leads to filter divergence for the reasons outlined above.

Modelling GPS Errors

The best thing a GPS measurement model can do in our loosely coupled GPS/INS setup (positioning solutions as measurements) is to have $h_{gps}(x_t)$ return the position of x_t and model all errors as part of the noise term δ_t. Recall that we postulated the following requirements for UKF measurements in 9.5.1 on page 67:

$$z_t = h(x_t) + \delta_t \tag{9.105}$$

$$\delta_t \sim \mathcal{N}(0, \Sigma_{\delta t}), \tag{9.106}$$

i.e., any measurement z_t is assumed to be the result of the measurement function $h(x_t)$ affected by additive white Gaussian noise δ_t. The combined effect of all GPS error sources (c.f. section 6.3.6 on page 37) on the position data reported by the receiver, however, clearly violates the additive white Gaussian noise requirement as we will see when looking at GPS data from outdoor experiments in 12.2.2 on page 121.

A significantly better model can only be achieved in a tightly coupled setup where the raw pseudoranges are available as measurements and $h_{gps}(x_t)$ returns the pseudoranges expected in state x_t such that the individual error sources can be taken into account by adequate physical error models in h_{gps} based on parameters also estimated as part of x_t.

While, as will also be obvious from actual GPS data in 12.2.2 on page 121, multipath errors have the most dramatic effect, atmospheric errors are a special case and can be modelled to some extent. Common error models assume that atmospheric effects on GPS signals are constant in a spatially limited area over a relatively short period of time. As discussed in section 6.3.9 on page 39 these assumptions form the basis of the EGNOS DGPS system and we can achieve a similar (but not identical; see below) effect by taking into account that the initial position of the robot is known and can be used to determine an offset by subtracting it from the average of a set of GPS positioning solutions received in that position. Since the conditions leading to errors in the GPS positioning solution change over time we added the correction offset to the state (c.f. section 9.8.2 on page 79).

It should also be noted that this simplistic offset-correction operates in position space whereas EGNOS distributes parameters that are taken into account by the receiver internal software when dealing with pseudoranges. An EGNOS-enabled receiver therefore has to deal with less ambiguity in computing its positioning solution and we cannot replicate this completely by post-processing the positioning solution.

Another problem with this approach is that it deals with the combined effect of all GPS error sources rather than solely with atmospheric effects. Multipath effects at the start position particularly affect the result negatively as we will see when analyzing data from the test scenario in 12.2.2 on page 121. However, the overall filter result will still be better since the offset-correction avoids the situation where the initial GPS measurements completely disagree with the true robot trajectory.

GPS Position Measurements

A 2D and a 3D position measurement model based on GPS data have been developed for the SAMSMobil. 2D GPS positioning solutions from the GPS receiver are first converted from the longitude/latitude format in WGS84 coordinates into local metric cartesian coordinates (c.f. section 7.1 on page 46) to form a 2D GPS measurement. For the 3D case the altitude reported by the GPS receiver is added as the z-ordinate.

With the above in mind, we implement the measurement model as follows. The 2D GPS measurement model basically transforms the GPS receiver position (in IMU coordinates) into world coordinates based on the pose contained in the current x_t, applies the current correction offset also contained in x_t, and cuts off everything but the x- and y-coordinates. The 3D measurement models keeps the full 3D position.

For the measurement noise covariance we use a σ_x and σ_y derived from the HDOP value reported by the GPS receiver as follows: The HDOP value corresponds to a 95% confidence interval which assuming a normal distribution is equal to $2\dot{\sigma}$. However, since the DOP values are based on the geometric configuration of the satellites used by the receiver in the positioning solution (c.f. section 6.3.5 on page 36), we degrade this by an empirically determined factor of 10 and clip the result at a minimum of $\sigma_x = \sigma_y = 5m$. The same is done based on the VDOP value for σ_z in the 3D case.

GPS 2D Orientation Measurements

The magnetic field measurement model and manifold representation from section 9.8.6 on page 83 can trivially be modified to handle GPS 2D Orientation Measurements. In fact, the same C++ code is used apart from a different preprocessing step that translates the mathematically unusual GPS orientation representation (decimal degrees from 0° to 360°, 0° pointing to true north, clockwise increasing) into a standard representation in radians.

For the measurement noise covariance the author tried a σ of 60° (converted to radians) since expected precision of GPS orientation data is significantly smaller. In practice, it turns out that the Navilock GPS receiver only estimates the orientation based on positioning solutions, i.e., orientation measurements lag behind the true robot orientation whenever the vehicle takes turns. Thus, this measurement model was discarded very early on.

9.8.10 Path Boundary Measurement Model

The path boundary measurement model takes inspiration from work by Montemerlo et al. (2008) who use lane marker and curb measurements for 1D-localization perpendicular to a road in an urban environment. In our case there are unfortunately no features which stand out as clearly as the highly reflective yellow lane markers typically found in the US (and in contrast to the Sick laser range finders our ROD4plus does not report remission intensity in the first place).

Instead, we use virtual "path boundary sensors" which are implemented by testing lines in the obstacle map for obstacles starting at the current robot position (using the supercover line approach presented in section 11.3.3 on page 113). The closest obstacle cell found in each line determines an artificially generates range measurement. For each side of the robot we use two of these virtual range sensors with one line pointing in a direction perpendicular to the heading and one line pointing 20° behind the robot as illustrated in Figure 9.6 on the next page . The result from these are combined into one measurement per side. This ensures that measurements cover both the (1D) position perpendicular to the path and the orientation of the robot given a path boundary map.

A path boundary map B is simply as set of k line segments with endpoints $a_i \in \mathbb{R}^2$ and $b_i \in \mathbb{R}^2$.

$$B = \{(a_i, b_i) | i = 1, \dots, k\} \tag{9.107}$$

The measruement model uses one path boundary map per side, i.e., B_l contains path boundaries expected on the left hand side of the robot while it traverses the goal path. B_r contains those on the right hand side respectively. In the tests performed for this thesis, path boundary segments are formed by the transition from the sidewalk to patches of grass or bushes and have been extracted manually from aerial imagery in Google Earth as depicted in Figure 9.7 on the following page and exported as KML files (c.f. section 5.4 on page 24).

Before a virtual range measurement (consisting of two distance values each) is fused into the state belief, we check whether any path boundary segment is in range based on its bounding circle and the current position estimate. This has the effect that the path boundary map defines where the path boundary measurement model is applied. Since our maps contain only a very small amount of boundary segments this is currently implemented by means of a linear search but could be replaced with a tree of bounding circles or another spatial data structure.

If a segment is in range, the UKF correction step is invoked. The measurement model is then provided with both path boundary maps and derives measurements from the current robot pose in x_t by computing the intersection of lines pointing in the same direction as the virtual range sensors with the path

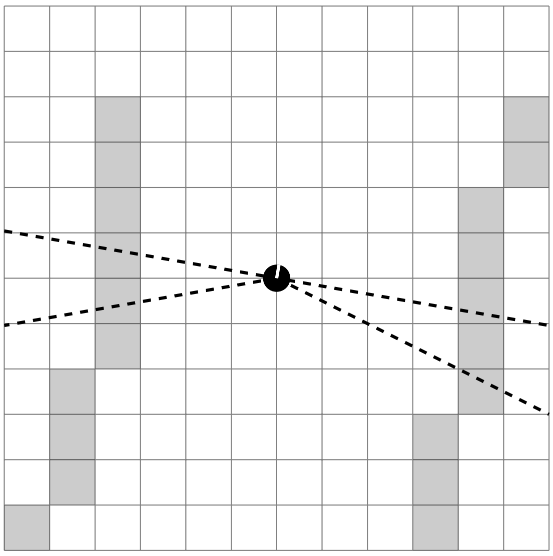


Figure 9.6: The “beams” of the virtual range sensors used to detect path boundaries are illustrated by the four dashed lines. The first obstacle cell intersected by each line determines the range measurement. Note that in the actual implementation the grid has a higher resolution and that the virtual sensor beams are rasterized using the supercover line approach from section 11.3.3 on page 113.

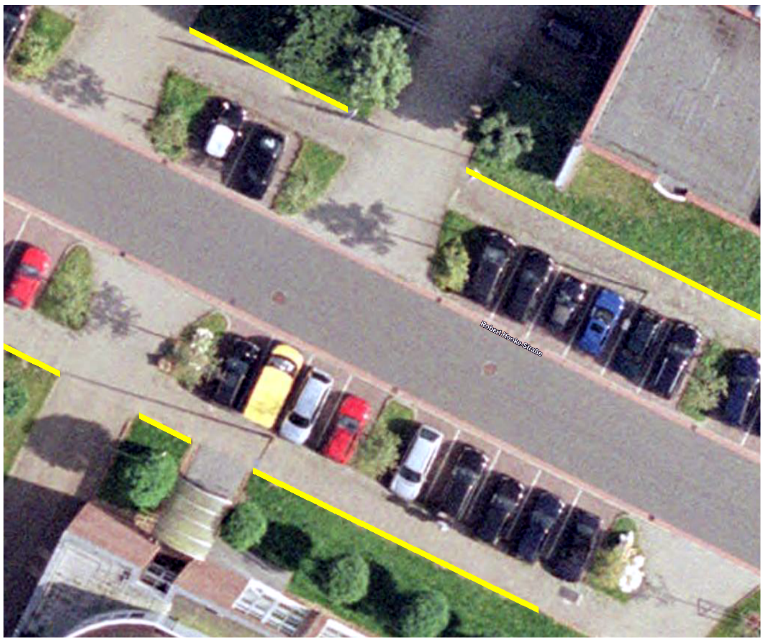


Figure 9.7: Part of the left hand side path boundary map (thick yellow line segments) for our test track in Google Earth.

boundary segments. Measurements for the left and right hand side are handled separately. The intersection code is based on geometry primitives from Eigen and currently computes the intersection with all path boundary segments in a map and chooses the shortest range as the result. Like above, this could be optimized with the help of bounding circles/rectangles and appropriate spatial data structures.

Note that there is one catch with this approach: The measurement model is not only applied to the current mean state estimate but to the complete set of sigma points representing the state distribution. Thus, given a large enough amount of orientation uncertainty there will be some sigma points which result in intersections while others do not. We cannot assign a “default value” for cases where no intersection occurs since the filter would interpret these default values as the correct distance value and erroneously change the state estimate. Consequently, the measurement model is a partial function and we reflect this by throwing a C++ exception if no intersection is found. This leads to the complete measurement being discarded.

The measurement model uses a measurement noise covariance with a σ of 1.5m. This may seem excessively large but models the fact that the path boundary can only be detected with a fairly low precision and resolution (due to the grid discretization). Additionally, we require that the Mahalanobis distance be smaller than 2. This threshold was determined empirically and represents a tradeoff between rejecting outlier measurements and allowing the filter to correct its position after a period with a road boundary segment in range.

As will be illustrated in 12.2.2 on page 128 this approach works well in practice with one exception. Due to the partial measurement model the orientation and position uncertainty must be kept fairly low at all times. After a long period of time without path boundary segments in range the measurement model can no longer be applied as we will see in 12.2.2 on page 128.

Other than that, it might be possible to improve the performance of this measurement model with a more robust implementation of the virtual range sensors. The author has not investigated this further due to time constraints but believes an edge detection algorithm applied to the local obstacle map (interpreted as a 2D image) followed by a line-Hough transform and a segment extraction step could significantly improve the robustness under the assumption that path boundaries are sufficiently long line segments.

9.8.11 Calibration and Initialization Phases

To complete the discussion of the SAMSMobil state estimation/localization filter we still have to look into how it is initialized. This works as follows:

- The initial position is the first point in the goal path passed in as a KML file (c.f. section 5.4 on page 24).
- The initial orientation (Robot2World) is the orientation of the first path segment.
- The initial accelerometer bias terms are all set to 0.
- The covariance matrix is initialized to be all zero except for the diagonal entries which are determined by squaring the σ values in Table 9.1 on the next page.
- Before the filter outputs any pose estimate it goes through a calibration and initialization phase.
- During the calibration phase gyroscope and GPS position measurements samples are collected to determine the initial gyroscope bias and GPS correction offset parts of the initial state estimate. In both cases a minimum amount of samples is required. Additionally, the GPS position samples must be reasonably stable. This handles the problem that on power-up the GPS receiver-internal filter usually outputs solutions although it has not converged yet so that the output exhibit a sometimes significant amount of drift.

Orientation	σ_φ	rad(15°)
	σ_ϑ	rad(15°)
	σ_ψ	0
Position	σ_x	$0m$
	σ_y	$0m$
	σ_z	$0m$
Velocity	σ_{v_x}	$0\frac{m}{s}$
	σ_{v_y}	$0\frac{m}{s}$
	σ_{v_z}	$0\frac{m}{s}$
Accelerometer Bias	$\sigma_{b_{a,x}}$	0.4
	$\sigma_{b_{a,y}}$	0.4
	$\sigma_{b_{a,z}}$	0.4
Gyroscope Bias	$\sigma_{b_{g,x}}$	0.01
	$\sigma_{b_{g,y}}$	0.01
	$\sigma_{b_{g,z}}$	0.01
GPS Correction Offset	σ_{o_x}	$0m$
	σ_{o_y}	$0m$
	σ_{o_z}	$0m$

Table 9.1: Square root of the initial covariance diagonal entries. For reasons of numerical stability all 0 entries are actually very small but non-zero values.

- Next is the initialization phase. The filter starts processing sensor inputs but does not produce any output until its pose estimate has converged.
- During this phase the robot is known to be completely at rest so that the measurement noise σ of the gravity measurement model is set to a comparably small value as discussed in section 9.8.5 on page 83. This ensures quick convergence of the orientation estimate.

Afterwards the filter enters normal operation and outputs state estimates.

9.8.12 Implementation

All of the above measurement models are implemented as free C++ functions which are plugged into the generic manifold-UKF as anonymous functors created with `boost::bind` by a wrapper class which also handles the calibration and initialization phases. The main pose estimator class feeds data received via Isil into this wrapper class and also forwards configuration options set on the command line to, e.g., determine the set of active measurement models. The wrapper class provides access to the current mean and covariance which the pose estimator publishes via Isil.

Chapter 10

Mapping

We have previously encountered the concept of a map within the context of localization in section 9.1 on page 60 where knowledge of a map helps the robot localize itself. In this chapter we will focus on the process of generating a map. We will start with a definition of the problem to be solved, then look at tools required in a mapping implementation, discuss the SAMSMobil implementation of the Probabilistic Terrain Analysis (PTA) mapping approach by Thrun et al. (2006b) and the development of a new, alternative approach, and finally look into how information from the map is propagated to other parts of the SAMSMobil software stack.

10.1 Problem Statement

As discussed in section 9.2 on page 63 we are not employing a SLAM approach. Instead, the robot trajectory is assumed to be known from the state estimation which means that within the probabilistic framework by Thrun et al. (2005a) we perform mapping with known poses, i.e., we aim to estimate the maximum likelihood map given all states (poses) and measurements up to the current ones (Thrun et al., 2005a, Sec. 9.2):

$$p(m|z_{1:t}, x_{1:t}) \tag{10.1}$$

The corresponding Bayesian network is depicted in Figure 10.1 on the next page. Note that the controls u_t have no influence since the sequence of states is already known.

The map m is often assumed to be a global map. In our application we are only interested in a local map that allows us to avoid obstacles and stay on the sensed path. Thus the local map only covers a certain area around the current robot position.

We further work with a static world assumption, i.e., we assume that the environment does not change while it is observed by the robot. Additionally, we aim to generate a map that is useful in practice but not necessarily strictly safe. In particular, we treat unseen terrain as drivable. This is necessary because due to the limited scan rate of the laser range finder (25Hz) and environment-induced tilting of the sensor the robot would have to be restricted to a very slow maximum velocity to avoid gaps in the map.

10.2 Smooth Coordinates

A problem with the mapping approach outlined in the previous section may arise whenever there are "unphysical" jumps in the pose estimate causing distortions in the map. In our setup these would typically result from the fusion of GPS positioning solutions generated by the consumer-grade GPS

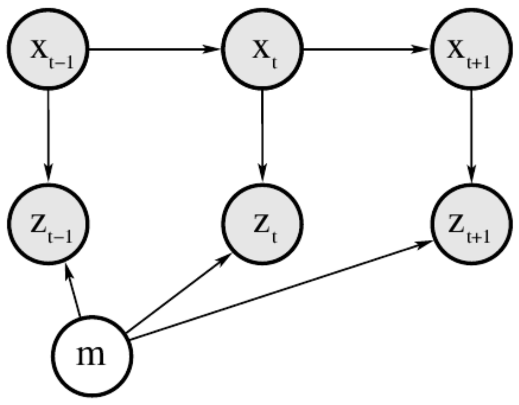


Figure 10.1: Bayesian network representing the problem of mapping with known poses (Image source: Thrun et al. (2005a)).

receiver. One way to avoid this is by never merging certain GPS measurements in the first place. We discussed and for the most part discarded this approach in section 9.8.7 on page 84.

An alternative is suggested by Montemerlo et al. (2008) who had a similar issue caused by the GPS receiver switching between different hypotheses on Junior, the Stanford entry in the Urban Challenge,. They worked around the problem by basing the local obstacle map on what they call "smooth coordinates" computed through integration of velocity estimates:

$$\bar{x} = x_0 + \sum_t \Delta t \cdot \dot{x} \tag{10.2}$$

For the SAMSMobil this is implemented by computing a second, smooth state estimate from the UKF mean by copying everything but the position which is calculated as in (10.2). Effectively, the smooth state estimate has the same orientation as the UKF estimate (which is hardly affected by GPS position jumps) and a smoothed position which is not globally consistent but generates locally physically plausible trajectories regardless of GPS-induced jumps in the regular position estimate.

Practical experience with this approach of building a local obstacle map shows that it works well. Note, however, that the obstacle avoidance part of path planning must happen relative to the robot position.

10.3 Interpolation of Poses

The mapping component on the SAMSMobil uses two input data streams: the smooth pose estimates discussed in the previous section and laser range finder scans. Temporally related data items can be identified by their timestamps (c.f. section 6.5 on page 42). Since the laser range finder clock is not synchronized with the clocks of any of the other sensors, however, the timestamps will rarely match. Thus, we need to apply an interpolation technique to artificially create matching pairs of poses and scans.

Physically it appears more plausible to interpolate between two poses to generate a pose that matches a laser scan lying temporally between them rather than the other way around. For this purpose the mapper uses the Eigen implementation of spherical linear interpolation (Slerp) on quaternions (Shoemake, 1985) a technique popular in computer graphics. Experience with this approach on the SAMSMobil shows that it generates maps which are visibly superior to maps generated by simply choosing the temporally closest pose.

10.4 Pre-Processing of Laser Range Scans

As discussed in section 6.4 on page 42 we essentially get a sequence of ranges r_k from the laser range finder. One of the issues that need to be addressed when dealing with range data is erroneous maximum range readings showing up in the scan data. Within the context of localization and SLAM these are commonly assumed to be caused by low-reflectivity surfaces and techniques exist to explicitely model surface reflectivity (Bennewitz et al., 2009) as part of the measurement model (i.e., a model of the probability of measurements given the current robot pose and map). In our case we can simply drop ranges exceeding a certain threshold since we primarily need to register valid range endpoints in a 2.5D (PTA) or 2D (GEM) map.

Each of these ranges corresponds to an angle φ_k. All software components developed for this thesis use a pre-computed table to map the index k to $\cos\varphi_k$ and $\sin\varphi_k$ which are needed to translate a range into endpoint coordinates in the scan plane. This optimization avoids repeated calls to the costly sine and cosine math library functions.

$$p_k = \begin{pmatrix} r_k \cdot \cos\varphi_k \\ r_k \cdot \sin\varphi_k \\ 0 \end{pmatrix} = \begin{pmatrix} r_k \cdot \text{CosTable}(k) \\ r_k \cdot \text{SinTable}(k) \\ 0 \end{pmatrix} \tag{10.3}$$

10.5 A Generic 2D Map Container

Both of the mapping algorithms implemented for this thesis use grid maps to store a local obstacle map of a certain area around the current robot position and otherwise only differ in the way individual cells are represented which obviously calls for a generic implementation of an efficient map container. The basic idea as illustrated in Figure 10.2 on the next page is that the world around the robot is discretized into square grid cells of a certain side length l so that points in world coordinates p are mapped into the grid as follows:

$$\begin{pmatrix} x \\ y \end{pmatrix} = p - p_{robot} \tag{10.4}$$

$$\begin{pmatrix} c_x \\ c_y \end{pmatrix} = \begin{pmatrix} \lfloor \frac{x}{l} \rfloor \\ \lfloor \frac{y}{l} \rfloor \end{pmatrix}, \tag{10.5}$$

where p_{robot} is the current robot position in world coordinates.

The tricky part is what to do when the robot position changes. After this *move* operation we want the robot to be at the center of the map again while retaining the relative position of previously entered obstacles. A simplistic grid implementation would need to move the map content every time the robot moves as illustrated in Figure 10.3 on the following page. There are two main problems with this approach. Positions in the grid are discrete grid while the robot position can take continuous values and it may be changed by less than the grid granularity at each time step so that we need to keep track of this separately. Secondly and more importantly, moving the entire content is rather expensive at larger grid sizes[1].

[1]For the purposes of this thesis grids typically have a size of 256x256 cells.

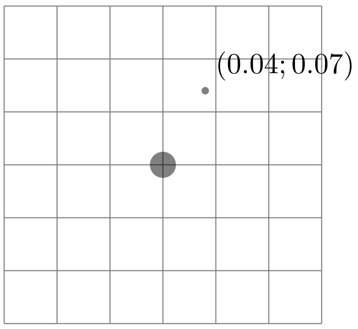


Figure 10.2: The local obstacle map is represented as a grid with the robot (large dot) at its center. Points are mapped into the grid through division by the cell side length and integer rounding. In this example the side length of cells is 0.05m.

Instead, we can use modular arithmetic and a two-dimensional offset which points to the origin of the map to maintain a mapping from world to grid coordinates and reflect moves by adjusting the offset so that the grid content can stay at the same memory locations. One can think of the resulting data structure as a two-dimensional circular buffer. Alternatively, one can visualize the map as a rectangular, transparent blanket wrapped around a toroid such that a change of the two-dimensional offset is equivalent to shifting the blanket around on the toroid surface. The map content would then be attached to the toroid surface and the blanket indicates to which world coordinates the content belongs. This wraparound effect is also illustrated in Figure 10.4 on the next page.

A point (p_x, p_y) is then stored in the array at (c_x, c_y) computed as follows:

$$c_x = \left\lfloor \frac{p_x - origin_x + offset_x}{l} + width \right\rfloor \mod width \tag{10.6}$$

$$c_y = \left\lfloor \frac{p_y - origin_y + offset_y}{l} + height \right\rfloor \mod height \tag{10.7}$$

Apart from the offset, the move operation needs to clear cells that drop off the map. Additional operations implemented by the container include the extraction of a 3x3 neighborhood for use in PTA

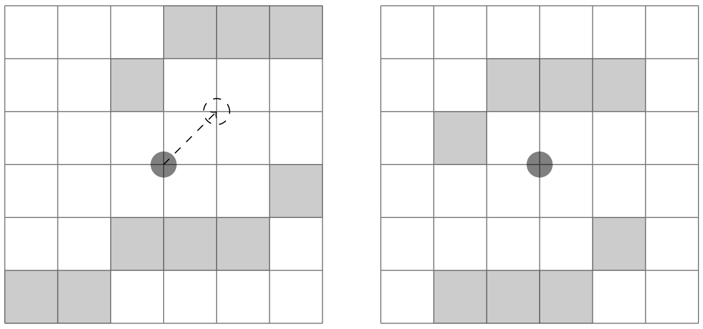


Figure 10.3: In a simplistic grid implementation previously entered obstacles (left) need to be moved to retain their relative position if the robot position changes (right).

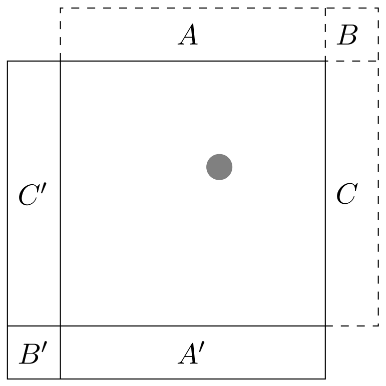


Figure 10.4: An illustration of 2D grid wraparounds. The big dot again marks the map center. The regions A, B and C are mapped to A', B' and C' respectively by the wraparound.

(c.f. section 10.6.1), line visiting to detect obstacles on a path of line segments (section 11.3.3 on page 113) and obstacle extrusion to take the size of the robot into account in the planner (section 11.3.1 on page 111).

The map container is implemented as a C++ template class that takes the side length of the grid and the cell type as template parameters. The array of grid cells is managed by a `boot::array` member which contains a fixed size, one-dimensional array to store the cells in a row-major memory layout[2].

10.6 Probabilistic Terrain Analysis (PTA)

The Probabilistic Terrain Analysis (PTA) mapping algorithm was developed by Thrun et al. for use on the Stanley robot which won the 2005 DARPA Grand Challenge. PTA generates a local obstacle map (Thrun et al., 2006a) which is used for navigation and to train a drivability classifier in the vision component (Dahlkamp et al., 2006) which takes care of long-range road finding and obstacle detection.

In the following we will discuss how, based on the respective papers, PTA was re-implemented, and how an appropriate parameter set has been determined through machine learning. Finally, a set of problems that have been encountered with PTA on the SAMSMobil will be presented which leads us to the development of an alternative algorithm in the next section.

10.6.1 Algorithm

PTA operates on laser scan point clouds and thus first transforms laser scan endpoints into world coordinates. Then the basic idea is to assess the drivability of terrain based on the vertical distance of nearby points. Thrun et al. (2006a) first define a simplified, non-probabilistic *terrain labeling function* which assigns to each point $(X_q, Y_q)^T$ in the 2D plane one of three drivability values:

[2]The array index a corresponding to grid cell (i, j) is simply $a = i \cdot width + j$.

- **Obstacle** if a vertical distance threshold δ is exceeded, i.e, if two points exist such that

$$\left|\begin{pmatrix} X_k^i \\ Y_k^i \end{pmatrix} - \begin{pmatrix} X_q \\ Y_q \end{pmatrix}\right| < \epsilon, \quad (10.8)$$

$$\left|\begin{pmatrix} X_m^j \\ Y_m^j \end{pmatrix} - \begin{pmatrix} X_q \\ Y_q \end{pmatrix}\right| < \epsilon, \quad (10.9)$$

  and

$$\left|Z_k^i - Z_m^j\right| > \delta, \quad (10.10)$$

  where k and m denote the time step when the scan was taken and i and j the index of the endpoint in the scan.

- **Drivable** if the obstacle test did not trigger, but at least one point exists such that

$$\left|\begin{pmatrix} X_k^i \\ Y_k^i \end{pmatrix} - \begin{pmatrix} X_q \\ Y_q \end{pmatrix}\right| < \epsilon. \quad (10.11)$$

- **Unknown** otherwise, i.e., if no point can be found such that

$$\left|\begin{pmatrix} X_k^i \\ Y_k^i \end{pmatrix} - \begin{pmatrix} X_q \\ Y_q \end{pmatrix}\right| < \epsilon. \quad (10.12)$$

Thrun et al. (2006a) further extend this in two ways – the labeling function is replaced with a probabilistic test to avoid phantom obstacles caused by measurement errors and a 2D grid makes the algorithm efficient.

The probabilistic test is based on the observation that the obstacle test compares points acquired at different time steps k and m. As $|m-k|$ increases, so do errors introduced by pose estimation errors which result in spurious vertical distances (Thrun et al., 2006b). The pose estimation error is modeled using a first order Markov model (Thrun et al., 2006a):

$$\begin{pmatrix} x_k^* \\ \Psi_k^* \end{pmatrix} = \begin{pmatrix} x_k \\ \Psi_k \end{pmatrix} + \beta_k + \gamma_k, \quad (10.13)$$

where x_k and Ψ_k denote the true position and orientation and x_k^* and Ψ_k^* their estimated counterparts, β_k is noise accumulating over time and γ_k momentary noise, i.e.,

$$\beta_k \sim \mathcal{N}(\beta_{k-1}, B), \quad (10.14)$$

$$\gamma_k \sim \mathcal{N}(0, C), \quad (10.15)$$

and

$$B = \operatorname{diag}(\sigma_{xyz}^2, \sigma_{xyz}^2, \sigma_{xyz}^2, \sigma_{\varphi\vartheta\psi}^2, \sigma_{\varphi\vartheta\psi}^2, \sigma_{\varphi\vartheta\psi}^2), \quad (10.16)$$

$$C = \operatorname{diag}(\tau_{xyz}^2, \tau_{xyz}^2, \tau_{xyz}^2, \tau_{\varphi\vartheta\psi}^2, \tau_{\varphi\vartheta\psi}^2, \tau_{\varphi\vartheta\psi}^2). \quad (10.17)$$

Based on this they derive a probabilistic test to determine whether the probability of two points being a witness of an obstacle exceeds an acceptance threshold α

$$p\left(\left|Z_k^i - Z_m^j\right| > \delta\right) > \alpha \quad (10.18)$$

as follows:

$$\begin{aligned}\left(\left|Z_k^i - Z_m^j\right| - \delta\right)^2 \;>\; & (\operatorname{quantile}(\mathcal{N}, 1-\alpha))^2 \cdot \\ & \cdot \left(|m-k|(\sigma_{xyz}^2 + r_k^i \sigma_{\varphi\vartheta\psi}^2) + 2 \cdot \tau_{xyz}^2 + r_k^i \tau_{\varphi\vartheta\psi}^2 + r_m^j \tau_{\varphi\vartheta\psi}^2\right)\end{aligned} \quad (10.19)$$

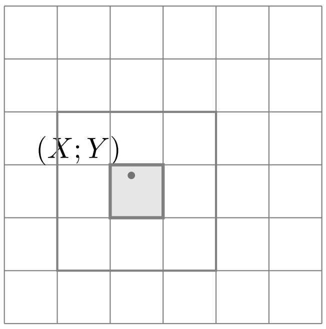


Figure 10.5: Every time a point is entered into the map our PTA update procedure applies the PTA obstacle test to all cells in a 3x3 neighborhood.

Apart from the expected Markov chain parameters, the vertical distance and the time difference, the test also includes the raw ranges which obviously amplify the effect of orientation errors. Note that Thrun et al. (2006a) use an example for the normal distribution p-quantile of the complement of α (1.64 for $\alpha = 0.05$) while this has been replaced in (10.18) and (10.19) to reflect the fact that α is a tunable parameter (see below).

To make the obstacle test efficient Thrun et al. (2006a) use a 2D grid which stores for each cell the minimum and maximum Z value along with the respective time steps. The search in the ϵ-neighborhood is replaced by storing the drivability value in each cell allowing for a drivability lookup in $O(1)$. The drivability value is maintained by an update procedure which is invoked every time a new laser scan endpoint is added to the map and operates as follows:

- If the obstacle test applied to a new Z value and the bounds stored in the current cell or any of its immediate neighbors triggers, the cell is marked as an obstacle and never touched again. Cells have a size of $\frac{\epsilon}{2}$ so that this is equivalent to the search above.
- If this is not the case, but the new value places a tighter lower bound on the Z value the respective data in the cell is replaced because it increases the chance of a future obstacle test triggering.
- The same is done for the upper bound on the Z value.

10.6.2 Implementation

Careful readers will have noticed that in the above description cells do not store the raw ranges although these are needed in the obstacle test (10.19). This is most likely an oversight in the paper by Thrun et al. (2006a). This part of the PTA cell representation developed for this thesis is implemented as follows: The lower and upper bounds are stored as two collective measurement – each consisting of the Z value, the range and the time step.

The PTA cell C++ class is further able to perform the update procedure including the probabilistic obstacle test. The update procedure employs a 3x3 cell neighborhood as illustrated in Figure 10.5. This class is used to instantiate the generic grid container template class and extend it with some PTA specific glue-code to form our PTA map. An additional wrapper class maintains the parameter set and takes care of feeding sensor range data into the map. This is then used as one of two mapping algorithms by the mapper.

10.6.3 Parameter Learning

We still have to discuss how the various parameters in the obstacle test (10.19) are determined. These are the vertical distance threshold δ, the acceptance threshold α and the noise parameters σ^2_{xyz}, $\sigma^2_{\varphi\vartheta\psi}$, τ^2_{xyz} and $\tau^2_{\varphi\vartheta\psi}$. Thrun et al. (2006a) suggest to use log data acquired from human driving and since manual tuning turned out to be nearly impossible this strategy was adopted for the SAMSMobil as well.

As illustrated in Figure 10.6 on the facing page[3] the basic idea is that terrain the robot has successfully traversed under human control was obviously drivable. Similarly, we need samples of obstacles. For this purpose we assume that a stretch to the left and to the right (with a certain margin of unknown cells) corresponds to the path boundaries (grass, bushes, etc.) and is thus marked as obstacle cells.

The author implemented a dedicated software module called `map_trainer` which is derived from the regular mapper but rather than processing data in real time stores a set of range scans along with matching poses as a training data set. It then uses the coordinate ascent method described by Thrun et al. (2006a) to find an optimal PTA parameter vector.

10.6.4 Problems with PTA

Tests with our implementation of PTA quickly revealed the following major issues:

- Small pose estimation errors translate to possibly significant classification errors in the map since the primary source of information for PTA is the vertical distance of range scan endpoints falling into a cell. Estimation errors in both poses used to calculate the lower and upper bound respectively add up to spurious height differences. The learned Markov error chain parameters can hide the resulting false positive obstacles but this is an inevitably lossy process that degrades the overall map quality.
- Once a map cell has been marked as an obstacle, PTA never reconsiders that classification decision even though a cell which appeared to be an obstacle from afar may turn out to be drivable later (since the effect of pose estimation errors is amplified by the range of laser scans).
- Negative obstacles cannot be detected at all in most cases. Here, negative obstacles refers to obstacles such as curbs seen from a sidewalk as illustrated in Figure 10.7 on page 100. The reason for this is that the upper end of the negative obstacle is registered in a grid cell that is too far away from that corresponding to the lower end. This shortcoming was the main incentive behind the development of a new mapping algorithm (see next section) after initial tests in our target environment since curbs present a major hazard to the SAMSMobil. Closer inspection of Figure 9 in the paper by Thrun et al. (2006a) reveals that Stanley, too, did not "see" the negative obstacle consisting of the cliff by the "Beer Bottle Pass" passage of the 2005 Grand Challenge course.
- The grid cell size is a parameter with subtle effects. Obvious criteria in selecting an appropriate value are the granularity one would like to achieve (smaller grid cells are better), the computational cost (larger grid cells mean less cost while covering the same area), and sensor precision/resolution (there is no point in having grid cells smaller than the sensor precision). However, the size of a grid cell also has a significant effect on the classification process. Consider a ramp with a constant gradient: If the cell size is too large it will trigger the height difference

[3]Note that Figure 10.6 on the next page shows one of the data sets where the approach worked exceptionally well because the grass had not been cut for a long time when the log was acquired. In later experiments this was never the case again.

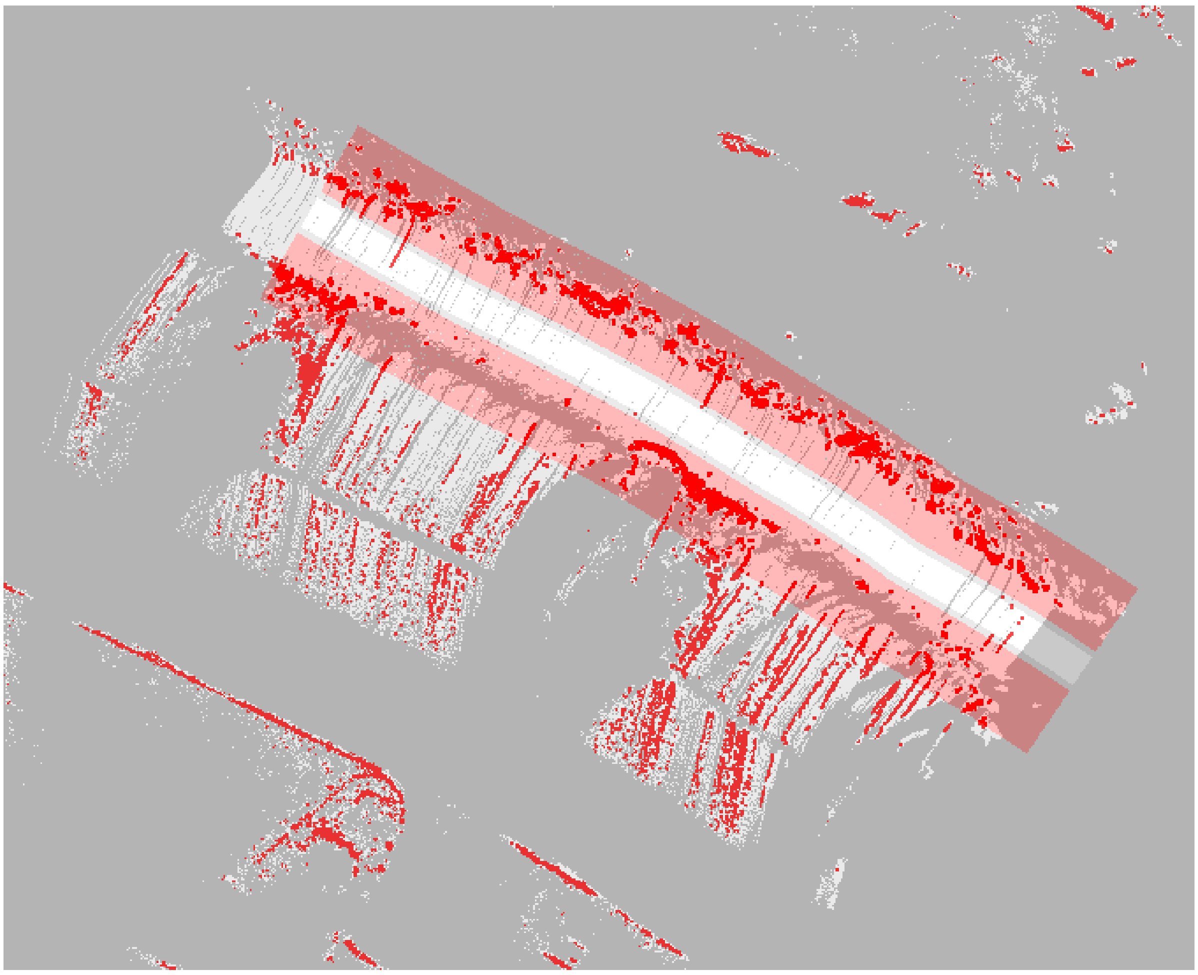


Figure 10.6: Automatic labeling of the grid cell drivability based on log data. The robot drove from the lower-right corner to the upper-left corner in the map. The white stretch is labeled as drivable because it was traversed by the robot. One stretch to the left and one to the right are assumed to correspond with the path and thus marked as non-drivable. All other cells are ignored in the machine learning process.

threshold as illustrated in Figure 10.8 on the next page. Conversely, obstacles might be classified as drivable if the cell size is too small because the cell size does not allow enough height difference to "accumulate". Note that PTA does not only consider height differences in individual cells – it uses a 3x3 cell neighborhood when comparing height values.

We will analyze what effects these issues have on a test data set in 12.2.3 on page 132.

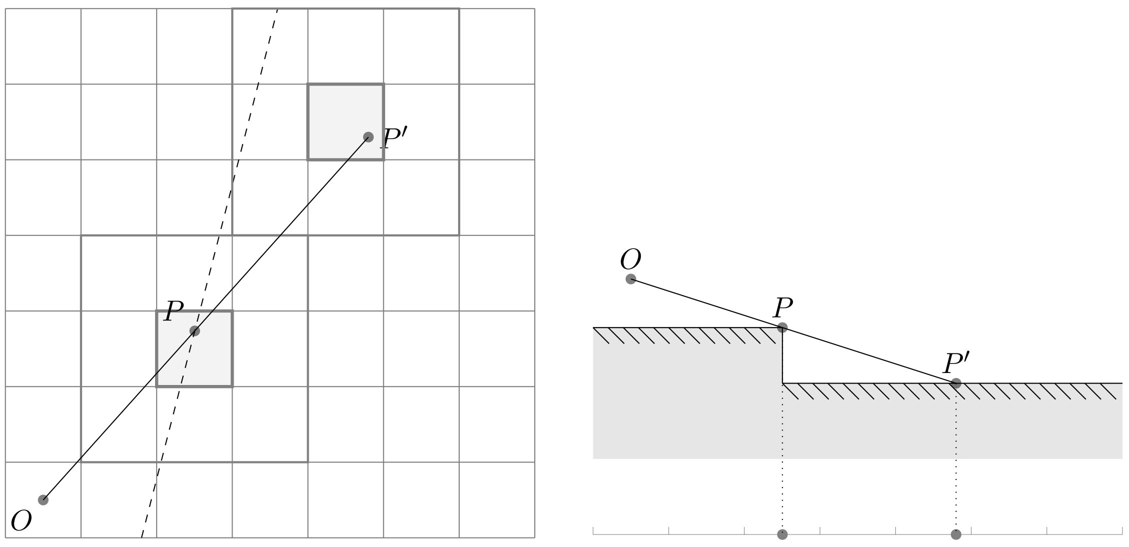


Figure 10.7: Left: A lidar beam originating in O touches the curb (dashed line) in P and hits the road below in P'. P and P' are entered in map cells (shaded) more than the 3x3 cell neighborhood away from each other. Right: A cross section of the situation with the sidewalk, the curb, and the road below (from left to right).

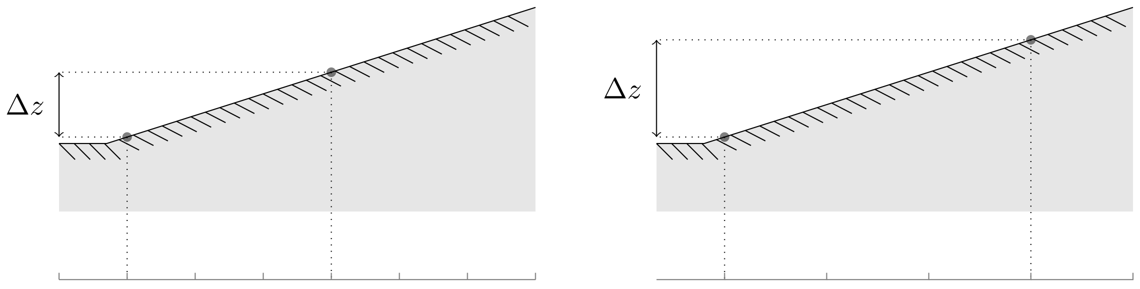


Figure 10.8: The classification of a ramp changes depending on the chosen grid cell size in PTA. The same gradient that is classified as drivable with a small cell size (left) is classified as an obstacle if the cell size is larger (right).

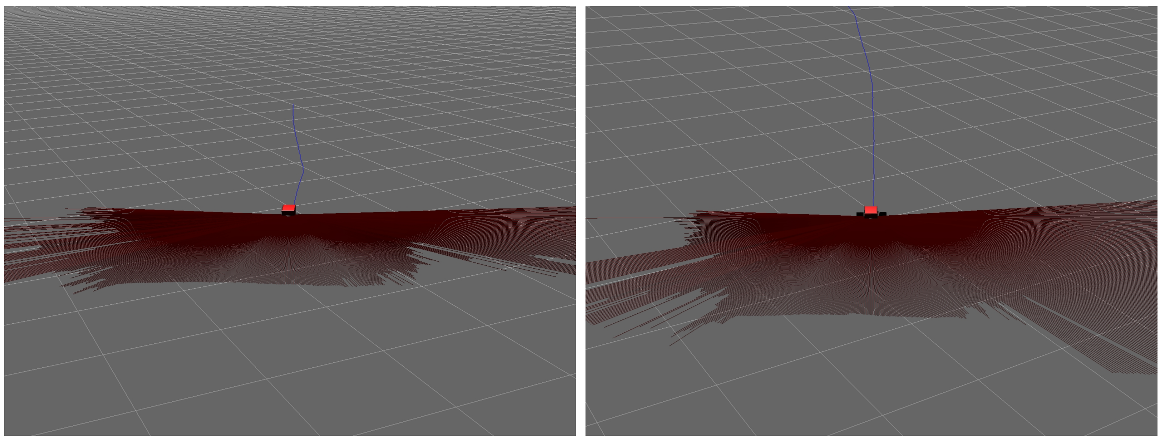


Figure 10.9: The visualization results that motivated the development of GEM. In front of the robot the range scans stem from hitting the ground on the drivable path. A human observer can easily distinguish range scans originating from positive or negative obstacles from these.

10.7 Ground Expectation Mapping (GEM)

This section discusses the development of a new mapping algorithm that would address several of PTA's shortcomings. In particular, the new algorithm should not just detect positive but also negative obstacles which as discussed by Kelly & Stentz (1997) is a requirement for safe outdoor mobility. This is instrumental since it allows the SAMSMobil to safely operate on sidewalks. We will start with a presentation of the general idea and successively refine the algorithm.

10.7.1 Towards a New Mapping Algorithm for the SAMSMobil

The idea for the mapping approach we are about to discuss came up while watching log replays in the visualization environment which represents laser range scans as a set of red line segments. It turns out that most of the time a human observer is perfectly able to classify range scan endpoints as evidence of drivable vs. non-drivable terrain from a single scan (Figure 10.9). The author believes this is because a human observer quickly establishes an expectation of how the ground plane should show up in the range scan. If the actual range scan does not match this expectation the respective endpoints are considered as obstacles. This works for both positive and negative obstacles and gives the new algorithm its name Ground Expectation Mapping (GEM).

We can mimic this human intuition by computing the intersection of the scan plane with the ground plane to generate an expected (simulated) range scan. The result is illustrated in Figure 10.10 on the following page for an indoor environment and in Figure 10.11 on the next page for an outdoor environment. In the indoor case the actual range scan nearly matches the expected line whenever the respective endpoints correspond to a non-obstacle area of the environment. In the outdoor plot the match is slightly less perfect but it is still easy to differentiate obstacles from the drivable path right in front of the robot (around $x = 0m$)

The endpoint classification can now be implemented by comparing the difference between the actual and the expected (simulated) range against two thresholds. A first threshold α provides a lower bound

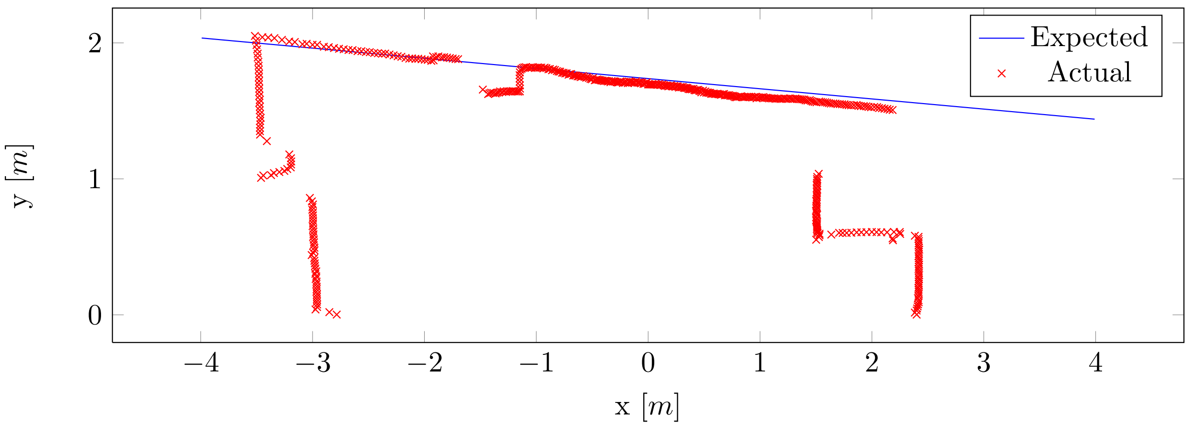


Figure 10.10: Expected ground plane intersection vs. actual laser range scan endpoints in an indoor setting. The plot shows the scan plane.

on range differences indicating obstacles (classification HIT). A second threshold $\gamma \leq \alpha$ establishes an upper bound on range differences for endpoints that are to be classified as evidence of drivable terrain (classification MISS). Range differences between these two bounds are classified as undecided (classification UNDECIDED).

This gives rise to Algorithm 1 CLASSIFYENDPOINTS which uses the as yet unspecified function MARKAS to register the classification result in the obstacle map.

The primary job of Algorithm 2 MARKAS is to choose whether the actual or the expected range should be used when entering the endpoint into the map. The idea is that, if $r_a < r_e$, the corresponding laser

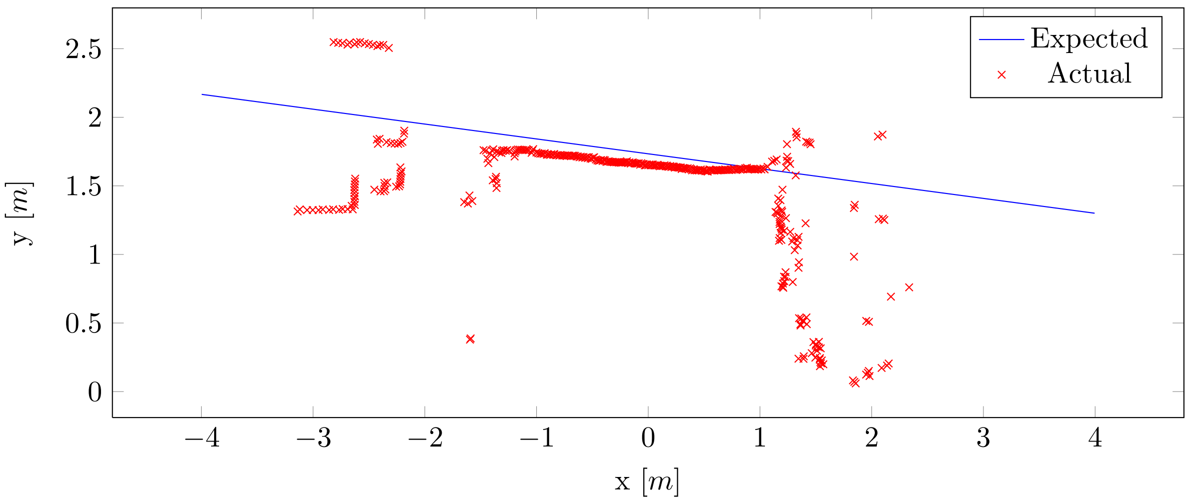


Figure 10.11: Expected ground plane intersection vs. actual laser range scan endpoints in an outdoor setting. The plot shows the scan plane.

Algorithm 1 ClassifyEndpoints(S_a, S_e)

Require: An actual laser range scan S_a and an expected one S_e each of size n.

```
1: for $k = 1$ to n do
2:     $(\varphi_{a,k}, r_{a,k}) \leftarrow S_a[k]$
3:     $(\varphi_{e,k}, r_{e,k}) \leftarrow S_e[k]$
4:     $d \leftarrow |r_{a,k} - r_{e,k}|$
5:     if $d > \alpha$ then
6:         MarkAs(Hit, $\varphi_{a,k}, r_{a,k}, r_{e,k}$)
7:     else if $d < \gamma$ then
8:         MarkAs(Miss, $\varphi_{a,k}, r_{a,k}, r_{e,k}$)
9:     else
10:        MarkAs(Undecided, $\varphi_{a,k}, r_{a,k}, r_{e,k}$)
11:    end if
12: end for
```

beam hit an obstacle before it should have hit any, i.e., the ground plane, as illustrated in Figure 10.12 (left). If $r_a > r_e$, the laser beam should have hit the ground plane at range r_e but only did so later indicating a hole in the ground or some other negative obstacle (Figure 10.12, right). In either case the safe range is the minimum of the two $r = \min(r_a, r_e)$. MarkAs then computes the point corresponding to (φ, r) in lidar coordinates, translates it to world coordinates using Lidar2World and passes the result together with the classification to RegisterInMap which takes care of storing it in the obstacle map.

Algorithm 2 MarkAs(c, φ, r_a, r_e)

Require: A laser range scan endpoint classifcation result $c \in \{\text{Hit}, \text{Miss}, \text{Undecided}\}$ with corresponding angle φ, actual range r_a and expected range r_e.

```
1: $r \leftarrow \min(r_a, r_e)$
2: $p \leftarrow \begin{pmatrix} r\cdot\cos\varphi \\ r\cdot\sin\varphi \\ 0 \\ 1 \end{pmatrix}$
3: RegisterInMap($c$, Lidar2World($p$))
```

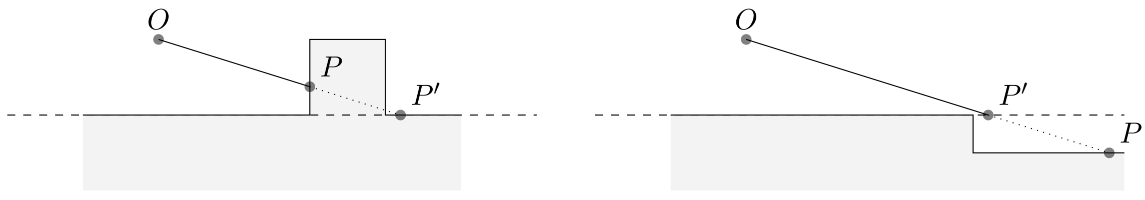


Figure 10.12: Positive obstacles (left) and negative obstacles (right) in GEM. The actual range measured is $r_a = |\overline{OP}|$ while a range of $r_e = |\overline{OP'}|$ was expected assuming there is nothing but the ground plane (dashed line). Obviously, anything beyond $min(r_a, r_e)$ must be considered as no longer drivable.

10.7.2 Simulating Laser Range Scans

The GEM classification algorithm takes as input an actual range scan S_a from the laser range finder and an expected range scan S_e and we still have to look at how the latter is generated. Conceptually, we are interested in the intersection of the scan plan with the ground plane. What we actually need is the sequence of ranges that would be reported by a virtual laser range finder in the same pose L2W as the real one but with nothing but the ground plane as its environment (defined by its normal n_g). So, for each laser beam starting at angle φ_1 with consecutive beams $\Delta\varphi$ apart Algorithm 3 SIMULATERANGESCAN computes the intersection of a line pointing in the same direction with the ground plane and adds the distance to the intersection (clipped to the maximum range r_{max}) to S_e.

Algorithm 3 SIMULATERANGESCAN(L2W, n_g)

Require: A transformation L2W representing the lidar pose and the normal vector n_g of the ground plane.
Ensure: A simulated range scan S_e of size n.

```
1:  $S_e \leftarrow []$
2:  $\varphi \leftarrow \varphi_1$
3:  $o \leftarrow \mathrm{L2W}(\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix})$
4:  for $k = 1$ to n do
5:      $p \leftarrow \mathrm{L2W}(\begin{pmatrix} \cos\varphi \\ \sin\varphi \\ 0 \\ 1 \end{pmatrix})$
6:      $\lambda \leftarrow \mathrm{INTERSECT}(\mathrm{PLANE}(n_g), \mathrm{LINE}(o, p))$
7:      if $\lambda < 0$ then // behind laser range finder
8:          $r \leftarrow r_{max}$
9:      else
10:         $r \leftarrow \min(r_{max}, o + \lambda(p - o))$
11:     end if
12:     $S_e \leftarrow S_e + [r]$
13:     $\varphi \leftarrow \varphi + \Delta\varphi$
14: end for
15: return $S_e$
```

As for L2W and n_g, the simplest choice for these is L2W := LIDAR2ROBOT and $n_g := (0,0,-1,1)^T$. This corresponds to the assumption that, since the robot follows the gradient of the surface it drives on, the plane defined by $(0,0,-1,1)^T$ in robot coordinates approximates the actual ground plane near the current robot position . By that logic we get from lidar to (approximated) world coordinates via LIDAR2ROBOT.

This works surprisingly well but obviously disregards information we get from the pose estimator on the orientation of the robot. The surface is not perfectly smooth which (amplified by the wheel geometry) leads to high frequency orientation changes in addition to the low frequency changes induced by the general gradient of the surface. This gives rise to the idea of low-pass filtering the robot's orientation and using $n_g := LP(\mathrm{LIDAR2WORLD})(0,0,-1,1)^T$ and L2W := LIDAR2WORLD where LIDAR2WORLD is the transformation from lidar to world coordinates as estimated by the pose estimator.

10.7.3 Low-Pass Filtering the Orientation Estimate

The easiest way to low-pass filter the orientation estimate is to compute the average of the previous k orientation estimates. As discussed in section 9.6 on page 71, special care is needed when dealing with orientations in 3D. In particular, it is (in general) impossible to represent our set of k orientations as vectors and compute their average using vector arithmetic.

Computing the average of a set of 3D orientations is apparently a common technique used within the fields of materials science and crystallography (Krieger Lassen et al., 1994; Humbert et al., 1996; Morawiec, 1998). Markley et al. (2007) discuss the topic within the context of spacecraft attitude estimation in a way that feels slightly more accessible.

Markley et al. (2007) use quaternions to represent orientations and basically start by defining the mean $\bar{q}$ of a set of n quaternions q_i as

$$\bar{q} = \underset{q \in \mathbb{S}^3}{\operatorname{argmin}} \sum_{i=1}^{n} \|A(q) - A(q_i)\|_F^2 \,, \tag{10.20}$$

where $A(q)$ denotes the rotation ("attitude") matrix corresponding to q, $\|\ldots\|_F^2$ the squared Frobenius norm, and $\mathbb{S}^3$ the unit 3-sphere. They further show that this is equivalent to solving the maximization problem

$$\bar{q} = \underset{q \in \mathbb{S}^3}{\operatorname{argmax}}\, q^T M q, \tag{10.21}$$

where M is the 4x4 matrix

$$M = \sum_{i-1}^{n} w_i q_i q_i^T \,, \tag{10.22}$$

w_i is a set of weights, and q (resp. q_i) is treated as a vector. Finally, Markley et al. state that the $\bar{q}$ solving (10.21) is the eigenvector corresponding to the maximum eigenvalue of M.

For use in GEM this approach has been implemented with all weights set to 1 using the QR decomposition based eigenvalue solver provided by Eigen. In the GEM based mapper it works well in practice and has the advantage of not needing the iterative process that was used to compute the mean of a set of manifold elements (sigma points) in section 9.6 on page 71 which reduces the computational cost of the mapper component. However, upon writing this (and re-reading the paper by Markley et al. (2007)) the author noticed that the distance metric expressed in (10.20) would need to be shown to be equivalent to the Riemannian distance between orientations (elements of $SO(3)$) and that $\bar{q}$ also minimizes $\sum_{i=1}^{n} \|q_i \boxminus q\|$. Although, intuitively, this seems doubtfuk, the author has not investigated the issue further since a mathematically sound method is already available (Equations (9.53), (9.54) and (9.55) on page 73) and the computational overhead it introduces by its iteration steps is acceptable.

10.7.4 Handling More Undecided Cases

Recall that ClassifyEndpoints (Algorithm 1 on page 103) classified laser range scan endpoints which have a range difference d to the expected range satisfying $\gamma \leq d \leq \alpha$, i.e., a range that is neither clearly close to the expected range nor very far from it, as Undecided.

We can improve on this by taking into account that many obstacles will cause "spikes" in the range scan, i.e., the gradient of the actual range scan will differ significantly from that of the expected range scan. This gives rise to the introduction of another threshold δ on the maximum allowed gradient difference for non-obstacles. Both gradients are computed through a linear regression process applied to the k endpoints before and after the current endpoint (Frese, 2009). While this is fairly straight forward there is one catch: Corners of clear obstacles would “drop shadows” as they always cause a large gradient difference. Thus we only apply this refinement step if none of the k ranges before and after the current one have been classified as obstacles yet.

10.7.5 A Probabilistic Map to Manage Classifications

So far we have described a method of classifying laser range scan endpoints. The question of how endpoint classifications from multiple scans are to be handled is yet to be answered. One could simply wait for the first HIT classification, mark the corresponding grid cell as an obstacle, and never look at it again. This might, however, make the algorithm even more susceptible to noisy input data affecting the map than PTA. Instead, one would like to “smoothen” the input data and allow the algorithm to reconsider the drivability of map decision later on.

In indoor robotics systems the probably most popular approach in this regard is the occupancy grid mapping approach originally proposed by Elfes (1987) which was popularized by Moravec (1988) and effectively standardized by Thrun et al. (2005a). The key idea is that the map m is modelled as a discrete grid (two-dimensional array) of map cells m_i each holding information on the probability that it is occupied by an obstacle. Thus the mapping problem is approximated (based on the assumption that individual map cells are independent) as a factorization of binary (occupied vs. free) estimation problems:

$$p(m|z_{1:t}, z_{0:t}) \approx \prod_i p(m_i|z_{1:t}, z_{0:t}) \tag{10.23}$$

Note that it is this independence assumption that makes grid mapping algorithms computationally tractable. Based on this, the binary Bayes filter definition under a static world assumption, and transformation into log-space (to avoid numerically problematic multiplications of (small) probabilities) one can derive (Thrun et al., 2005a, Sec. 9.2) the update rule of the occupancy grid mapping algorithm:

$$l_t(m_i) = l_{t-1}(m_i) + \bar{l}_t(m_i) - l_0 \tag{10.24}$$

which is applied for every grid cell m_i where l_0 is the prior (“baseline probability”) as a log-odds ratio

$$l_0 = \log \frac{p(m_i)}{\neg p(m_i)} = \log \frac{p(m_i)}{1 - p(m_i)}, \tag{10.25}$$

$\bar{l}_t(m_i)$ is the inverse sensor model

$$\bar{l}_t(m_i) = \log \frac{p(m_i|z_t)}{1 - p(m_i|z_t)} \tag{10.26}$$

and $p(m_i|z_{1:t}, z_{0:t})$ can be recovered from the log-odds representation via:

$$p(m_i|z_{1:t}, z_{0:t}) = 1 - \frac{1}{1 + \exp(l_t(m_i))}. \tag{10.27}$$

The map quality of occupancy grid maps is highly dependent on good inverse sensor models. Reference implementations exist (Thrun et al., 2005a, Sec. 9.2) for laser range finders mounted in such a way that the scan plane is parallel to the ground. Alternatively, appropriate models may be learned from sensor data (Thrun et al., 2005a, Sec. 9.3) which is particularly useful for “tricky” sensors such as sonar distance sensors.

A similar (although not fully equivalent) approach, sometimes referred to as a reflection map (Bennewitz et al., 2009), is based on simple counting. The idea is that each grid cell holds the probability that an obstacle in that cell reflects the lidar or sonar beam (Thrun et al., 2005b, Slide 21):

$$p(m_i) = \begin{cases} \frac{\text{hits}(m_i)}{\text{hits}(m_i)+\text{misses}(m_i)}, & \text{if } \text{hits}(m_i) + \text{misses}(m_i) \neq 0 \\ 0.5 \quad , & \text{otherwise} \end{cases} \tag{10.28}$$

This approach does not require an inverted sensor model. All that is needed is to count the number of times a beam hit an object (and was reflected) in cell m_i (hits(m_i)) and the number of times a beam crossed it without being reflected (misses(m_i)). It can also be shown (Thrun et al., 2005b, Slides 23–26) that this reflection counting yields the maximum likelihood reflection map. Note, however, that this approach in general does not yield a map identical to that generated by the occupancy grid map algorithm as the inverted sensor model allows occupancy grid maps to model cases where a cell is occupied but due to low reflectivity of the respective surface the beam is not reflected (Thrun et al., 2005b, Slide 27).

For our purposes the occupancy grid map approach is problematic as we intend to feed classification results rather than the original sensor readings into the algorithm and the development of an adequate inverted sensor model for this "virtual sensor" does not seem tractable. Instead, we employ the reflection counting method but redefine hits(m_i) and misses(m_i) as the number of times the classification algorithm classified an endpoint with (x, y) coordinates falling into cell m_i as a HIT (obstacle) or a MISS (drivable) respectively. $p(m_i)$ then denotes the probability of a cell being an obstacle and we consider a cell as drivable if $p(m_i) < 0.5$. Note how the redefinition of hits(m_i) and misses(m_i) changes the way a single scan is entered into the map: Cells which are crossed by a beam but do not contain an endpoint classified as HIT or MISS are left unmodified as illustrated in Figure 10.13 on the following page.

A further modification allows us to take into account that due to (minor) pose estimate inaccuracies endpoint classifications near the vehicle are more precise than those farther away from it. We simply count nearby classifications (range less than 3m) twice.

As we will see in 12.2.3 on page 134, this combination of ground expectation based classification and probabilistic grid mapping works well in practice and overcomes several limitations (c.f. section 10.6.4 on page 98) of PTA.

10.7.6 Implementation

The author implemented a C++ class representing a reflection counting map cell as described in section 10.7.5 on the preceding page and again plugged this into the generic map cointainer template. A GEM class manages the parameter set, interfaces with the orientation low pass filter and laser scan simulator, and implements the classification algorithm (CLASSIFYENDPOINTS, MARKAS) including the extension to take gradient differences into account from section 10.7.4 on page 105. This class is the second mapping algorithm used by the mapper module.

10.8 Propagation of Obstacle Information

In the literature grid maps are often shared between different processing modules through shared memory (Thrun et al., 2006b). For the SAMSMobil the author decided against this in order to avoid overlapping address spaces and retain full network transparency. Instead, the map container (section 10.5 on page 93) has replication support. Each module that requires access to the map (normally

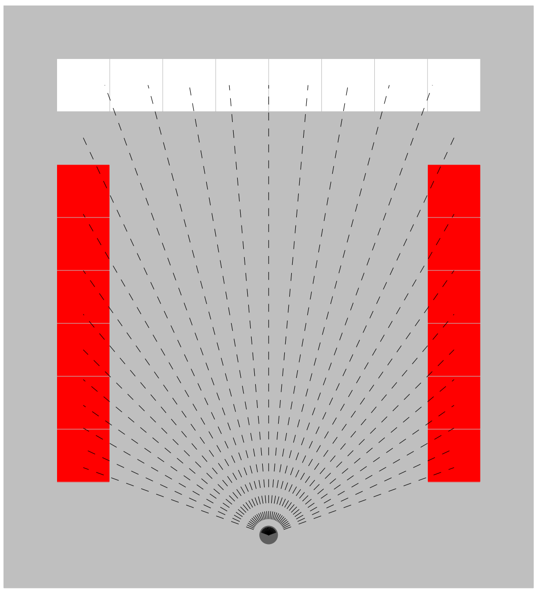


Figure 10.13: Effect of a single scan (dashed lines) acquired in a corridor scenario on the GEM map. Gray cells are left unchanged. Endpoints classified as HIT add to hits(m_i) as indicated by the red cells which correspond to the corridor walls to the left and right. Endpoints classified as MISS increase misses(m_i) for each cell filled in white which is the case for the endpoints hitting the ground at the top. The two gaps in between the red and white bars are caused by UNDECIDED classifications.

just the planner) maintains a local replicated map. The mapper sends map updates through Isil which when applied consecutively yield the same obstacle map as that generated by the mapper internally. Each map update contains the new robot position (center of the map) along with drivability and height information for each grid cell that was affected by the respective laser range scan. On the replicating end the new robot position is first passed to the move operation and then each grid cell (identified by its integer index in the grid array) is updated.

Chapter 11

Path Planning and Control

This chapter discusses the path planning and control components which, based on the information from the localization and mapping components, close the control loop and allow the robot to drive autonomously.

11.1 Overview

In our application scenario the robot is supposed to follow a predefined global goal path. Local planning is required to avoid obstacles and keep the robot on the path despite possible inconsistencies between the goal path and the actual environment and GPS induced state estimation offset. The job of the path controller is to execute the path chosen by the planner.

The planning strategy implemented below is basically the same as the one by Thrun et al. (2006b) who propose to have the planner operate in a lateral offset space relative to the current path. The planner is invoked every time the local obstacle map changes. It then finds the path segment closest to the current robot position, chooses and applies an offset function, and sends an updated path to the path controller. Repeated application of different offset functions can generate sophisticated maneuvers as illustrated in Figure 11.1.

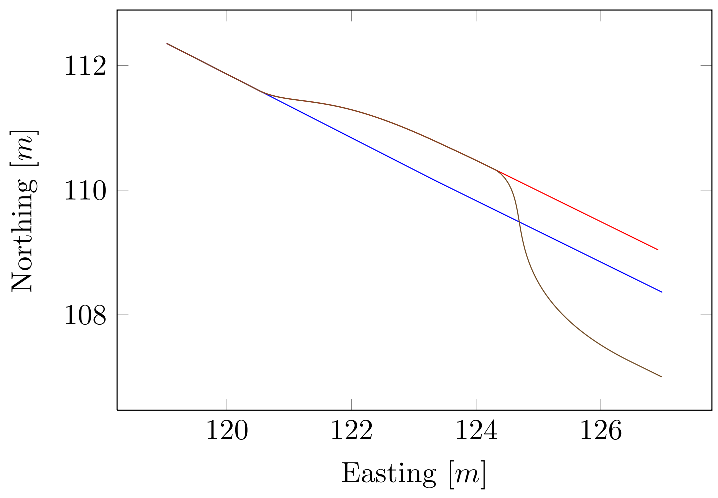


Figure 11.1: Consecutive application of two path offset functions with different parameters and in opposing directions.

11.2 Path Representation

The input path (a KML file; see section 5.4 on page 24) consists of a sequence of waypoints in WGS84 coordinates. We convert these into a local Cartesian coordinate system (c.f. section 7.1 on page 46) and normalize the path by introducing additional waypoints such that each path segment has a length of 0.1 m. For the purposes of this thesis the input path is assumed to be reasonably smooth and in contrast to Thrun et al. (2006b) no preparatory path smoothing is performed.

The data structure representing a path must support two operations:

- Project a point onto the closest segments on the path.
- Apply a path offset function.

An array of path segments allows for fast access to segments by index. The search for the closest segment in the projection operation is then implemented as a linear search within a 10 m neighborhood around the previously closest segment starting 5 m before it. Path offset functions are a little more involved and will be discussed next.

11.2.1 Path Offset Functions

Thrun et al. (2006b) differentiate gradual offset changes ("swerves") and more aggressive ones ("nudges") but do not specify how exactly these are generated. In the following we will use this definition:

- As proposed by Thrun et al. (2006b), a path offset function is defined by two parameters, one that influences the rate of change and another one that determines the total amount of lateral offset.
- The return value is a sequence of 40 rotation angles (reaching out four meters on the path ahead of the vehicle).

To get a smooth path change the rotation angles should start at 0 smoothly increase up to a certain maximum and then go back down to 0. We can use a sine function over the domain $[0;\pi]$ for this purpose. Additionally, we want to influence how quickly the maximum is reached to model the rate parameter. This is handled by an exponential function in the argument of the sine function. Finally, we want the planner to choose from a variety of offset functions with a small total offset which we model through another exponential function. The maximum rotation angle is $\pm\frac{\pi}{2}$. Thus, the sequence of rotation angles is generated as follows:

$$a(x,i,r) = \pm\frac{\pi}{2} \cdot \frac{\exp(i * 0.5) - 1}{\exp(5) - 1} \cdot \sin(\pi \cdot \frac{1 - \exp(-r \cdot x)}{1 - \exp(-r \cdot \pi)}), \tag{11.1}$$

where $x \in \{0, \dots, 39\}$ is the index in the return sequence, $i \in \{0, \dots, 9\}$ influences the total offset, and $r \in \{0.8, 1.2, 1.6\}$ the rate of change. The $\pm$ allows for offset changes to the left and to the right hand side of the path respectively.

As the discrete choice of values for i and r indicates the actual planner implementation uses a precomputed table of offset function results represented as $\mathbb{R}^{2x2}$ rotation matrices to avoid unnecessary math library calls at runtime.

11.2.2 Applying a Path Offset Function

The application of a path offset function starts at a certain index k in the path segment array and works as follows[1]:

Algorithm 4 ApplyOffsetFunction(n, i, r)

```
1: Initialize offset $o = (0, 0)^T$
2: for $n = k$ to k+39 do
3:     translate segment $n$ by $o$.
4:     rotate segment $n$ about its start point by $a(n - k, i, r)$.
5:     add difference of end point of segment $n$ before and after rotation to $o$.
6: end for
```

The question is what to do with o after the end of the loop. Translating all later path segments would be too expensive. Instead, the path representation stores o and remembers the index after which it needs to be applied. The path segments then take care of the translation on demand. An example of the application of path offset functions can be found in Figure 11.1 on page 109.

11.3 Local Path Planning

The planner first replicates and extrudes the drivability classification in the local obstacle map. Then it searches for a drivable path that minimizes a cost function. We will discuss each of these steps in more detail now.

11.3.1 Obstacle Extrusion

The planner needs to check a path for drivability. One way to implement this is by looking for obstacles within a polygon defined by the geometry of the vehicle, a motion (and braking) model and the current velocity (c.f. (Frese et al., 2008a)). A simpler approach is based on artificially extruding obstacles in the local map such that a certain radius around actual obstacles is also treated as non-drivable. The extrusion radius is defined by the radius of a bounding circle around the robot plus the distance it would take the robot to come to a halt given maximum deceleration. This reduces the obstacle test to a test of the drivability of the line segments the path consists of (see below). The effect of the extrusion operation is illustrated in Figure 11.2 on the following page.

The implementation of the extrusion operation is based on a precomputed, approximately circular extrusion mask which has the same effect as a large brush in a graphics editor. Whenever a cell is classified as an obstacle, each cell covered by a mask around it is marked as extruded which the planner treats as non-drivable. Internally, cells keep an extrusion count which works analogously to a reference count. This is essential when it comes to handling changes in the classification of a cell (as can happen in GEM, but not PTA): If a cell changes its classification from obstacle to drivable the extrusion count of all cells within the mask is decremented (if previously greater 0) such that extruded cells, too, will no longer be treated as obstacles.

[1]Note that this description is slightly simplified for clarity. The actual implementation additionally needs to handle corner cases arising from path offset functions being applied to overlapping parts of the path or when there are gaps in between.

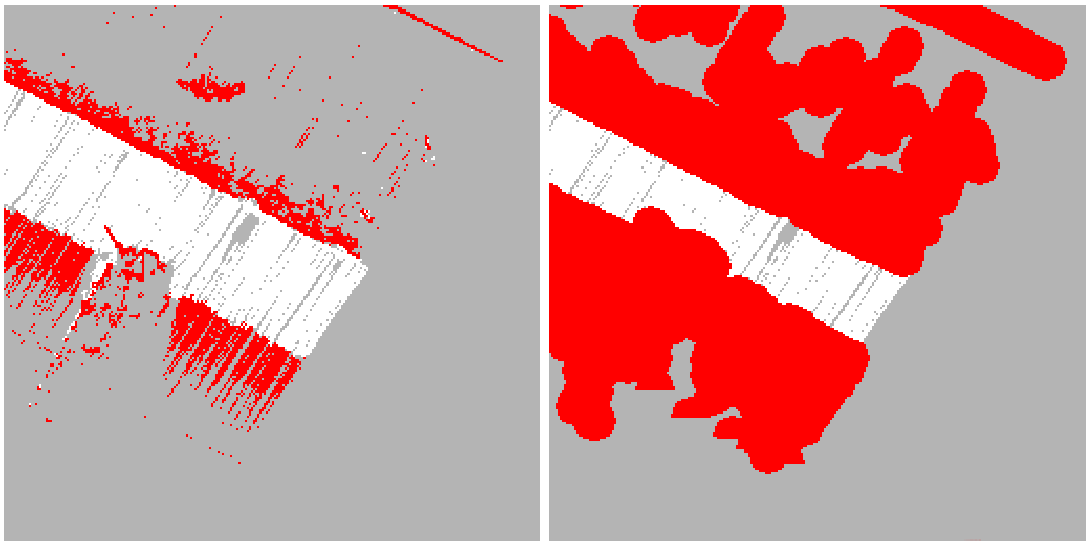


Figure 11.2: The local obstacle map before (left) and after obstacle extrusion (right).

11.3.2 Path Search

At the core of the planner is a path search algorithm that finds an optimum path satisfying the following goals:

- Avoid obstacles.
- Stay near the center of the sensed path.
- Avoid turning if possible.

The first thing the planner does is to project the current robot position onto the path to determine the index of the path segment the path search starts at. The application of path offset functions at this index creates candidate paths which form a tree. The root node is the start point of the first segment. It has one child node for each possible path offset function (i.e., one for each parameter pair in $\{(i, r)|i \in \{0, \ldots, 9\} \wedge r \in \{0.8, 1.2, 1.6\}\}$). Each of these is the start of a branch corresponding to the following 39 path segments after application of the candidate path offset functions. An example of such a tree for a singe rate ($r = 0.8$) is depicted in Figure 11.3 on the next page.

The SAMSMobil mapper uses a slightly modified version of Dijkstra's shortest path algorithm (Dijkstra, 1959) to find the minimum cost path offset function. At the expansion of each node the path segment corresponding to each arc is checked for drivability in the local obstacle map (see below) to filter out non-drivable candidates early. To encode the remaining goals defined above, the cost of an arc is determined as the sum of

- the result of a "valley" function which penalizes candidate paths that get too close to the sensed path boundary and
- the absolute rotation angle.

The valley function consists of the combination of two sigmoid functions, one for each path boundary as illustrated in Figure 11.4 on the facing page. The distance of the sensed path boundaries is determined by scanning a line perpendicular to the path for obstacles and feeding the result as measurements into a separate one-dimensional UKF instance for low pass filtering.

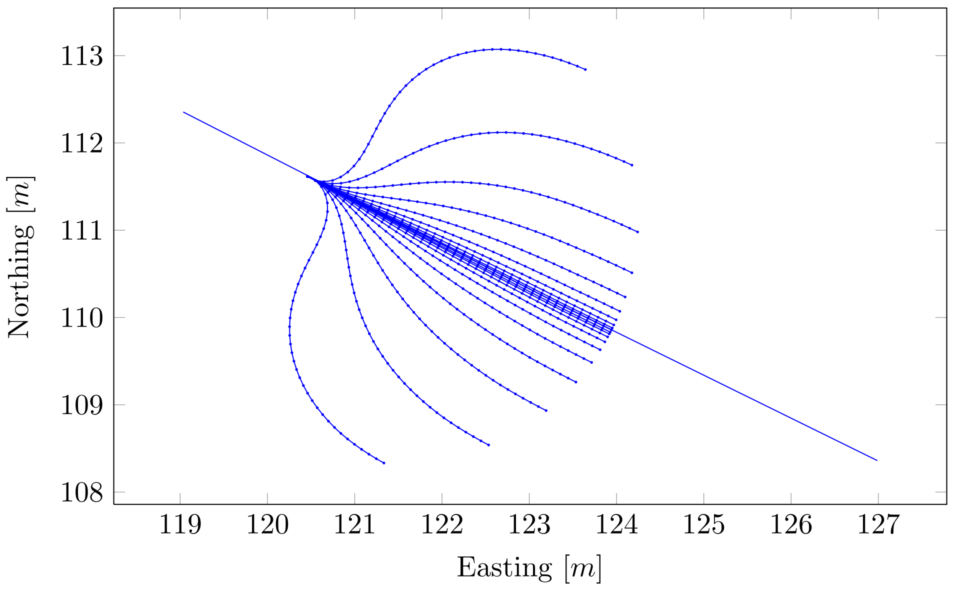


Figure 11.3: The tree formed by the application of candidate path offset functions with $r = 0.8$ and $i \in \{0, \ldots, 9\}$ in both directions. Evaluation of candidates ends at a lookahead distance of 4 m.

Note that, due to the way offsets are accumulated as a 2D translation when an offset function is applied, this path centering compensates for GPS induced offsets in the position estimate which is essential for operation of the robot despite multipath and atmospheric errors in the GPS signal.

11.3.3 Obstacle Test

The obstacle test needs to check whether a candidate path segment can be traversed without encountering an obstacle. Recall that the local obstacle map is computed based on the "smooth coordinates" technique described in section 10.2 on page 91. Thus, the first thing the obstacle test does is to translate

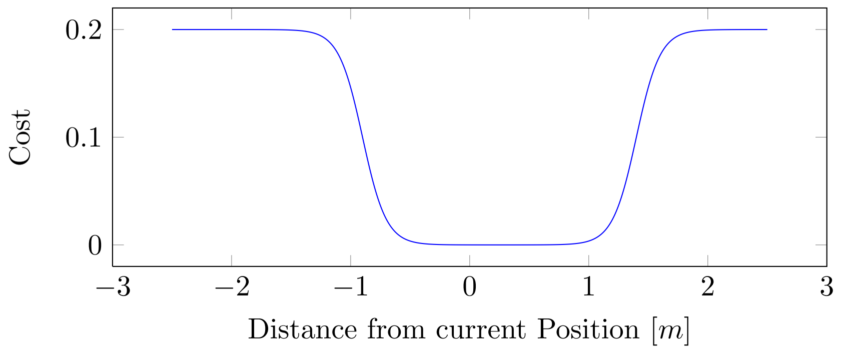


Figure 11.4: Valley function with sensed path boundaries at -1 m and 1.5 m relative to the current position perpendicular to the path.

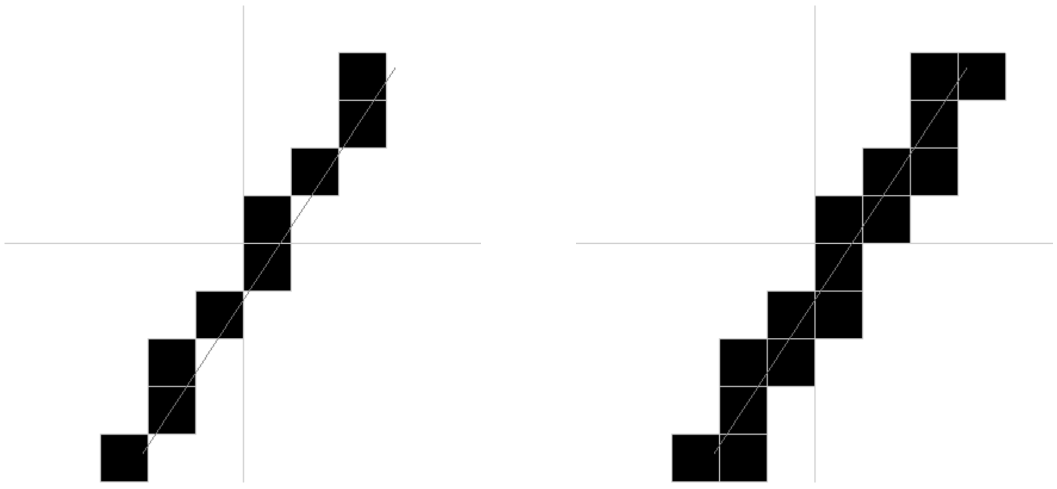


Figure 11.5: Bresenham line (left) vs. supercover line (right). The Bresenham line is visually pleasant but does not activate some cells although the straight line intersects them.

the start and end point of a path segment into the smooth map coordinate system by adding their distance from the current robot position in regular coordinates to the robot pose in smooth coordinates. Then it checks whether the path segment intersects a cell in the local obstacle map that has been marked as an obstacle.

This is similar to the well known problem of rasterizing a line in computer graphics so it can be displayed as a set of pixels in a raster image (Hill & Kelley, 2007, Sec. 9.2) which can be solved efficiently using Bresenham's line algorithm Bresenham (1965). However, as illustrated in Figure 11.5, Bresenham's line algorithm yields a visually pleasant line but is not sufficient for our purposes: For some cells (pixels) there is an intersection with the straight line but they are not part of the Bresenham line. To catch those cells as well, we need a modified version of the algorithm, called supercover line by Dedu (2001), which considers more than a single cell at each time step and is thus able to activate all intersected cells (Figure 11.5, right).

Unfortunately, there is one problem left still: Both, Bresenham's line algorithm and the supercover line algorithm as presented by Dedu (2001) work with the mid-points of each cell. We, however, need to handle lines starting and ending at arbitrary points within a cell. This can be solved as follows. The core of the algorithm deals with line slopes in the first octant – all other octants can be mapped onto this case. At each time step this part advances the x-coordinate by one to get to the next column of cells and we can simply change it to keep track of where the line intersects the boundary to the next column of cells – still within the current row of cells or beyond that boundary. The current implementation works with floating point arithmetic but this has never been the source of any performance issues on the SAMSMobil.

11.4 Path Control

The path controller on the SAMSMobil is based on the algorithm described by Frese et al. (2008b). The basic idea is that the robot aims for a point a certain distance ahead of the current position on the path trajectory. In our setup the path is initially the goal path. Later on the controller receives path offset changes from the planner and adjusts the path accordingly.

Every time the path controller receives a new pose estimate it performs the following steps (Frese et al., 2008b):

- The current position $p = (x, y)^T$ in the plane is projected onto the path to yield a point p'.
- Another point $p'' = (x'', y'')^T$ is determined by starting at p' and following the path for a certain lookahead distance $d = v_0 \cdot l$.
- The angle to p'' is computed as follows:

$$\vartheta = \text{atan2}(y'' - y, x'' - x) \tag{11.2}$$

- The new translational velocity v is then

$$\bar{v} = v_0 \cdot \cos(\text{normalize}(\vartheta - \psi)), \tag{11.3}$$

  where ψ is the current orientation in the plane and normalize() normalizes the angular difference to be within $[-\pi; \pi)$.

The SAMSMobil path controller uses a baseline velocity parameter of $v_0 = 0.5 \frac{m}{s}$ and a lookahead parameter of $l = 1m$ and passes the new translational velocity v and rotational velocity $\omega = \text{normalize}(\vartheta - \psi)$ as target values to the low-level PID controller on the STM32 (via Isil and the `stm32bridge`). Note how the rotational velocity term effectively defines a P-controller.

Chapter 12

Experiments

This chapter presents a set of experiments conducted to evaluate the performance of the software system developed as part of this thesis. We will start with tests of the mapping, planning and path control components in the simulation environment. Next is the evaluation of the state estimation and mapping components based on log data collected in outdoor experiments. Finally, we will discuss a set of autonomous outdoor runs.

12.1 Experiments in the Simulation Environment

The simulation environment was used to test the mapper, planner and path control components individually and in combination to make sure the interaction between them works out as intended. For this purpose a test course consisting of a worst case scenario has been designed where a single obstacle avoidance response is not sufficient to traverse the path and a second obstacle only becomes visible while the first avoidance action is still in progress. A visualization of the chosen obstacle course in Google Earth is depicted in Figure 12.1.

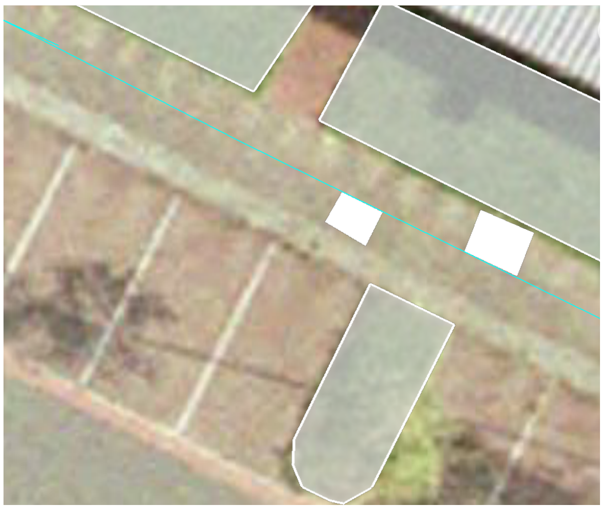


Figure 12.1: The obstacle course is defined as a set of polygons in Google Earth. To get past the two solid white obstacles, the planner needs to apply multiple different offset functions.

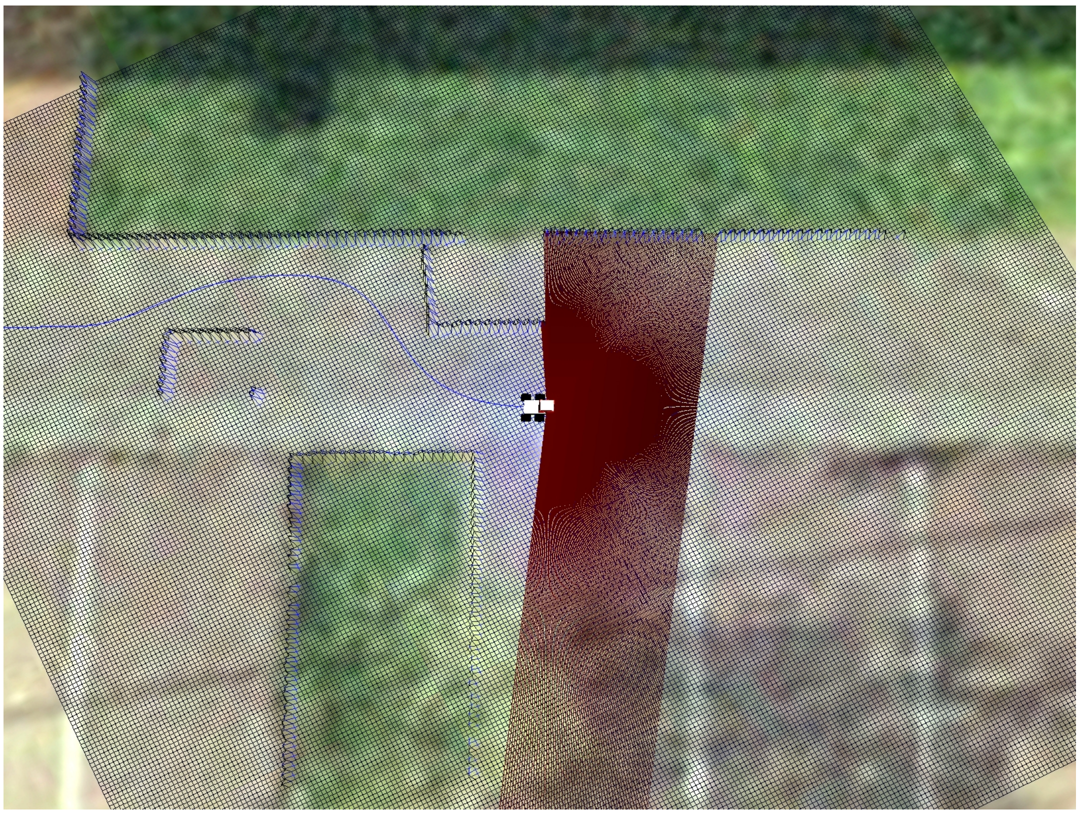


Figure 12.2: The robot after traversing the obstacle course in simulation. The local map is visualized as a wireframe surface.

As illustrated in Figure 12.2, the robot successfully traverses the obstacle course. Although both obstacles are detected fairly late due to the limited effective range of the laser range finder, the planner finds a smooth trajectory which is then executed by the path controller. It should be noted that in simulation, too, the *guard* component is active and would have triggered an emergency stop if the robot had got too close to any obstacle for the robot to safely come to halt. A sequence of step-by-step visualization screenshots of the simulation run can be found in Figure 12.3 on the following page.

Not shown in the figures is the fact that afterwards the robot traverses the complete simulated “Robert-Hooke-Straße” test course (c.f. Figure 5.1 on page 26 and Figure 12.4 on page 119).

12.2 Outdoor Experiments

In this section we will evaluate the performance of the system as a whole based on outdoor experiments.

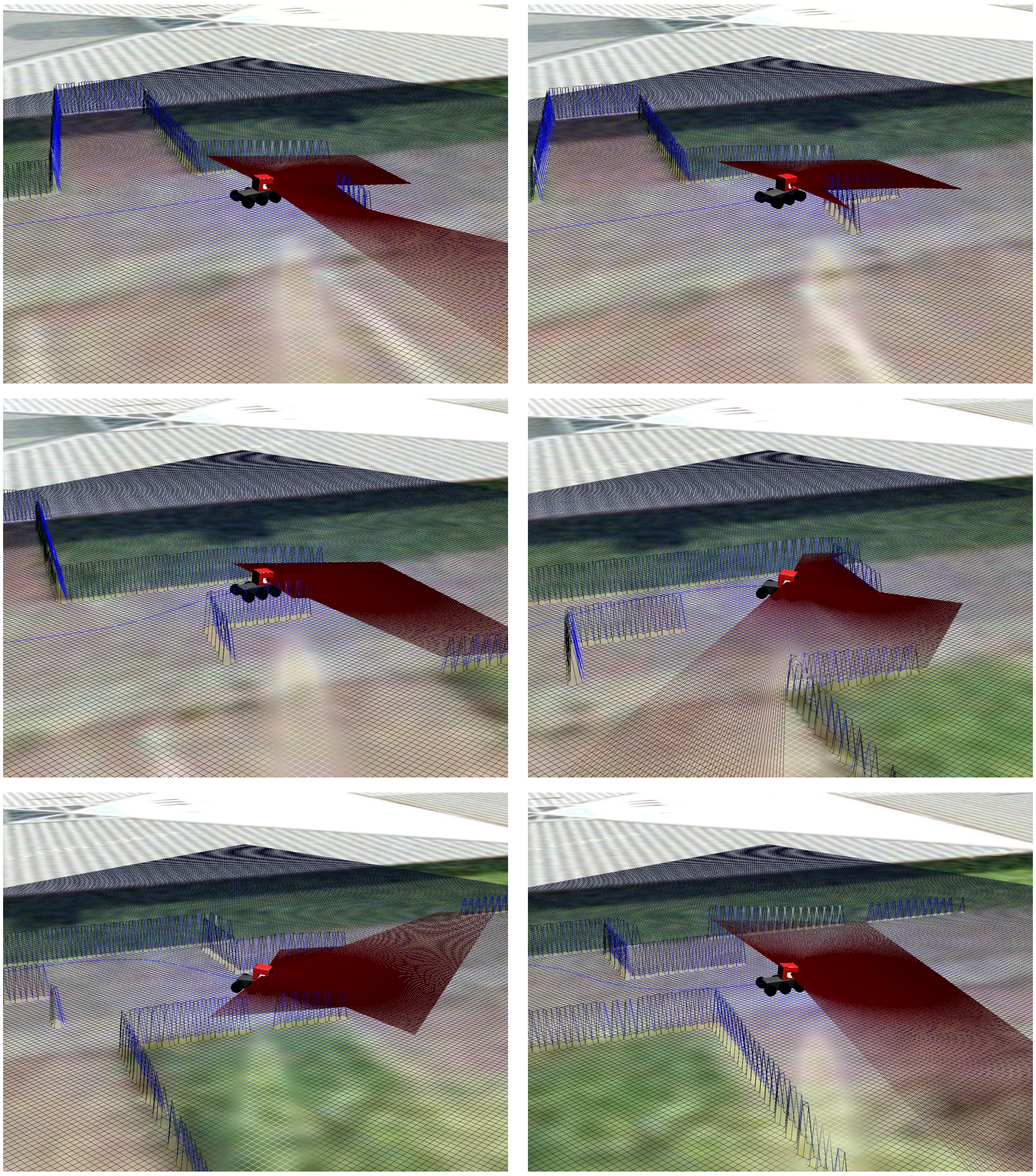


Figure 12.3: A sequence of visualization screenshots (top-left to bottom-right) showing the simulated robot as it succesfully negotiates the two test obstacles.

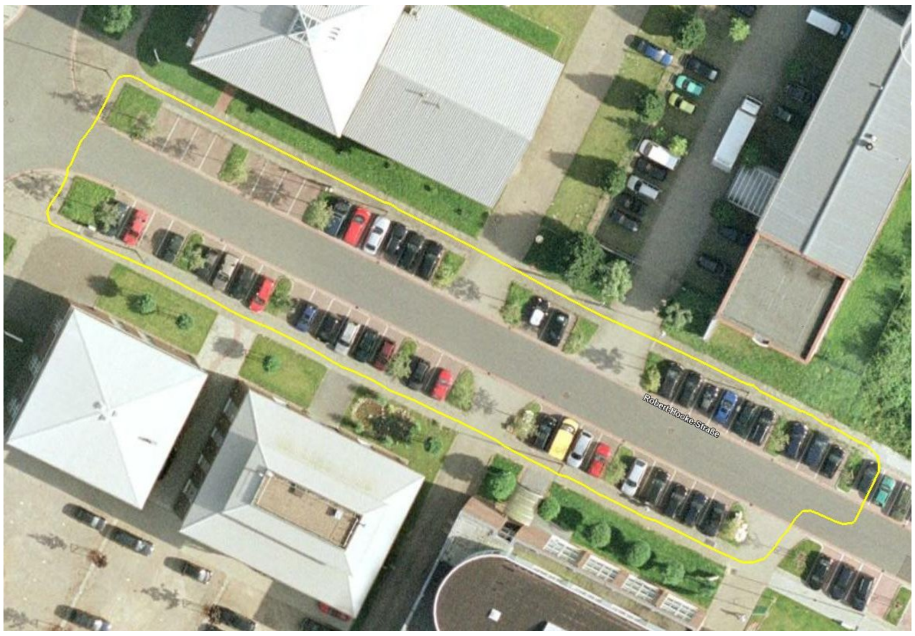


Figure 12.4: The main test track in Google Earth

12.2.1 The Test Site

A prerequisite to conducting any outdoor experiments was the selection of an adequate test site. Potential test sites had to meet the following requirements:

- Classification of drivable vs. non-drivable terrain must be possible from laser range scans. Preferrably, a drivable path should be bounded by non-drivable patches to the left and to the right of the path.
- The path should not involve public roads for safety reasons.
- Similarly, use by cyclists and pedestrians should be minimal.
- The area of the test site should be covered by reasonably recent satellite/aerial imagery on Google Maps/Earth so that state estimation results can be compared to a ground truth trajectory. The path must also be visible (i.e., not obstructed by overhanging trees, etc.) from above to make this possible.
- The test site should not be too far away from the university campus to reduce the transportation overhead.

Initially, a track through the "Bürgerpark", a public park in Bremen, had been selected as a test site, but it fails to meet the latter three requirements. Instead, tests were performed in the "Robert-Hooke-Straße", one of the less busy streets at the northern edge of the university campus. A screenshot of Google Earth visualizing the path the robot was instructed to follow in the autonomous runs is depicted in Figure 12.4. The same path is plotted in Figure 12.5 on the following page. The plot view will be used as the primary visualization method throughout the remainder of this chapter for better readability. It should be noted that the path length must be seen in relation to the size of the robot which is about ten times smaller than a regular car.

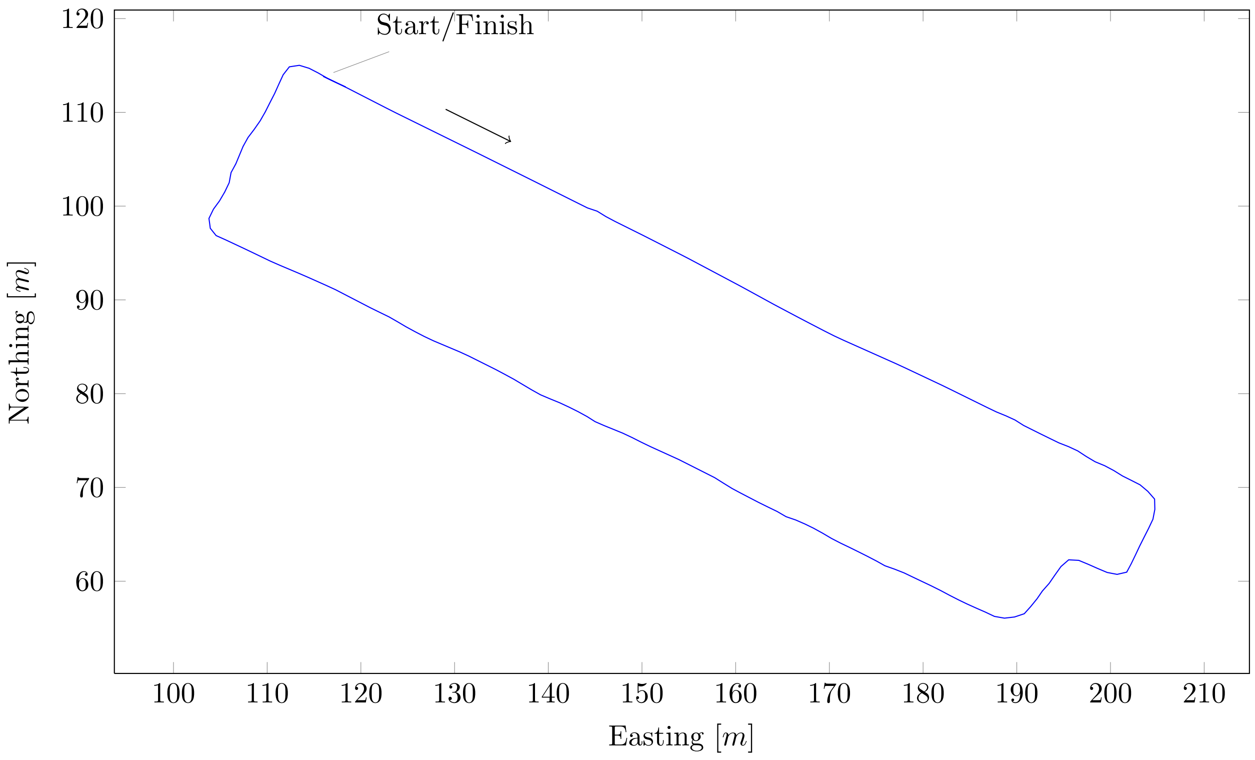


Figure 12.5: Plot view of the 246m long target trajectory which the robot is instructed to traverse.

12.2.2 State Estimation Evaluation

The first large component to be evaluated of the navigation approach implemented on the SAMSMobil is the state estimation or localization. For the overall system to have a chance of working the robot must be able to localize itself up to a precision similar in magnitude to the width of the path.

The experimental setup consisted of several manual runs along the test track (Figure 12.4 on the previous page). The robot was steered via the gamepad interface by one person while another person filmed the SAMSMobil with a digital camera. Although the then current version of the state estimation software was active during the test runs and thereby provided an initial idea of the state estimation quality all sensor inputs were also logged for later offline processing.

The following subsections will use these logged data sets to analyze the state estimation performance in several configurations that differ in the type of sensor input taken into account as shown in Table 12.1 on the facing page.

Localization Performance without GPS

The first thing to look at is how well the state estimation filter works without access to positioning information from GPS. This applies to the filter configurations A and B (Table 12.1 on the next page).

Configuration	A	B	C	D
3D-Odometry process model	√	√	√	√
Gravity measurement model	√	√	√	√
Magnetic field measurement model		√		
GPS measurement model			√	√
Path boundary measurement model				√

Table 12.1: Active process and measurement models in the different filter configurations.

In configuration A the filter solely relies on its 3D-odometry process model, i.e., it takes nothing but wheel encoder and gyroscope data into account. The estimated trajectory of test run 2 in this configuration is plotted in Figure 12.6 on the following page along with the position and orientation uncertainty (i.e., the square root of the respective covariance matrix diagonal entries). Note that after every 15 m travelled the trajectory is annotated with timestamps which can be used to match data in the different sub-plots.

Of particular interest about the plots is that the estimated position degrades as errors accumulate. This is expected, as is the fact that (at first) the position uncertainty grows without bounds. Perhaps less expected are the peaks in σ_x and σ_y. Both result from turns the robot takes at the respective time steps.

The fact that σ_ψ grows linearly after a short period of time may be surprising at first since one would expect the addition of process noise to the covariance matrix alone to result in a growth proportional to $\sqrt{t}$. However, the respective bias terms also influence the orientation uncertainty and cause nonlinearities which the filter takes into account when estimating the covariance.

Without prior knowledge of the inner workings of the filter the peak in the orientation uncertainty (standard deviation) around the global z-axis (σ_ψ) may also come as a surprise. It occurs at $\sigma_\psi = \pi$ which as described in section 9.8.3 on page 79 corresponds to a "wraparound" in the sigma point representation of the estimated probability distribution. Beyond this point, the filter output should be treated as inherently undefined, particularly when it comes to the orientation around the z-axis.

Given the above, the obvious conclusion is that uncertainty in the estimate of the global z-orientation must be kept well below π either by direct observation through magnetometer measurements or indirectly through GPS position measurements. Figure 12.7 on page 123 shows the estimated trajectory obtained with magnetometer measurements enabled (configuration B; Table 12.1). This achieves the desired effect of keeping the z-orientation uncertainty within fairly strict bounds. Unfortunately, the magnetometer measurements are only locally useful. If the robot doesn't change its direction significantly the estimated trajectory comes out reasonably straight. The overall trajectory, however, suffers from the fact that the magnetometers are obviously not correctly calibrated. The calibration routine provided by XSens is supposed to address this issue but could not be used as discussed in section 7.4.3 on page 50.

GPS Quality

Before looking at incorporating GPS measurements into the filter estimate it is necessary to analyze the raw GPS quality. Plots of the positioning solutions generated by the Navilock GPS receiver during three different test runs are depicted in Figure 12.8 on page 124. The data was acquired as follows:

- The atmospheric conditions that day can be best described as "sunny with few clouds".

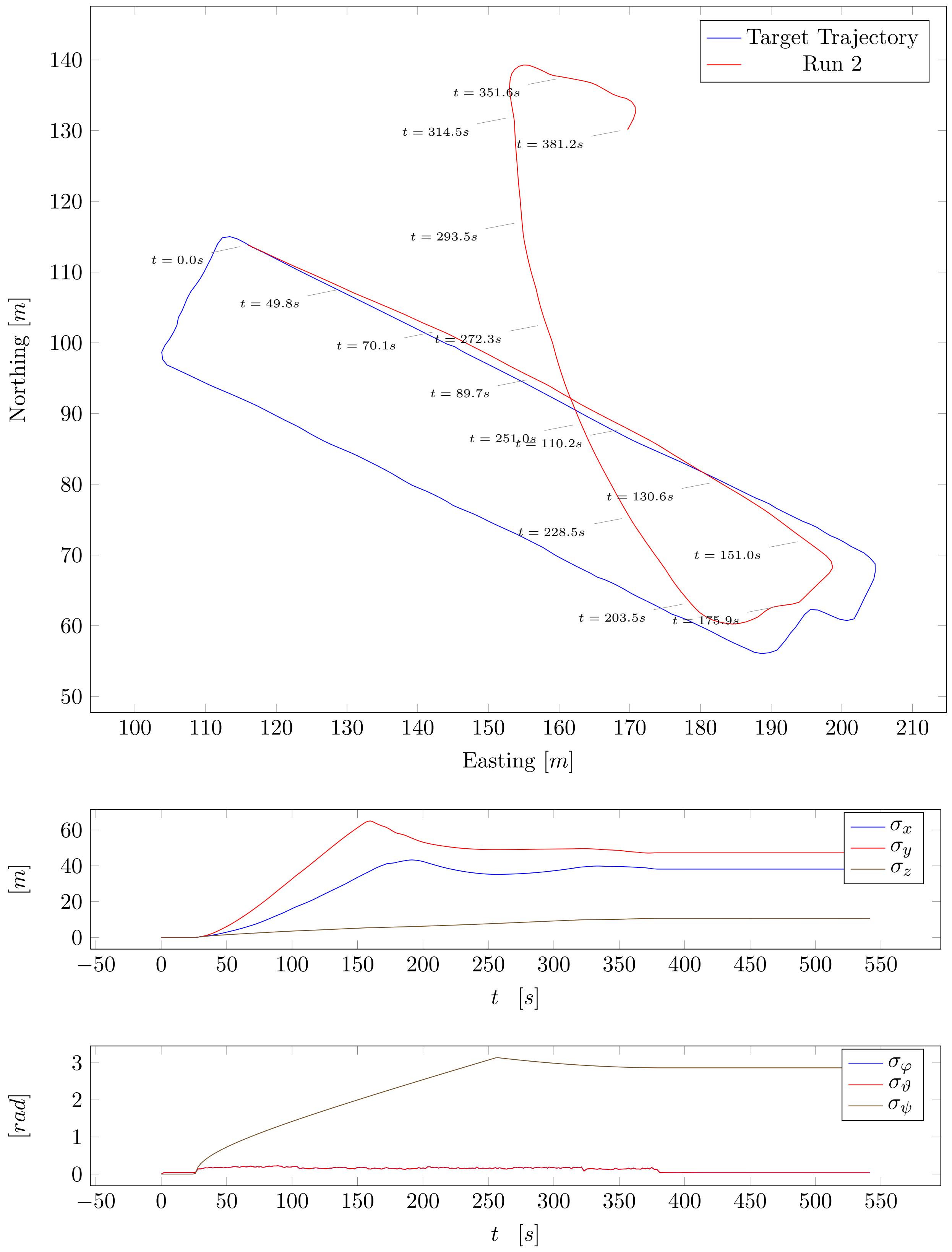


Figure 12.6: UKF position estimates with GPS and magnetometer measurements disabled. Note that, on the scale of the graph, σ_φ and σ_ϑ overlap as both are dominated by the same accelerometer measurements.

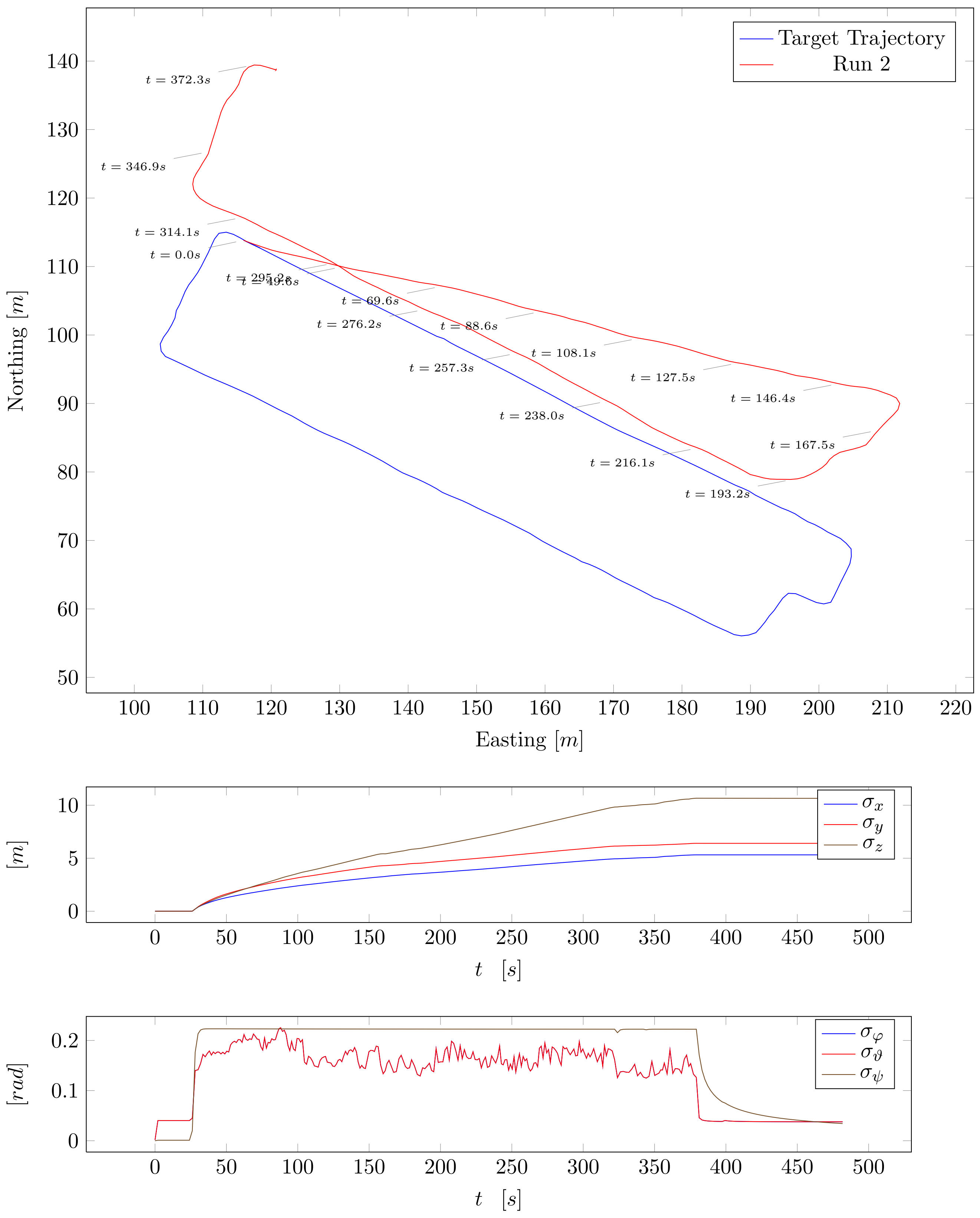


Figure 12.7: UKF position estimates with magnetometer measurements enabled and GPS disabled. Again, σ_φ and σ_ϑ overlap.

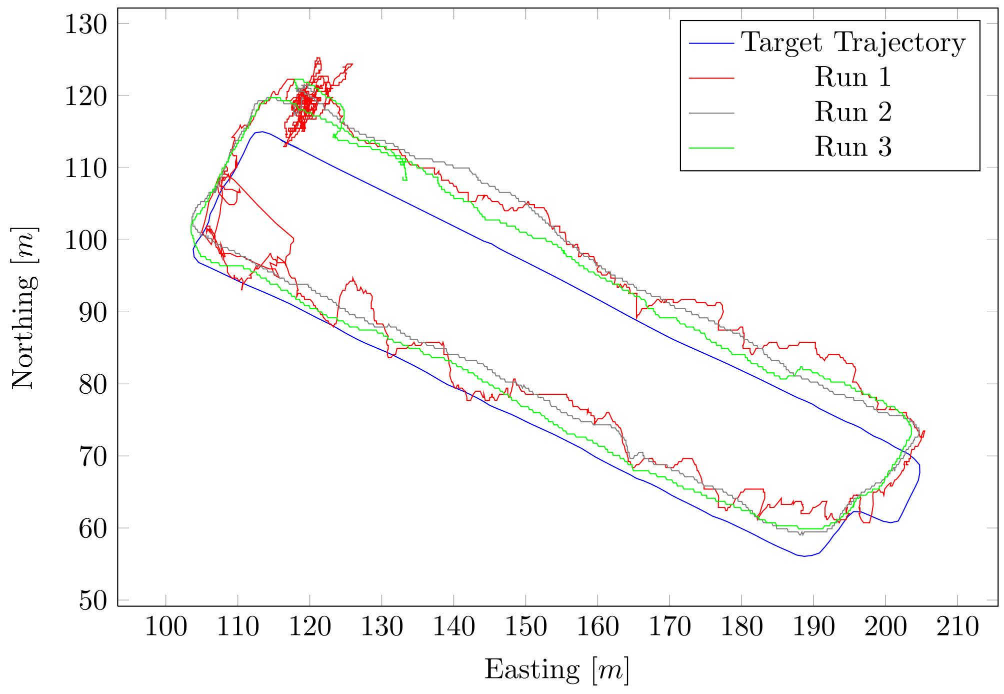


Figure 12.8: Raw GPS data from several test runs. The obvious quantization steps stem from the limited number of decimal digits transferred through the NMEA interface.

- The GPS receiver was configured to determine positioning solutions at a rate of 4Hz.
- For the first run the receiver internal motion model was set to "Airborne – 6g".
- All following runs used the "Automotive" motion model.

The idea behind choosing the "Airborne – 6g" motion model first was that it would provide the most direct access to the raw GPS data that is possible with the receiver in use since access to the actual pseudorange measurements (section 6.3.3 on page 34) is not possible. The system can only be operated in a loosely coupled setup wherein the receiver outputs positioning solutions calculated by its internal Kalman filter which is then incorporated by the UKF as a position measurement.

This has two effects which, unfortunately, make the overall localization performance worse rather than improving it. First, the receiver internal filter uses a much less restrictive motion model when configured as "Airborne – 6g". The resulting positioning solutions have the desired property of quickly reflecting actual position changes (which is particularly useful whenever the robot turns). However, in case of atmospheric, multipath or other errors in the pseudoranges (section 6.3.6 on page 37) the receiver-internal filter will now also be much more likely to accept these as legitimate measurements because the motion model allows it. The net result is that we see more errors in the positioning solutions generated by the receiver.

The second effect is caused by a violation of the requirements we postulated for UKF measurements. The plots in Figure 12.8 clearly illustrate that the combined GPS errors cannot be correctly modelled as white Gaussian noise added to the position as previously discussed in 9.8.9 on page 85. It is difficult to attribute individual error sources to anomalies in the plots but a comparison with the position of tall buildings in Figure 12.4 on page 119 suggests that multipath errors and/or loss of satellite visibility

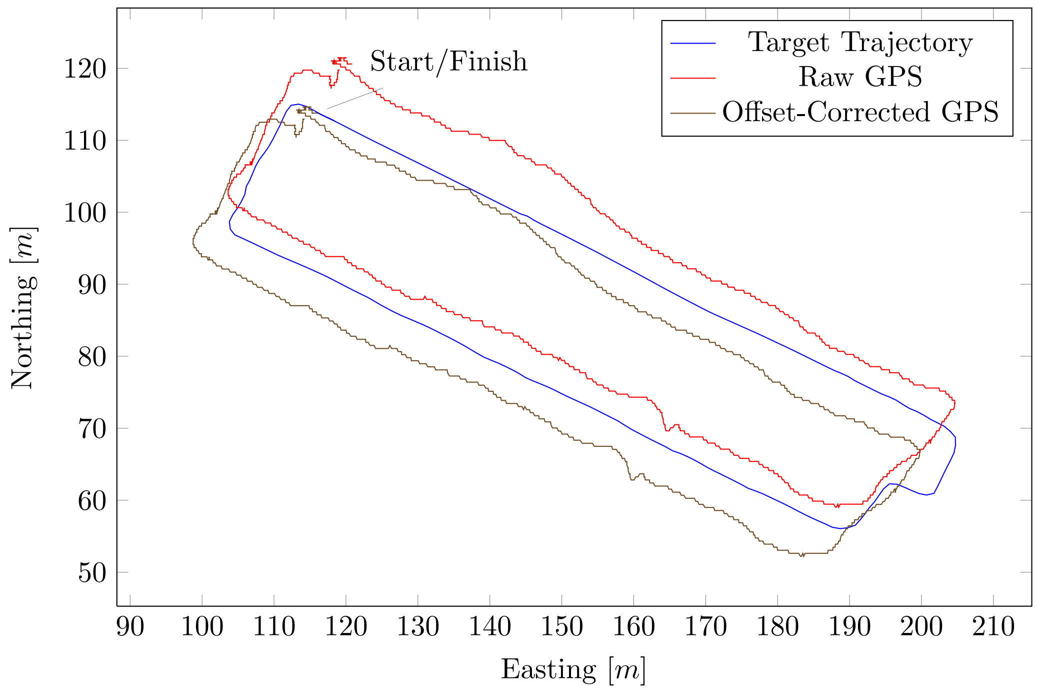


Figure 12.9: Raw vs. offset-corrected GPS. The offset correction compensates for a constant offset determined at the known initial position.

play a predominant role. We will get back to this when analyzing the UKF performance with GPS enabled in 12.2.2 on the next page.

As far as atmospheric errors are concerned, at the time the logs were recorded EGNOS was still in its test phase and disabled on the SAMSMobil upon recommendation by the receiver chip vendor, uBlox AG (Paparazzi Project, 2009). The result of the offset-correction step which assumes that atmospheric effects are constant in a spatially limited area over a relatively short period of time (c.f. 9.8.9 on page 85) is illustrated in Figure 12.9. Multipath effects at the start position particularly affect the result negatively in the presented test scenario[1].

Up to now we have only looked at 2D GPS data. GPS receivers also provide altitude information which is, however, inherently less precise due to the geometric configuration of GPS satellites relative to the receiver. Figure 12.10 on the next page shows plots of the altitude data logged during the three test runs. Again, the effect of the “Airborne – 6g” receiver-internal motion model is strikingly obvious from the comparison of run 1 with the others. Ground truth elevation data is unfortunately not available[2], but it is clear that the altitude changes even in runs 2 and 3 (which use the “Automotive” receiver-internal motion model) are unrealistic. Combined with the fact that the GPS receiver may (depending on the geometric configuration of the visible satellites among other factors) decide not to output a 3D GPS fix (although this didn’t happen in the test runs shown in the plots) this resulted in the decision to use GPS for 2D position measurements only.

[1]Note the tall building north of the start position in Figure 12.4 on page 119.

[2]Google Earth does report an altitude ranging from -1m to 1m in the test site area but it is unclear whether this was derived from actual elevation data.

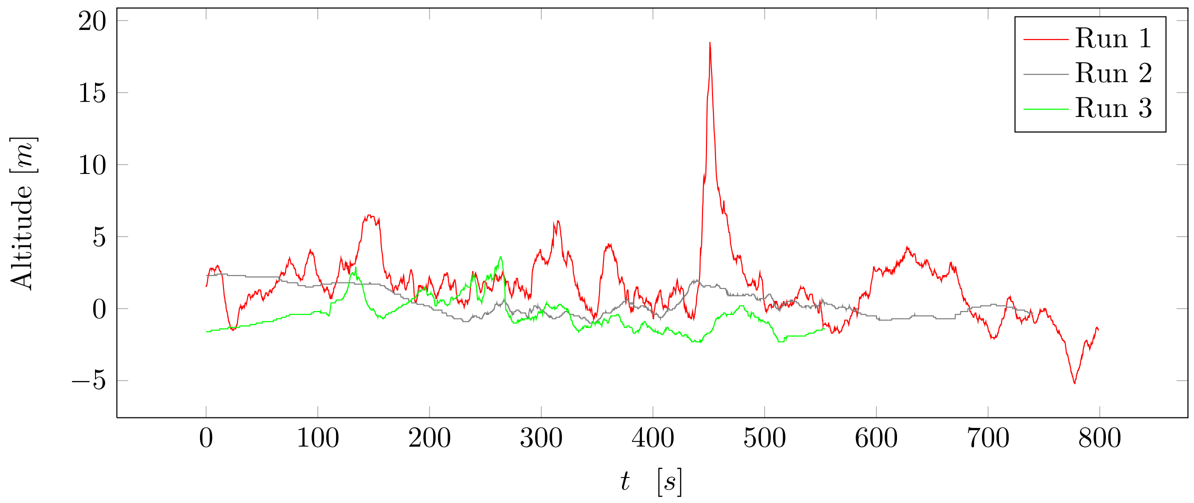


Figure 12.10: Raw GPS altitude data from several test runs.

UKF Performance with GPS

With the above analysis of the GPS positioning solution quality in mind it is now time to look at how its use in the UKF influences the overall localization performance in filter configuration C (Table 12.1 on page 121). The resulting trajectory is depicted in Figure 12.11 on the next page. It should be noted that the GPS measurement model uses the offset-corrected 2D GPS data as measurements.

Looking at the plots the first impression is that the UKF successfully merges the 2D GPS measurements into the combined state estimate. Short term GPS errors that do not match the process model are effectively ignored leading to a fairly smooth trajectory. Since no other measurement covers the global z-orientation GPS dominates the global shape of the trajectory. Similarly, GPS measurements keep the z-orientation uncertainty under control and avoid undefined filter behaviour.

Closer inspection of the position estimates reveals that they come very close to the ground truth trajectory up to about 60 seconds into the log. Then the combination of two factors makes the estimated trajectory deviate – the robot reaches the end of the tall building next to the start of the test track which changes the multipath error situation and sufficient z-orientation uncertainty has build up to allow GPS measurements to change the orientation. The high GPS update rate (4Hz) and the fact that the GPS output is still plausible given the process model [3] also have a strong influence on the state estimation outcome for this part of the test track. Later on we primarily observe a constant offset in the position estimates. Other significant deviations from the reference trajectory are mainly due to the fact that the human driver chose a slightly different path, particularly at 150 seconds and 310 seconds into the log.

Overall it is obvious that in filter configuration C (magnetometer measurements disabled, 2D GPS enabled) it is not possible to reach a positioning precision of the same magnitude as the path width with a consumer-grade GPS receiver. Whether access to the raw pseudorange measurements (tightly coupled filter setup) would be sufficient or whether a survey-grade GPS receiver is needed to achieve

[3] See the plot showing offset-corrected GPS in Figure 12.9 on the preceding page.

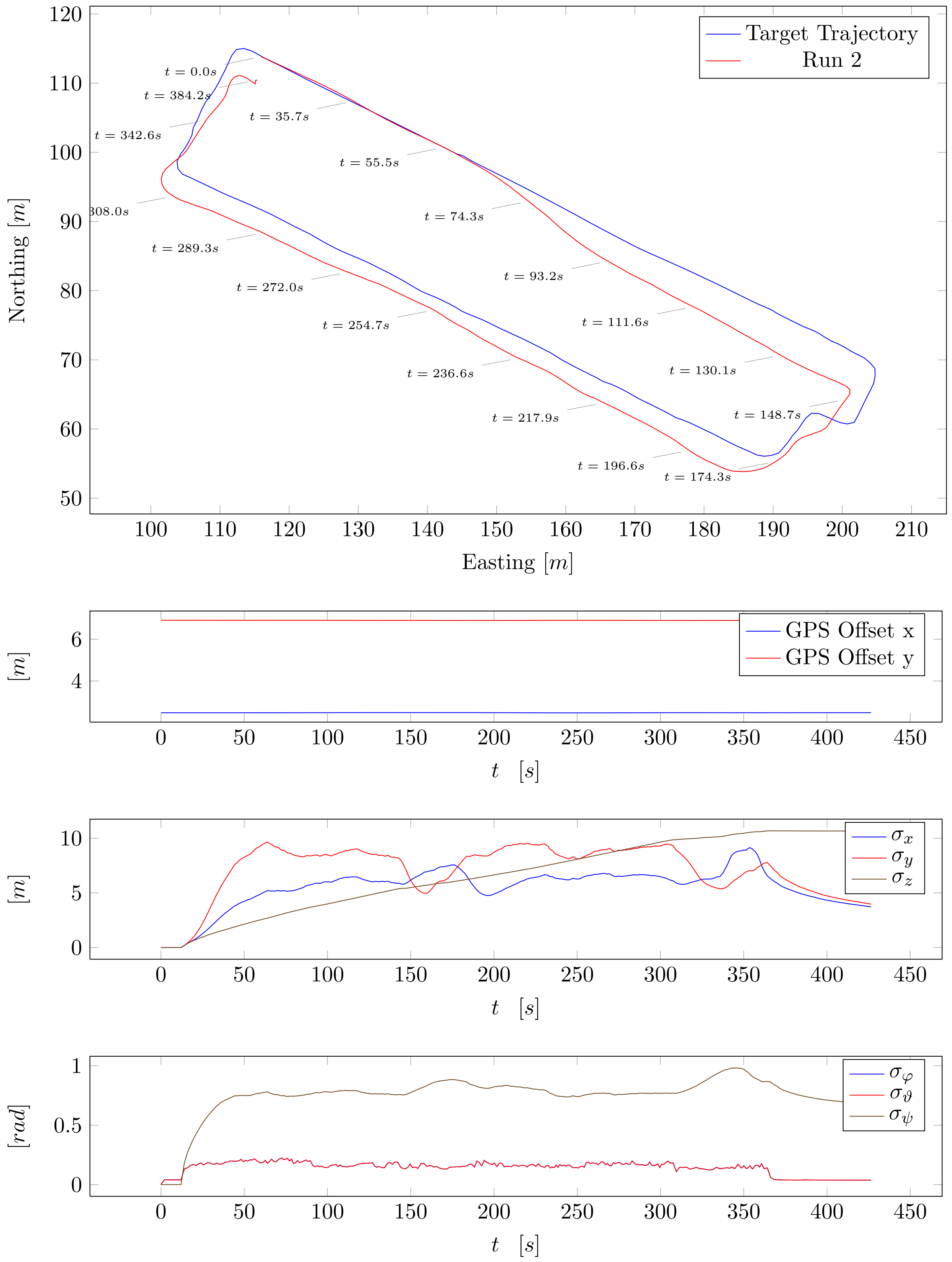


Figure 12.11: UKF position estimates with GPS measurements enabled. Here, too, σ_φ and σ_ϑ overlap.

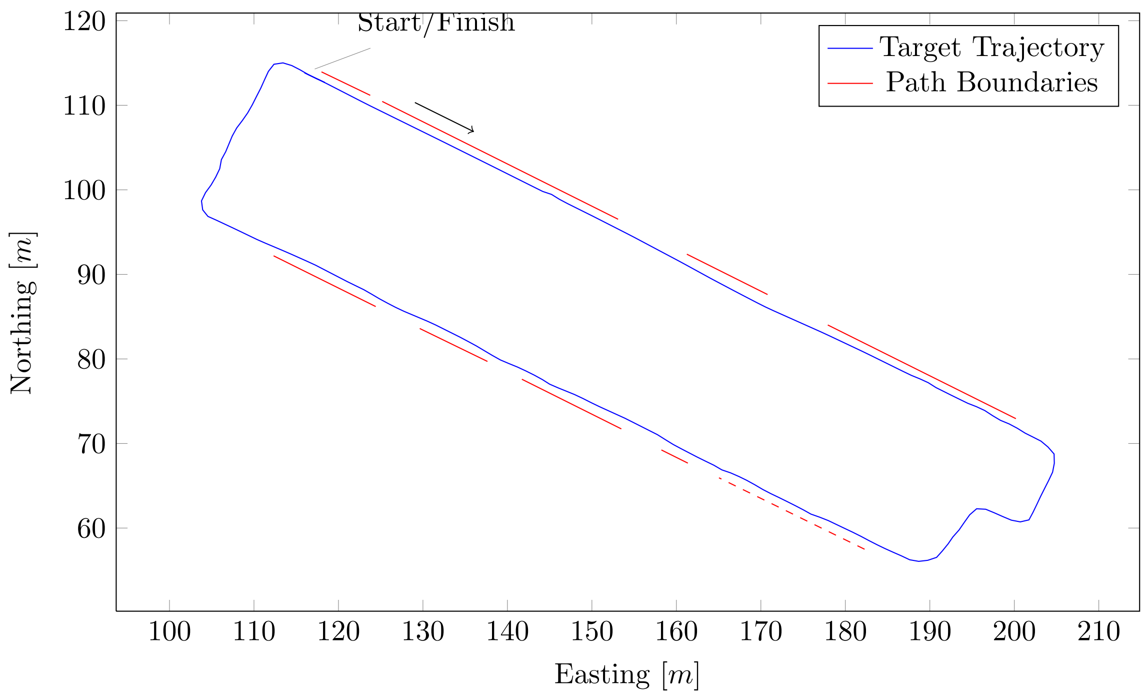


Figure 12.12: Path boundary map. The dashed line represents a disabled path boundary. See text for details.

this goal cannot be determined without tests with the respective hardware (which was not available on the SAMSMobil).

UKF Performance with GPS and Path Boundary Measurements

To further increase the localization accuracy we developed the path boundary measurement model in section 9.8.10 on page 87. The key idea is to supply the UKF with a map of detectable path boundary segments (Figure 12.12) which can then be used as part of a measurement model to merge the distance to a path boundary detected by the mapper into the combined state estimate.

The resulting trajectory is depicted in Figure 12.13 on page 130. Apart from two minor issues caused by the mapper reporting slightly misplaced path boundaries the trajectory is nearly exact during the first half of the log. Problems occur as the robot crosses the road: There are no path boundaries for a longer period of time causing the position and orientation uncertainty to increase significantly. The map on the far side of the road (c.f. 12.2.3 on page 134) unfortunately makes a reliable path boundary detection difficult. In combination with the uncertainty accumulated while crossing the road the first path boundary in the path boundary map on that side of the road would make the filter diverge which prompted its removal from the boundary map (Figure 12.12).

The remaining path boundary segments on the far side of the road are ignored either because the Mahalanobis distance check fails or because the measurement model is not applicable for all sigma points. As discussed in section 9.8.10 on page 87 the two virtual path boundary “sensors” work by computing the intersection of a line with the respective path boundary segment in the boundary map. As

3D-Odometry process model	265.175
Gravity measurement model	182.925
Magnetic field measurement model	106.594
2D GPS position measurement model	108.707
Path boundary measurement model	180.806

Table 12.2: Average computation time in μsec for the respective process and measurement models when operating on the log data from Run 2.

the position and/or orientation uncertainty goes up there may be sigma points for which no intersection occurs. In this case the UKF update step (c.f. section 9.6.3 on page 73) cannot be completed.

UKF Performance with GPS Projected onto the Path

A prerequisite to autonomous navigation with the SAMSMobil is that the pose estimation error must be at most in the order of the path width. With GPS and path boundary measurements enabled the filter achieves this until the robot crosses the road (Figure 12.13 on the following page). However, mapping errors can lead to pose estimation errors which may manifest as jumps in the estimate. To keep the risk to the SAMSMobil as low as possible a different strategy was chosen for the first autonomous runs. A note by Röfer (2009) about projecting GPS measurements onto an OpenStreetMap road network to gain ground truth for a vehicle moving on the road inspired the idea to use the goal path trajectory as a map and project GPS positioning solutions onto it.

The reasoning behind this is that, with the exception of the two times it crosses the road, the robot cannot deviate much from the path or it will get stuck in front of obstacles. Thus, the most likely position is always somewhere in the vicinity of the goal path. Also note that GPS multipath errors affect the positioning solution primarily perpendicular to the road while in the direction of the road the receiver output is much more precise. The assumed GPS measurement noise terms are, however, already large enough to express the fact that the projection onto the path only yields an approximation. A plot of the resulting trajectory is depicted in Figure 12.14 on page 131 and clearly shows that within the area around the goal path the filter still takes IMU and odometry data into account to estimate the precise pose. We also see that compared to the original filter configuration D (unmodified GPS only) the estimate deviates much less from the goal path.

Real Time Performance

Despite the comparably large size of the vectorized state space (18) the manifold-UKF implementation is easily real time capable on common PC hardware. On a 2.40GHz Intel Core 2 Duo 6600 with 4MB cache it takes less than a millisecond on average to run all process and measurement models combined when processing the log data from Run 2. The individual time intervals measured with the help of `gettimeofday(2)` are listed in Table 12.2.

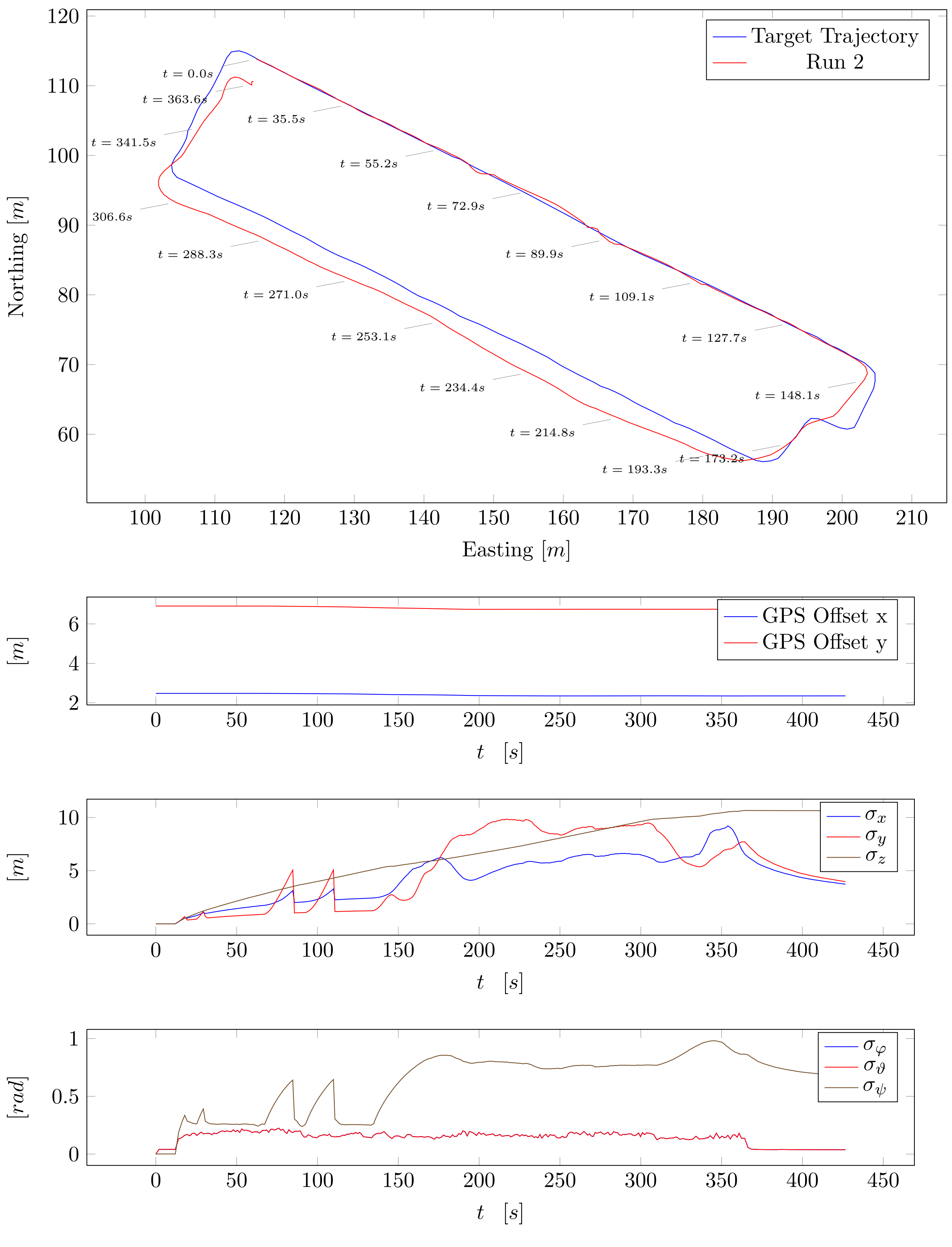


Figure 12.13: UKF position estimates with GPS and path boundary measurements enabled. Again, σ_φ and σ_ϑ overlap.

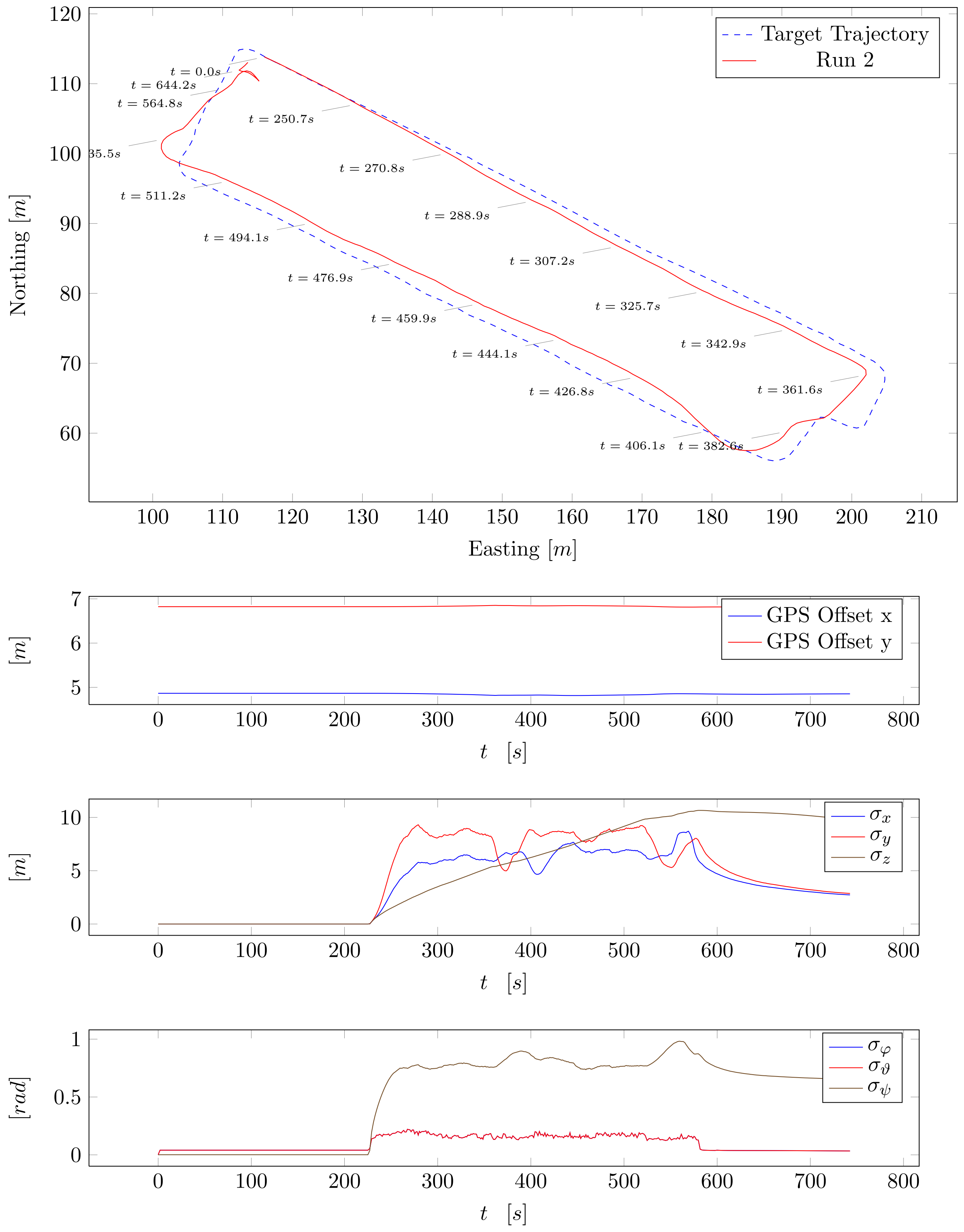


Figure 12.14: UKF position estimates with GPS projected onto the path. Again, σ_φ and σ_ϑ overlap.

δ	0.0105
α	0.095
σ_{xyz}	0.01
$\sigma_{\varphi\vartheta\psi}$	0.0
τ_{xyz}	0.01
$\tau_{\varphi\vartheta\psi}$	0.01

Table 12.3: The learned PTA parameter set which has been used to generated the PTA maps in this section.

12.2.3 Mapping Evaluation

After localization the second large component to evaluate is the mapper that was developed in chapter 10 on page 91. Under normal operation the mapper generates a local obstacle map only. For visualization purposes this section will work with graphical representations of global maps produced by the map_trainer tool from section 10.6.3 on page 98. These global maps are not globally consistent and never used on the robot but allow for a visual analysis of how well the classification of drivable vs. non-drivable terrain works.

All maps in this section have been generated from the log of test run 2 based on position estimates with the UKF set to configuration C. To avoid overlaps the last few meters of the path have been omitted from the maps. Each map is visualized as an image consisting of one pixel per 5x5cm^2 map cell. Unknown terrain is drawn in grey, known obstacles in red and known drivable cells in white.

PTA

The map generated by the implementation of Thrun et al.'s PTA algorithm developed in section 10.6 on page 95 is depicted in Figure 12.15 on the facing page. The parameters used are documented in Table 12.3. Before analyzing the map in detail a few points are noteworthy about the different environments the algorithm operates in on the SAMSMobil compared to Thrun et al.'s Stanley:

- The SAMSMobil is equipped with only a single laser range finder mounted at a very low position above the ground scanning a single low-angle plane while Stanley has an array of five roof-mounted laser range finders scanning multiple planes at different angles.
- Experiments with laser range finders of the same model as the SICK range finders used on Stanley indicate that the precision of the raw range data is significantly better than what the Leuze rotoScan ROD4plus on the SAMSMobil produces.
- The SAMSMobil is an order of magnitude smaller than Stanley requiring the detection of much smaller obstacles, yet the range scan precision is actually worse.
- Although the papers on Stanley do not specify what type of IMU was in use it is probably safe to assume that it was a higher precision device (i.e., Crossbow or similar).

All of these points make the classification task more difficult for the mapping algorithm in use. PTA is particularly affected because it looks at height differences and the possible range of these is very limited due to the low position the laser range finder is mounted in .

With the above in mind it may be less surprising that PTA does not fare too well on the SAMSMobil. It should be noted that the map in Figure 12.15 on the facing page was generated using a parameter set learned from the same data set to achieve this level of classification performance. Still, PTA fails to

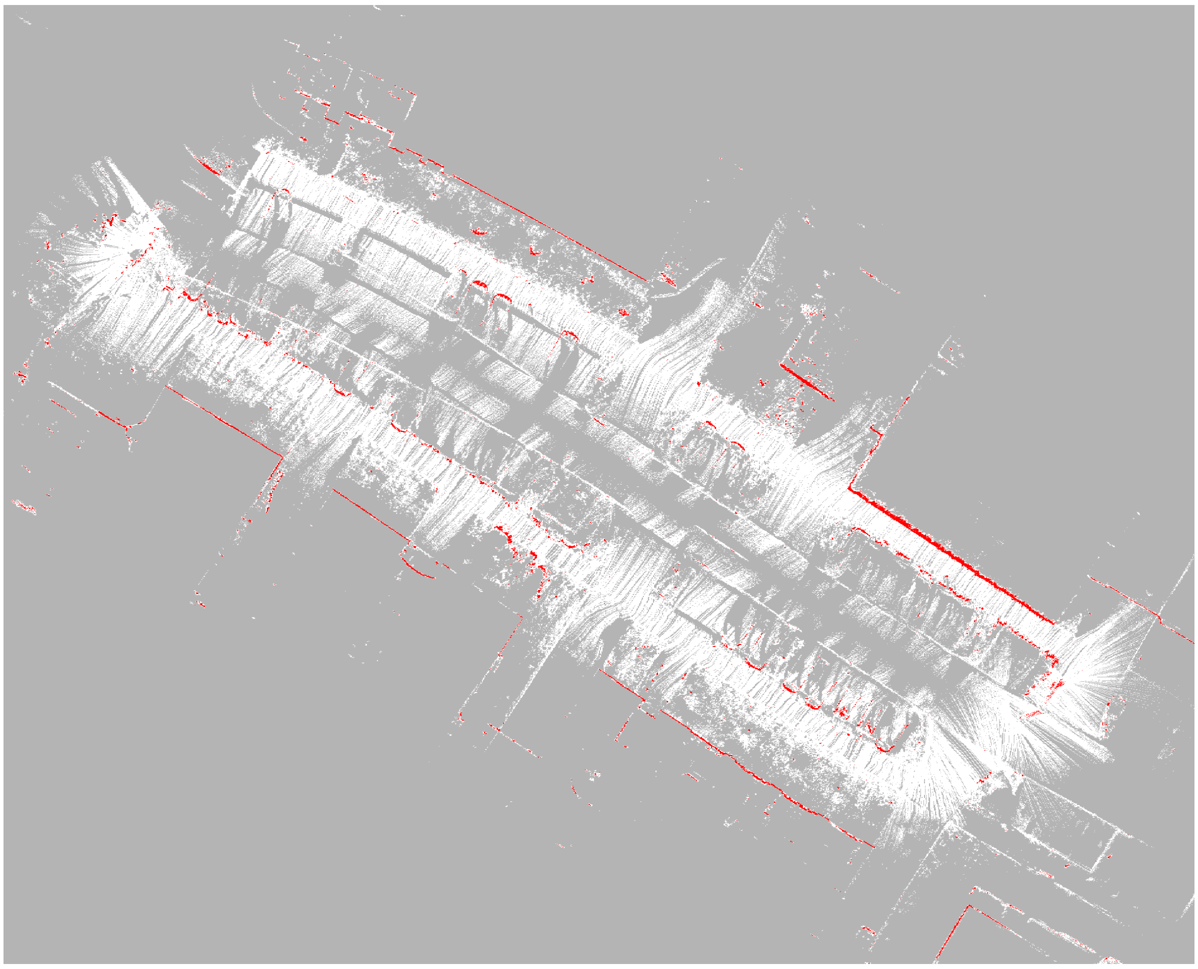


Figure 12.15: Global PTA map.

detect most path boundaries. Patches of grass are rarely detected as obstacles at all (12.16(a) on the next page). Only cars (12.16(b) on the following page), building walls (Figure 12.15), hedges (12.16(c) on the following page) and larger bushes are detected reliably.

Negative obstacles such as curbs seen from the sidewalk cannot be detected either since the low angle under which they are "seen" results in the top of the curb being entered in a map cell far away from where the road surface below ends up in the map. PTA, however, only takes height differences in a confined neighborhood of 3x3 cells into account and "misses" the curbs which only manifest as unknown gaps in the map (12.16(a) on the next page).

One might expect that PTA at least would not cause false positives in the sense of drivable cells detected as obstacles. Unfortunately, there are false positives – in an area where the robot drives down a ramp from the sidewalk in order to cross the road (12.17(a) on page 135). Even though the ramp is far from being considered steep by a human its gradient triggers the height difference test: The same 3x3 neighborhood that was too small to detect curbs (see above) is now large enough (15x15cm^2 in the SAMSMobil configuration) for a small gradient to generate height differences exceeding the respective

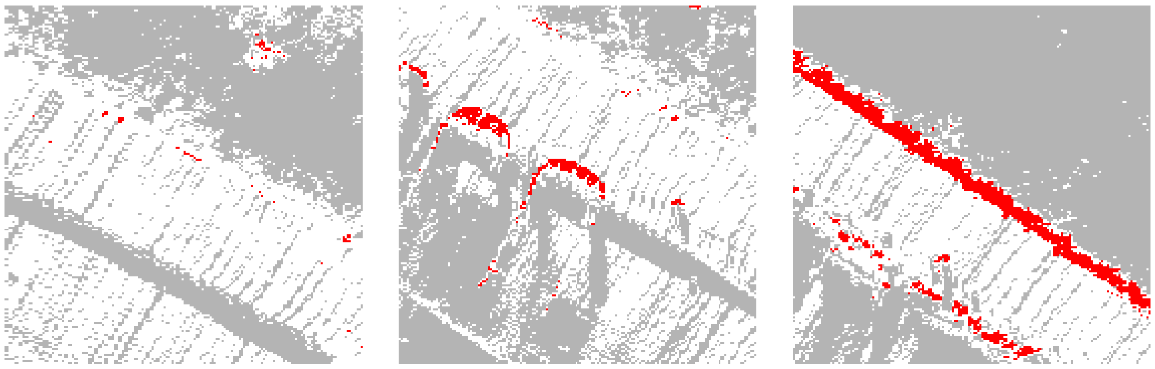

(a) Neither most of the patch of grass (above) nor the curb (below the path) are detected as obstacles.
(b) Cars are detected reliably.
(c) A hedge (above the path) as seen by PTA.

Figure 12.16: Excerpts from the global PTA map.

threshold.

The fact that PTA classifies the transition from the ramp through the gutter onto the road (12.17(a) on the next page) and vice versa on the far side of the road (12.17(b) on the facing page) as drivable is a bit of a double edged sword: It allows the robot to traverse this part of the test track although it is at the edge of what the vehicle can physically handle (). Careful manual control was required to get across the gutter without the SAMSMobil toppling over.

Overall it is clear that, with the sensor setup on the SAMSMobil, PTA is not able to generate a map that would allow for safe obstacle avoidance.

GEM

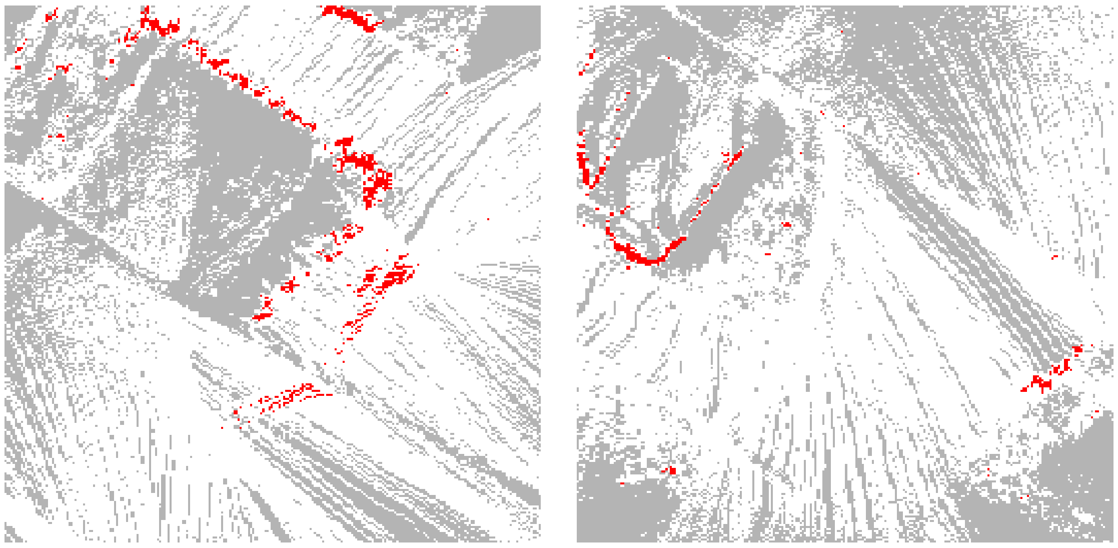
GEM is the mapping algorithm developed as part of this thesis in section 10.7 on page 101 to overcome some of the limitations of PTA. The global map generated by GEM is depicted in Figure 12.18 on page 136. The GEM parameters used are documented in Table 12.4.

Most importantly, GEM is able to detect virtually all path boundaries, particularly, patches of grass and negative obstacles such as curbs (12.19(a) on page 137). The ability to detect cars (12.19(b) on page 137), hedges (12.19(c) on page 137), bushes and other obstacles also found by PTA is retained.

The fact that there is a lot more unknown terrain in the GEM map results from a combination of two factors: First of all GEM requires a certain amount of evidence to be collected for each cell while PTA

α	0.5
γ	0.2
δ	0.7

Table 12.4: The GEM parameter set which has been used to generated the GEM maps in this section.


(a) Part of the ramp from the sidewalk (top) onto the road (bottom) is incorrectly detected as obstacles.

(b) The ramp from the road (top-right) onto the sidewalk (bottom-left) on the far side of the road.

Figure 12.17: More excerpts from the global PTA map.

already marks a cell as known with the first laser range scan endpoint falling into a cell. Secondly, the robot "sees" a lot of terrain due to the slightly elevated position on the sidewalk. While PTA simply registers the corresponding laser range scan endpoints in the map GEM takes them as evidence of negative obstacles and registers them where it expected to find the ground as discussed in section 10.7 on page 101.

As for the ramps that were problematic for PTA, parts of the ramp that leads from the sidewalk down to the road are first correctly detected as negative obstacles while the robot is still on the sidewalk. As the robot approaches the ramp it re-classifies these parts as drivable as can be seen from the horizontal drivable stretches that overlap a vertical obstacle stretch in 12.20(a) on page 137. On the far side of the road this situation does not arise as the ramp is never seen from the sidewalk and the ramp is classified as drivable.

However, the way GEM classifies the gutter on both sides of the road is different from PTA – GEM considers both areas as obstacles. This is in contrast to the fact that (with careful manual control) the robot was able to traverse the gutter but correct in so far as the gutter violates the requirements postulated for drivable terrain in section 10.7 on page 101. The area around the gutter is not always seen as smooth up to where the laser scan plane intersect with the ground: At the latest when the robot enters the gutter with its front wheels it tilts forward and the new orientation is detected by the pose estimator. Now, GEM expects to find terrain in front of the robot with roughly the same gradient as in the robot's current position as would be the case with, e.g., a ramp. The gutter does not fulfill this smoothmess criterion and is classified as an obstacle by GEM. .

With this decision GEM clearly stays "on the safe side" especially given the sparsity and (compared to the size of the robot) low precision of the information it has about the environment.

In any case, GEM succeeds in classifying the drivability of terrain similar to the test track in a way that in contrast to PTA allows the SAMSMobil to perform path planning and obstacle avoidance.

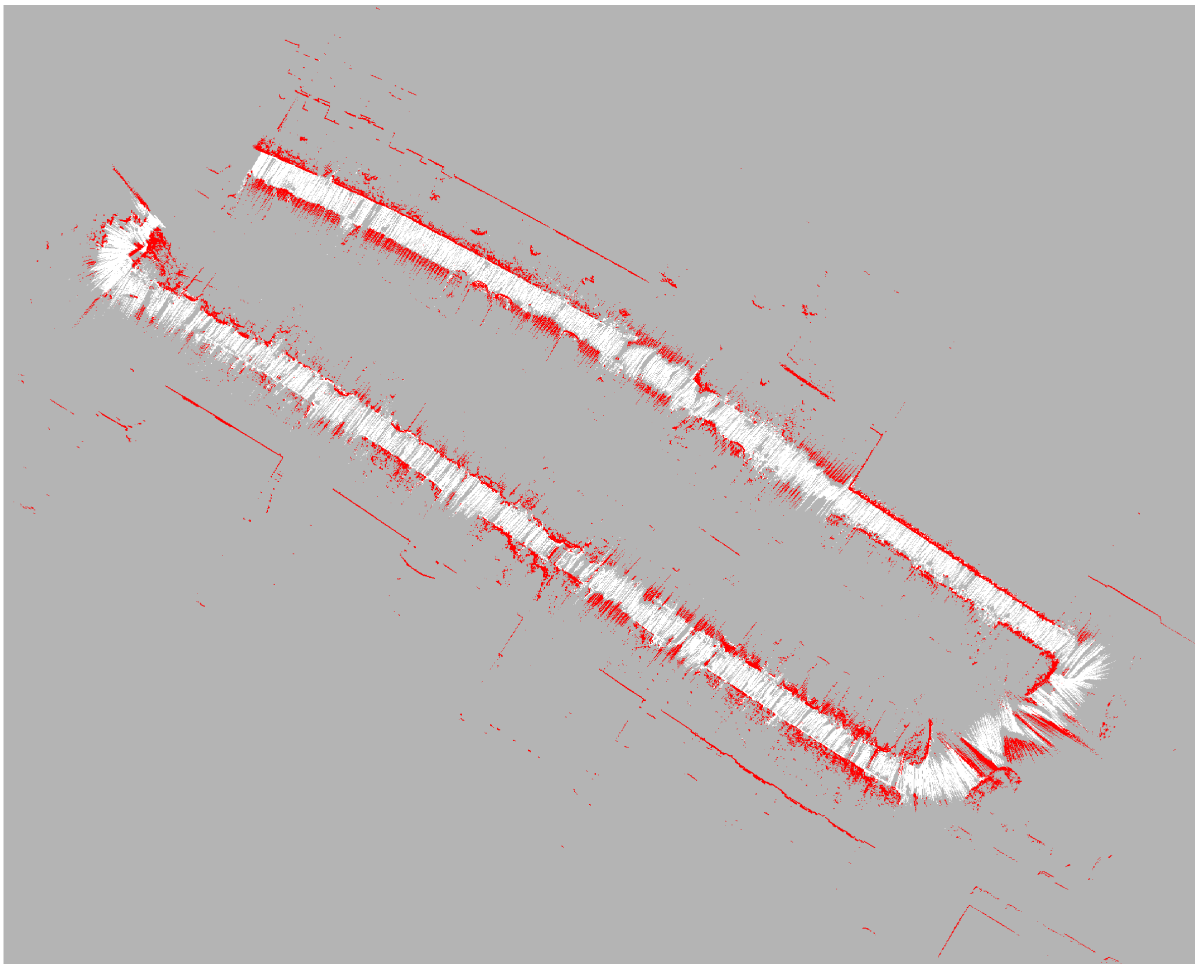
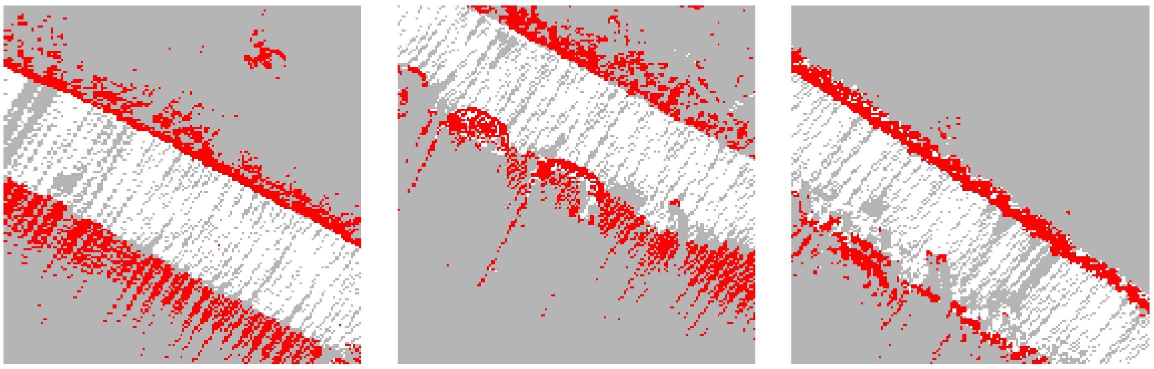


Figure 12.18: Global GEM map.

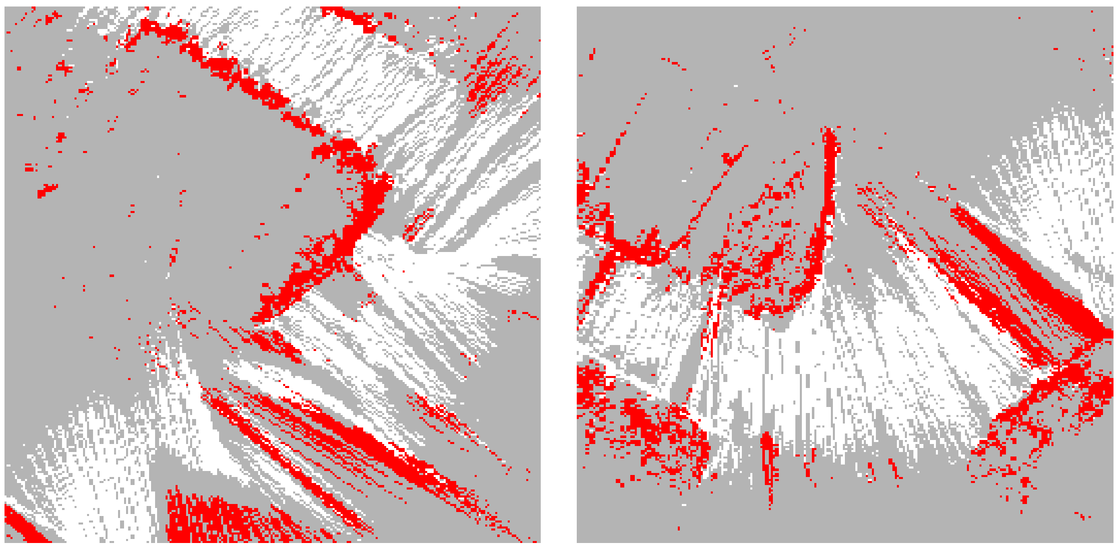


(a) GEM successfully detects the patch of grass (above) and the curb (below the path) as obstacles.

(b) Cars are still detected but do not stick out because the curb is classified as an obstacle, too.

(c) Taller obstacles such as the hedge (above the path) is still seen by GEM.

Figure 12.19: Excerpts from the global GEM map.



(a) The ramp from the sidewalk (top) onto the road (bottom) is detected as drivable. Both sides of the gutter are not.

(b) The ramp from the road (top-right) onto the sidewalk (bottom-left) on the far side of the road. Again the gutter is detected as an obstacle.

Figure 12.20: More excerpts from the global GEM map.

12.2.4 Tests under Autonomous Control

Once the requirement that the pose estimate be accurate up to the width of the drivable path was met autonomous outdoor runs were planned. The weather and various other constraints eventually allowed these to be conducted on two consecutive days in November. The tests were performed with the filter in configuration C but with GPS projected onto the path. The mapping algorithm was GEM.

On the first day the SAMSMobil only made it 25 m far on the test track until an obstacle avoidance response caused a driving command that blew the motor controller fuse because the path controller was configured to aggressively. The fuse unfortunately sits deep in the chassis of and replacing it requires the robot to be disassembled in the lab. This problem was not unknown but in indoor autonomous tests and under manual control had never occured ever since the allowed rotational velocity was limited in software.

On the second test day the SAMSMobil completed two test runs. In the first it got stuck after 25 m because it a bycicle on the sidewalk was classified as an obstacle very late with the initial mapping parameters. After adjusting these the vehicle made it about 43 m far on the test track. The resulting trajectory and the control commands are illustrated in the plot in Figure 12.21 on the facing page. The test run was captured on video – some stills along with a visualization of the internal state belief and obstacle map are depicted in Figure 12.22 on page 140, Figure 12.23 on page 141 and Figure 12.24 on page 142.

The test run ended when the mapper classified too much of a driveway as non-drivable for the robot to fit between this and the path boundary on the right. Recall that the planner uses circular obstacle extrusion based on the distance it takes for the robot to stop safely. The planner first attempted to avoid the obstacle and the resulting turn again blew the motor controller fuse although the turn was fairly slow and smooth. Due to time constraints it was not possible to investigate this phenomenon further. The small, but sustained oscillations caused by the path controller may contribute to it.

In any case the 43 m test run showed the following:

- All components work under “race conditions” and control the vehicle in real time.
- The state estimation performance is sufficient to navigate based on it.
- The projected GPS measurement model certainly helped with this. In particular, the corner of the first tall building by the test track was passed which had previously been problematic due to multipath errors.
- The travelled distance is sufficiently long to say that it can only be traversed if the complete control loop works. It must be seen in relation to the size of the vehicle which is about ten times smaller than a regular passenger car.
- The robot sees all relevant obstacles, curbs which pose a major hazard to the small vehicle in particular.
- The robot successfully centers itself on the path and avoids obstacles, i.e., the planner works and particularly compensates for offsets in the state estimate.

It may well be possible to further improve the autonomous performance through more parameter tuning which was unfortunately not possible due to time constraints. At the end of the day, however, one has to acknowledge that the robot is limited by the comparably low precision of its sensors relative to the small size of the vehicle.

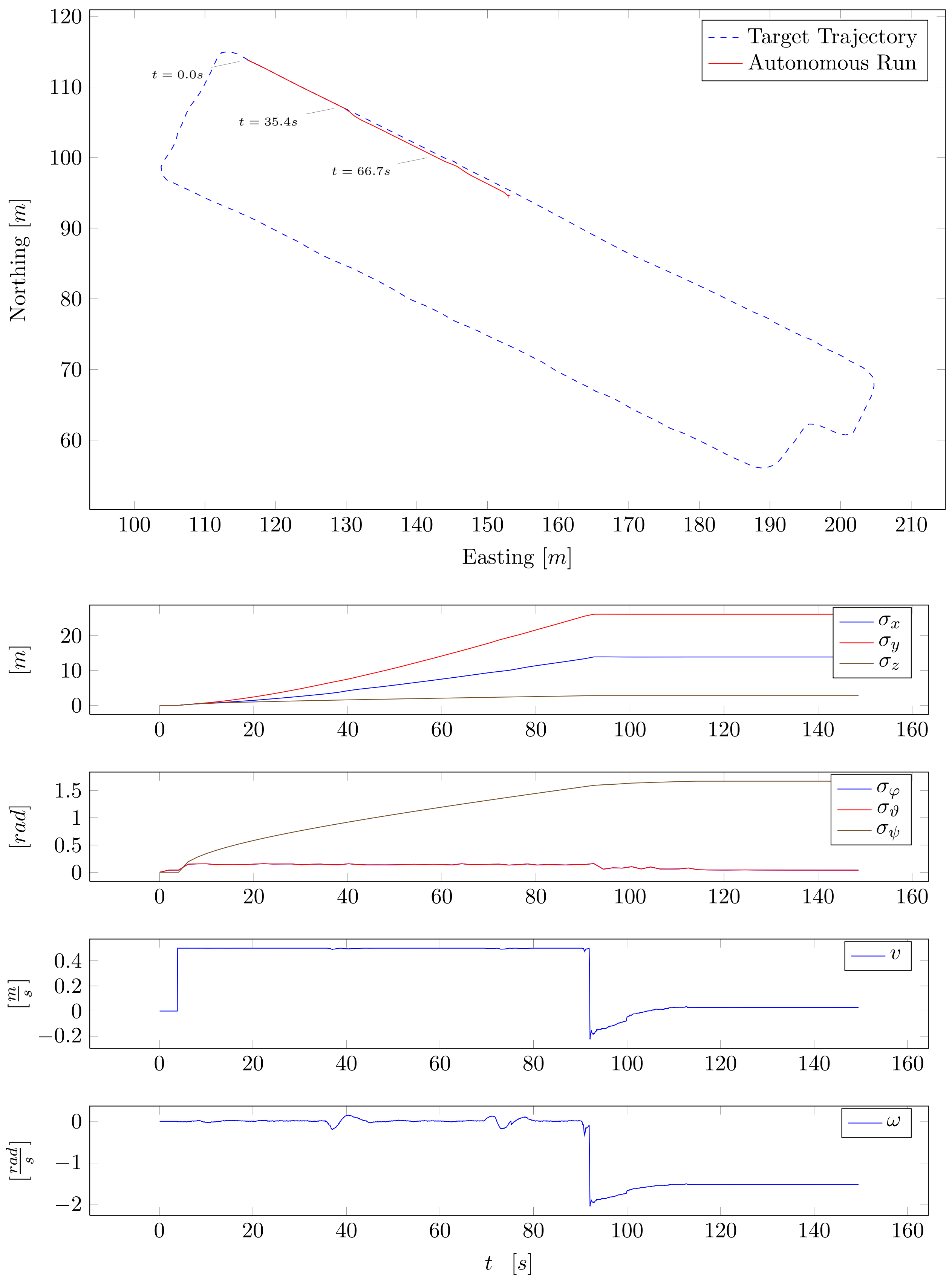
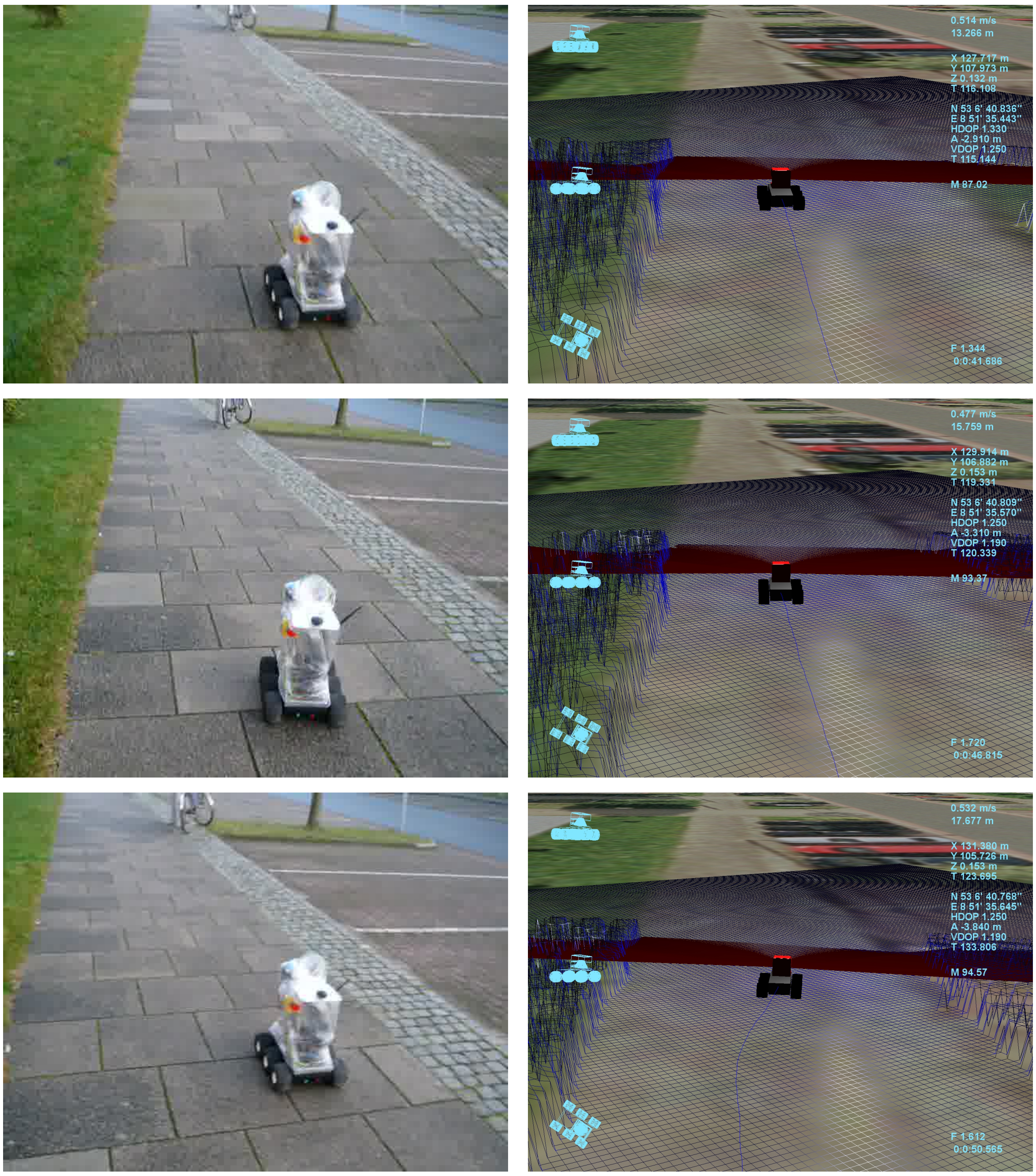


Figure 12.21: UKF position estimates and control commands from the 43m autonomous run.

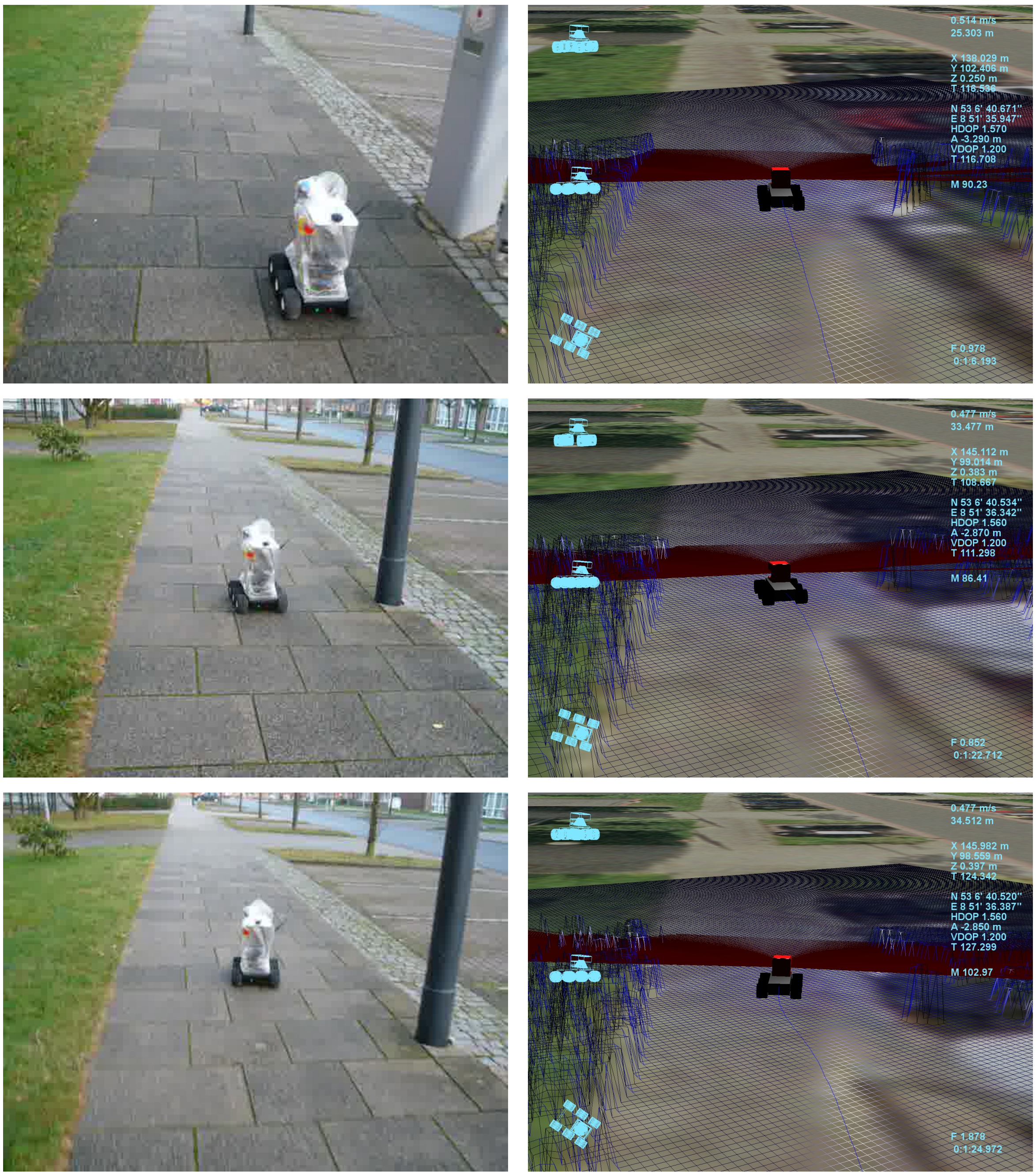


```
0.514 m/s
13.266 m

X 127.717 m
Y 107.973 m
Z 0.132 m
T 116.108

N 53 6' 40.836''
E 8 51' 35.443''
HDOP 1.330
A -2.910 m
VDOP 1.250
T 115.144

M 87.02

F 1.344
 0:0:41.686
```

```
0.477 m/s
15.759 m

X 129.914 m
Y 106.882 m
Z 0.153 m
T 119.331

N 53 6' 40.809''
E 8 51' 35.570''
HDOP 1.250
A -3.310 m
VDOP 1.190
T 120.339

M 93.37

F 1.720
 0:0:46.815
```

```
0.532 m/s
17.677 m

X 131.380 m
Y 105.726 m
Z 0.153 m
T 123.695

N 53 6' 40.768''
E 8 51' 35.645''
HDOP 1.250
A -3.840 m
VDOP 1.190
T 133.806

M 94.57

F 1.612
 0:0:50.565
```

Figure 12.22: The SAMSMobil under autonomous control by the complete software stack in the 43 m test run. The real robot is shown on the left, its internal state belief and the obstacle map on the right, first shortly after the start of the test track (top), then while initiating a path centering response (middle). and after completing it (bottom).



```
0.514 m/s
25.303 m

X 138.029 m
Y 102.406 m
Z 0.250 m
T 116.536

N 53 6' 40.671''
E 8 51' 35.947''
HDOP 1.570
A -3.290 m
VDOP 1.200
T 116.708

M 90.23

F 0.978
 0:1:6.193
```

```
0.477 m/s
33.477 m

X 145.112 m
Y 99.014 m
Z 0.383 m
T 108.667

N 53 6' 40.534''
E 8 51' 36.342''
HDOP 1.560
A -2.870 m
VDOP 1.200
T 111.298

M 86.41

F 0.852
 0:1:22.712
```

```
0.477 m/s
34.512 m

X 145.982 m
Y 98.559 m
Z 0.397 m
T 124.342

N 53 6' 40.520''
E 8 51' 36.387''
HDOP 1.560
A -2.850 m
VDOP 1.200
T 127.299

M 102.97

F 1.878
 0:1:24.972
```

Figure 12.23: The SAMSMobil under autonomous control by the complete software stack in the 43 m test run: The bycicle and parking meter which had previously been a problem are passed successfully (top) and the lamp post is also avoided (middle and bottom).

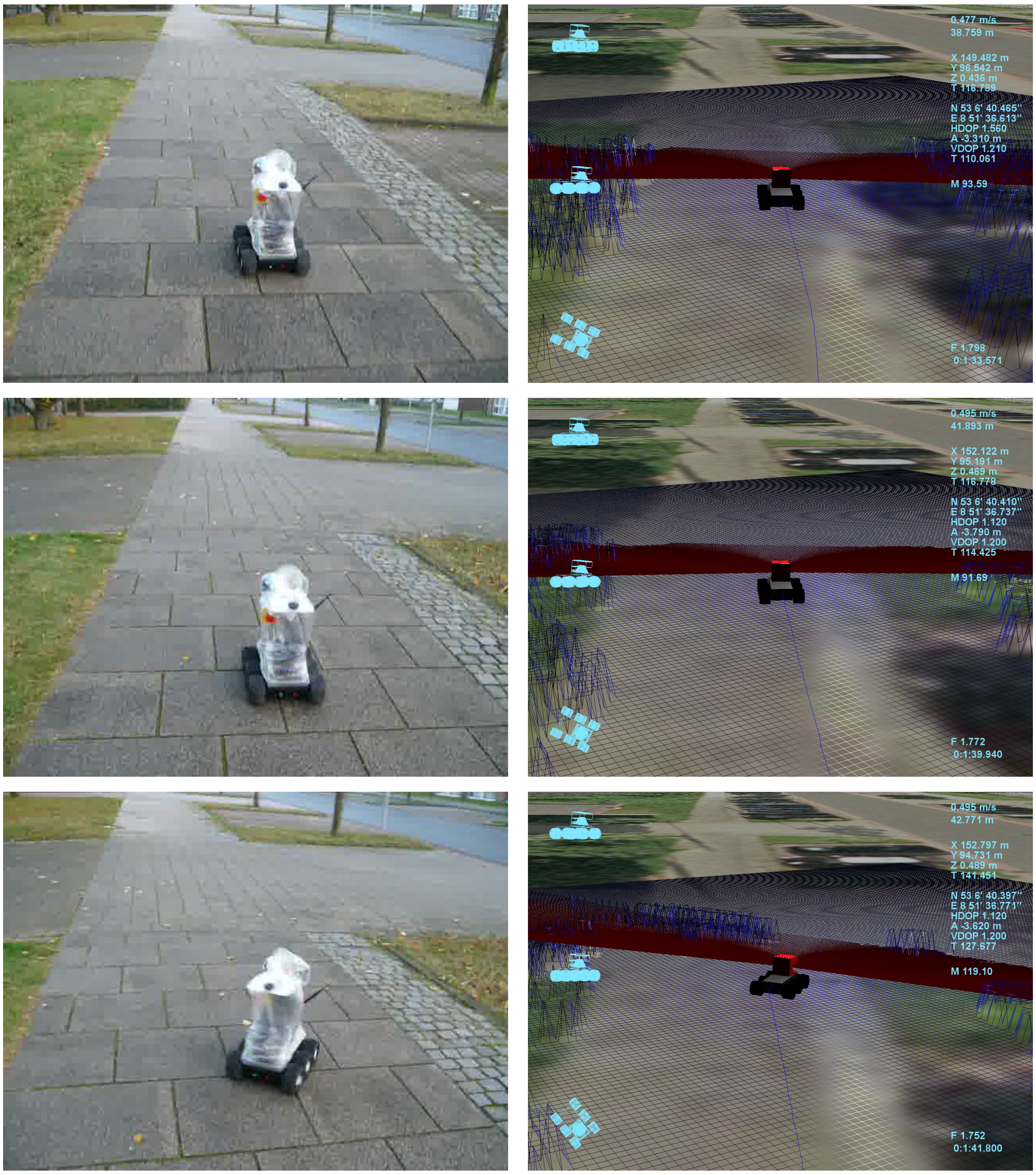


Figure 12.24: The SAMSMobil under autonomous control by the complete software stack in the 43 m test run: The robot drives nicely centered on the sidewalk (top) but classifies the driveway as an obstacle (middle), veers to the side and blows the motor controller fuse (bottom).

Chapter 13

Conclusions and Future Work

A complete, autonomous robotic vehicle for outdoor navigation was developed. The performance of the state estimation/localization and mapping components were evaluated on log data. Planning and control was first tested in simulation. Later the complete system was evaluated in actual outdoor tests during which the robot demonstrated autonomous navigation abilities.

State estimation given the comparably low precision of the available sensors (particularly, the consumer grade GPS receiver) turned out to be the most challenging and was thus the primary focus of this thesis. The manifold based unscented Kalman filter (UKF) algorithm was extended to handle manifold measurements and not just a manifold state representation. It was shown that a mathematically sound filter algorithm and a numerically robust implementation are essential. The flexibility and performance of the generic manifold-UKF implementation that was developed enabled the seamless integration of a variety of measurement models. While pure state estimation (more specifically pose tracking) based on odometry, gyroscope, accelerometer and (consumer grade) GPS data was not sufficient to get a pose estimate with a precision in the order of the path width, localization relative to a path boundary map based on virtual sensors detecting path boundary in the obstacle map achieves this goal but relies on an accurate map. The projection of GPS positioning solutions onto the goal path achieved a state estimation performance that enabled autonomous driving.

Mapping was difficult for similar reasons. First of all the laser range finder data needs to be transformed into world coordinates based on the current pose estimate. Inaccuracies in this are amplified by the distance to objects when it comes to mapping. The initially evaluated mapping algorithm Probabilistic Terrain Analysis (PTA) that was successful on Stanley, the winning robot of the 2005 Grand Challenge, manages to discard the resulting erroneous vertical distances but the remaining information is insufficient to tell smaller obstacles such as patches of grass from drivable terrain. This, in combination with the fact that PTA cannot detect negative obstacles, prompted the development of a new mapping algorithm, Ground Expectation Mapping (GEM), as part of this thesis. GEM classifies range scans directly based on matches or mismatches with an expected range scan given the pose of the robot and the ground plane and successfully detects virtually all relevant obstacles but places stronger requirements on the smoothness of the environment. Thus it enabled the SAMSMobil to safely navigate the outdoor test track but did not allow the full course to be traversed due to a false positive obstacle classification.

Although the planner and path controller have not been a focus of this thesis they generally worked successfully. Given the fact that hardly any parameter tuning could be performed due to time constraints, and considering the relatively low precision of the sensor suite compared to the small size of the robot, it got remarkably far in the autonomous outdoor runs.

With regard to future work, there are a variety of areas open for investigation. The SAMSMobil in its current configuration is a first prototype and experience with it could be fielded into a new hardware revision. In particular, the motor controller fuse problem should be solved and a redesign of the way the

IMU is mounted could introduce damping to decouple the IMU from the frame which might improve its accuracy.

On the algorithmic end it would be interesting to find out how availability of calibrated magnetometer data influences the overall state estimation performance. And it should be possible to improve the mapping performance with GEM given more parameter tuning.

More generally, the use of GPS as the dominant source of positioning information should be reconsidered primarily because it is virtually impossible to tell whether a measurement is "correct" or the result of (multipath) errors unless the receiver does this internally (some survey grade receivers guarantee a certain positioning quality and do not output any data at all in case of errors in the GPS signal). Also, considering the delayed GPS modernization, the resulting satellite outages predicted for as soon as 2010 (United States Government Accountability Office, 2009) and the time it will take for Galileo to become fully operational, availability of satellite navigation systems should not be taken for granted in the first place.

On the SAMSMobil the single ROD4plus laser range finder had to be used since the hardware was shared with the SAMS project. The chosen forward tilted setup is the only one that allows the detection of all relevant obstacles. This, however, rules out the entire SLAM family of algorithms. While traditional SLAM goals such as loop closing would not be of much benefit for autonomous navigation scan matching could help stabilize the state estimate. Replacing the ROD4plus with two smaller, more lightweight Hokuyo UTM-30LX units, where, e.g., one of them is tilted downward but the other scans a plane parallel to the ground or is pointed slightly upward, would enable this type of research. The UTM-30LX can also be triggered externally which solves the data synchronization problem between IMU and laser scan data which has caused significant trouble during the course of this thesis.

Towards the broader goal of driverless cars, the next big step will clearly need to address safety. The sensor equipment and algorithms need to be able to handle any terrain safely in any situation, e.g., the systematic failure to detect certain types of obstacles must be ruled out completely. Formal modelling of the application domain along with formal verification of the software implementation is instrumental in achieving this as the SAMS project has demonstrated.

In parallel, a wide variety of questions ranging from liability concerns to the way we as humans react to driverless cars will need to be answered.

Bibliography

Abbeel, P. (2008). *Apprenticeship Learning and Reinforcement Learning with Application to Robotic Control.* PhD thesis, Department of Computer Science. Stanford University. http://www.cs.stanford.edu/%7Epabbeel/thesis/thesis.pdf.

Agilent Technologies (2002). *Quick Assembly: Two and Three Channel Optical Encoders HEDM-550x/560x HEDS-550x/554x HEDS-560x/564x.* Datasheet.

Alexandrescu, A. (2001). *Modern C++ Design, Generic Programming and Design Patterns Applied.* Boston, MA: Addison-Wesley.

Analog Devices (2004). *ADXRS 150 Datasheet.* Norwood, MA. http://www.analog.com/UploadedFiles/Data_Sheets/ADXRS150.pdf.

Analog Devices (2006). *ADXL203 Datasheet.* Norwood, MA. http://www.analog.com/UploadedFiles/Data_Sheets/ADXL103_203.pdf.

Anonymous (2009). 2002 überlingen mid-air collision. Wikipedia article. http://en.wikipedia.org/wiki/2002_œberlingen_mid-air_collision. Last checked: 2009-12-12.

Antone, M. & Friedman, Y. (2007). Fully automated laser range calibration. In *Proceedings of the British Machine Vision Conference 2007 (BMVC07).* http://www.dcs.warwick.ac.uk/bmvc2007/proceedings/CD-ROM/papers/paper-29.pdf.

Aubert, D., Kluge, K., & Thorpe, C. (1990). Autonomous navigation of structured city roads. In *Proceedings of SPIE Mobile Robots V.*

Bennewitz, M., Stachniss, C., Behnke, S., & Burgard, W. (2009). Utilizing reflection properties of surfaces to improve mobile robot localization. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)* Kobe, Japan. http://europa.informatik.uni-freiburg.de/files/bennewitz09icra.pdf.

Birbach, O. (2008). Accuracy analysis of camera-inertial sensor-based ball trajectory prediction. Diploma thesis. University of Bremen. Faculty of Mathematics and Computer Science.

Bräunl, T. (2003). *Embedded Robotics.* Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Bresenham, J. E. (1965). Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1), 25–30. http://www.research.ibm.com/journal/sj/041/ibmsjIVRIC.pdf.

Burns, M. (2009). A look at how the Volkswagen Automotive Innovation Laboratory (VAIL) is changing the world. CrunchGear.com online article. http://www.crunchgear.com/2009/10/24/a-look-at-how-the-volkswagen-automotive-innovation-laboratory-vail-is-changing-the-world/. Last checked: 2009-10-27.

Collett, T. H., MacDonald, B. A., & Gerkey, B. P. (2005). Player 2.0: Toward a practical robot programming framework. In *Proceedings of the Australasian Conference on Robotics and Automation (ACRA 2005).*

Cremean, L., Foote, T., Gillula, J., Hines, G., Kogan, D., Kriechbaum, K., Lamb, J., Leibs, J., Lindzey, L., Rasmussen, C., Stewart, A., Burdick, J., & Murray, R. (2007). *The 2005 DARPA Grand Challenge*, chapter Alice: An Information-Rich Autonomous Vehicle for High-Speed Desert Navigation, (pp. 437–482). Springer: Berlin.

Dahlkamp, H., Kaehler, A., Stavens, D., Thrun, S., & Bradski, G. (2006). Self-supervised monocular road detection in desert terrain. In G. Sukhatme, S. Schaal, W. Burgard, & D. Fox (Eds.), *Proceedings of the Robotics Science and Systems Conference* Philadelphia, PA. http://robots.stanford.edu/papers/thrun.mapping-Stanley.pdf.

Daily, R., Travis, W., Bevly, D., Knoedler, K., Behringer, R., Hemetsberger, H., Kogler, J., Kubinger, W., & Alefs, B. (2007). *The 2005 DARPA Grand Challenge*, chapter SciAutonics-Auburn Engineeringâ??s Low Cost High Speed ATV for the 2005 DARPA Grand Challenge, (pp. 281–309). Springer: Berlin.

Dedu, E. (2001). Bresenham-based supercover line algorithm. http://lifc.univ-fcomte.fr/~dedu/projects/bresenham/index.html. Last checked: 2009-11-29.

Dickmanns, E. D. (1992). *Active Vision*, chapter Expectation-Based Dynamic Scene Understanding. MIT Press: Cambridge, MA.

Dickmanns, E. D. (1994). The 4D-approach to dynamic machine vision. In *Proceedings of the 33rd Conference on Decision and Control. Lake Buena Vista, FL.*

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1, 269–271. http://www-m3.ma.tum.de/twiki/pub/MN0506/WebHome/dijkstra.pdf.

Elfes, A. (1987). Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation*, 3(3), 249–265.

ESA (2005a). EGNOS fact sheet 12: The EGNOS signal explained. http://www.egnos-pro.esa.int/Publications/2005%20Updated%20Fact%20Sheets/fact_sheet_12.pdf. Last checked: 2009-11-05.

ESA (2005b). EGNOS fact sheet 2: The ionosphere explained. http://www.egnos-pro.esa.int/Publications/2005%20Updated%20Fact%20Sheets/fact_sheet_2.pdf. Last checked: 2009-11-05.

ESA (2009). EGNOS 'Open Service' available: A new era for European navigation begins today. Press release. http://www.esa.int/esaNA/SEM2HGF280G_egnos_0.html. Last checked: 2009-11-05.

Fox, D., Burgard, W., & Thrun, S. (1997). The dynamic window approach to collision avoidance. *IEEE Robotics and Automation*, 4(1). http://robots.stanford.edu/papers/fox.dwa_rob_magazine.ps.gz.

Frese, U. (2009). Discontinuities in laser range scan data. *Unpublished.*

Frese, U., Hausmann, D., Lüth, C., Täubig, H., & Walter, D. (2008a). The importance of being formal. In H. Hungar (Ed.), *International Workshop on the Certification of Safety-Critical Software Controlled Systems SafeCert'08*, Electronic Notes in Theoretical Computer Science: Elsevier Science.

Frese, U., Laue, T., & Röfer, T. (2008b). Pilot – Bahnregler, Hindernisvermeidung. Lecture slides of the course "Autonome Navigation im Außenraum". University of Bremen. Winter term 2007/2008.

Frese, U. & Schröder, L. (2008). *Theorie der Sensorfusion.* Lecture notes. https://svn-agbkb.informatik.uni-bremen.de/ufrese/teaching/tds/skript/pdf/tds08skript.pdf.

Gersdorf, B. (2009). Laser range finder calibration for the Bremen Autonomous Wheelchair. *Unpublished*.

Goto, Y. & Stentz, A. T. (1987). Mobile robot navigation: The cmu system. *IEEE Expert*, 2(1), 44–55.

Grewal, M. S., Weill, L. R., & Andrews, A. P. (2007). *Global Positioning Systems, Inertial Navigation, and Integration*. John Wiley, 2nd edition.

Grisetti, G., Stachniss, C., & Burgard, W. (2005). Improving grid-based SLAM with Rao-Blackwellized particle filters by adaptive proposals and selective resampling. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)* (pp. 2443–2448). Barcelona, Spain. http://www.informatik.uni-freiburg.de/~stachnis/pdf/grisetti05icra.pdf.

Grisetti, G., Stachniss, C., & Burgard, W. (2006). Improved techniques for grid mapping with Rao-Blackwellized particle filters. *IEEE Transactions on Robotics*. http://www.informatik.uni-freiburg.de/~stachnis/pdf/grisetti06tro.pdf.

Hertzberg, C. (2008). A framework for sparse, non-linear least squares problems on manifolds. Diploma thesis. University of Bremen. Faculty of Mathematics and Computer Science.

Hertzberg, C., Frese, U., Schröder, L., & Wagner, R. (2009). Encapsulating manifolds for sensor fusion. *Unpublished*. Planned for publication in 2010. Title subject to change.

Hill, Jr., F. S. & Kelley, S. M. (2007). *Computer Graphics: Using OpenGL*. Upper Saddle River, NJ: Pearson Prentice Hall, 3rd edition.

Huang, A., Olson, E., & Moore, D. (2009). *Lightweight Communications and Mashalling for Low-Latency Interprocess Communication*. Technical Report MIT-CSAIL-TR-2009-041, MIT Computer Science and Artificial Intelligence Laboratory.

Humbert, M., Gey, N., Muller, J., & Esling, C. (1996). Determination of a mean orientation from a cloud of orientations. application to electron back-scattering pattern measurements. *Journal of Applied Crystallography*, 29(6), 662–666.

Julier, S. J. (2002). The scaled unscented transformation. In *Proceedings of the 2002 American Control Conference*, number 6 (pp. 4555–4559). http://www.cs.unc.edu/~welch/kalman/media/pdf/ACC02-IEEE1357.PDF.

Julier, S. J. & Uhlmann, J. K. (1996). *A General Method for Approximating Nonlinear Transformations of Probability Distributions*. Technical report.

Julier, S. J. & Uhlmann, J. K. (1997). A new extension of the Kalman filter to nonlinear systems. In *Int. Symp. Aerospace/Defense Sensing, Simul. and Controls, Orlando, FL* (pp. 182–193).

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D), 35–45. http://www.cs.unc.edu/~welch/kalman/media/pdf/Kalman1960.pdf.

Kelly, A. & Stentz, A. (1997). An analysis of requirements for rough terrain autonomous mobility. *Autonomous Robots*, 4(1). http://www.frc.ri.cmu.edu/~alonzo/pubs/papers/jar96_theory.pdf.

King, D. (2005). A resonant MEMS magnetometer. In *The IEE Seminar and Exhibition on MEMS Sensor Technologies*.

Ko, N. Y., Simmons, R., Reid, K., & Simmons, G. (1998). The lane-curvature method for local obstacle avoidance. In *Proceedings of the 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3 Victoria, BC, Canada. http://www.cs.cmu.edu/afs/cs.cmu.edu/user/reids/www/papers/lcm.ps.gz.

Kohlhoff, C. (2008). Boost.Asio. Documentation from Boost 1.37. http://www.boost.org/doc/libs/1_37_0/doc/html/boost_asio.html. Last checked: 2009-11-27.

Kouba, J. (2003). A guide to using International GPS Service (IGS) products. http://igscb.jpl.nasa.gov/igscb/resource/pubs/GuidetoUsingIGSProducts.pdf. Last checked: 2009-09-02.

Kraft, E. (2003). A quaternion-based unscented Kalman filter for orientation tracking. In *Proceedings of the Sixth International Conference of Information Fusion*, volume 1 (pp. 47–54). http://hep1.physik.uni-bonn.de/fileadmin/Publications/Mqube/Kraft_FusionPaperWeb.pdf.

Krieger Lassen, N. C., Juul Jensen, D., & Conradsen, K. (1994). On the statistical analysis of orientation data. *Acta Crystallographica Section A*, 50(6), 741–748.

Kümmerle, R., R.Triebel, Pfaff, P., & Burgard, W. (2008). Monte carlo localization in outdoor terrains using multilevel surface maps. *Journal of Field Robotics*, 25(6–7), 346–359.

Kurlbaum, J. (2007). Verfolgung von Ballflugbahnen mit einem frei beweglichen Kamera-Inertialsensor. Diploma thesis. University of Bremen. Faculty of Mathematics and Computer Science.

Lamon, P. & Siegwart, R. (2003). 3D-odometry for rough terrain – towards real 3D navigation. In *Proceedings of the 2003 IEEE International Conference on Robotics and Automation*. http://asl.epfl.ch/aslInternalWeb/ASL/publications/uploadedFiles/Odometry3DIcra2003_.pdf.

Lamon, P., Stachniss, C., Triebel, R., Pfaff, P., Plagemann, C., Grisetti, G., Kolski, S., Burgard, W., & Siegwart, R. (2006). In *Proceedings of the Workshop on Safe Navigation in Open and Dynamic Environments at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)* Beijing, China. http://www.informatik.uni-freiburg.de/~stachnis/pdf/lamon06iros.pdf.

Lee, J. M. (2003). *Introduction to Smooth Manifolds*. New York: Springer.

Leedy, B., Putney, J., Bauman, C., Cacciola, S., Michael Webster, J., & Reinholtz, C. (2007). *The 2005 DARPA Grand Challenge*, chapter Virginia Tech's Twin Contenders: A Comparative Study of Reactive and Deliberative Navigation, (pp. 155–182). Springer: Berlin.

Leuze electronic (2008). *rotoScan ROD4plus / ROD4-08plus*. Technical Manual.

Lingemann, K., Nüchter, A., Hertzberg, J., & Surmann, H. (2005). About the control of high speed mobile indoor robots. In *Proceedings of the Second European Conference on Mobile Robotics (ECMR '05)* Ancona, Italy. http://kos.informatik.uni-osnabrueck.de/download/ecmr2005.pdf.

Madsen, K., Nielsen, H. B., & Tingleff, O. (2004). *Methods for Non-Linear Least Squares Problems (2nd ed.)*. Informatics and Mathematical Modelling, Technical University of Denmark, DTU. http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/3215/pdf/imm3215.pdf.

Mahalanobis, P. C. (1936). On the generalised distance in statistics. *Proceedings of the National Institute of Sciences of India*, 2(1), 49–55. http://ir.isical.ac.in/dspace/handle/1/1268.

Mandel, C., Huebner, K., & Vierhuff, T. (2005). Towards an autonomous wheelchair: Cognitive aspects in service robotics. In *Proceedings of Towards Autonomous Robotic Systems (TAROS 2005)* (pp. 165–172). http://www.iis.ee.ic.ac.uk/~taros05/papers/mandel.pdf.

Markley, F. L., Cheng, Y., Crassidis, J. L., & Oshman, Y. (2007). *Quaternion Averaging*. Technical report, NASA Goddard Space Flight Center. http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20070017872_2007014421.pdf.

Mason, R., Radford, J., Kumar, D., Walters, R., Fulkerson, B., Jones, E., Caldwell, D., Meltzer, J., Alon, Y., Shashua, A., Hattori, H., Frazzoli, E., & Soatto, S. (2007). *The 2005 DARPA Grand Challenge*, chapter The Golem Group / UCLA Autonomous Ground Vehicle in the DARPA Grand Challenge, (pp. 205–243). Springer: Berlin.

McCallum, E. (2005). Processing XML with Xerces and SAX. O'Reilly ONLamp.com tutorial article. http://onlamp.com/pub/a/onlamp/2005/11/10/xerces_sax.html. Last checked: 2009-07-02.

McLean, S., Macmillan, S., Maus, S., Lesur, V., Thomson, A., & Dater, D. (2004). *The US/UK World Magnetic Model for 2005–2010*. Technical Report NESDIS/NGDC-1, NOAA.

Montemerlo, M., Becker, J., Bhat, S., Dahlkamp, H., Dolgov, D., Ettinger, S., Haehnel, D., Hilden, T., Hoffmann, G., Huhnke, B., Johnston, D., Klumpp, S., Langer, D., Levandowski, A., Levinson, J., Marcil, J., Orenstein, D., Paefgen, J., Penny, I., Petrovskaya, A., Pflueger, M., Stanek, G., Stavens, D., Vogt, A., & Thrun, S. (2008). Junior: The Stanford entry in the Urban Challenge. *Journal of Field Robotics*, 25(9), 569–597. http://robots.stanford.edu/papers/junior08.pdf.

Montemerlo, M., Roy, N., & Thrun, S. (2003a). Perspectives on standardization in mobile robot programming: The Carnegie Mellon Navigation (CARMEN) toolkit. In *Proceedings of the Conference on Intelligent Robots and Systems (IROS)*.

Montemerlo, M., Thrun, S., Koller, D., & Wegbreit, B. (2002). FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the AAAI National Conference on Artificial Intelligence* Edmonton, Canada: AAAI.

Montemerlo, M., Thrun, S., Koller, D., & Wegbreit, B. (2003b). FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)* Acapulco, Mexico: IJCAI.

Moravec, H. P. (1988). Sensor fusion in certainty grids for mobile robots. *AI Magazine*, 9(2), 61–74. http://www.aaai.org/ojs/index.php/aimagazine/article/view/676/594.

Morawiec, A. (1998). A note on mean orientation. *Journal of Applied Crystallography*, 31(5), 818–819.

National Imagery and Mapping Agency (2000). *Department of Defense World Geodetic System 1984 – Its Definition and Relationships With Local Geodetic Systems*. Technical report. http://earth-info.nga.mil/GandG/publications/tr8350.2/wgs84fin.pdf.

Newman, P. (2009). Introduction to programming with MOOS. Programming Manual. http://www.robots.ox.ac.uk/~pnewman/MOOSDocumentation/ProgrammingWithMOOS/latex/ProgrammingWithMOOS.pdf. Last checked: 2009-11-30.

Nüchter, A., Surmann, H., Lingemann, K., & Hertzberg, J. (2004a). 6D SLAM - preliminary report on closing the loop in six dimensions. http://kos.informatik.uni-osnabrueck.de/download/iav2004.pdf.

Nüchter, A., Surmann, H., Lingemann, K., Hertzberg, J., & Thrun, S. (2004b). 6D SLAM with application in autonomous mine mapping. In *Proceedings IEEE 2004 International Conference Robotics and Automation (ICRA '04)* (pp. 1998–2003). New Orleans, USA. http://kos.informatik.uni-osnabrueck.de/download/icra2004.pdf.

OpenStreetMap Project (2009). Slippy map tilenames. Wiki page. http://wiki.openstreetmap.org/wiki/Slippy_map_tilenames. Last checked: 20091012.

Paparazzi Project (2009). GPS. Wiki page. http://paparazzi.enac.fr/wiki/GPS. Last checked: 2009-11-05.

Plasswich, S. (2008). Kollisionsvermeidung und Fahrzeugführung per Laserscanner auf einem schnellen Modellauto. Master's thesis, University of Bremen. Faculty of Mathematics and Computer Science.

Quigley, M., Abbeel, P., Lorenzo, D. S. D., Gu, Y., Bolouki, S., Akos, D., & Ng., A. Y. (2007). Portable gnss baseband logging. In *Institute of Navigation (ION) GNSS Conference*. http://www-cs.stanford.edu/~pabbeel/pubs/QADGBAN_pgnssbl_ion2007.pdf.

Quine, B., Uhlmann, J., & Durrant-Whyte, H. (1995). Implicit jacobians for linearised state estimation in nonlinear systems. In *Proc. of the American Control Conference, Seattle, WA* (pp. 1645–1646).

Raymond, E. S. (2009). NMEA revealed. v2.1, Sep 2009. http://gpsd.berlios.de/NMEA.txt. Last checked: 2009-11-03.

RescueRobotics Project (2007). Abschlussberich des Projekts “Rescue Robotics”. Student Project Final Report. University of Bremen. Faculty of Mathematics and Computer Science.

Röfer, T. (2009). Using gps data in combination with openstreetmap road maps. *Unpublished*.

R.Triebel, Pfaff, P., & Burgard, W. (2006). Multi-level surface maps for outdoor terrain mapping and loop closing. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2006*. http://www.informatik.uni-freiburg.de/~triebel/publications/triebel06multi.pdf.

Russell, S. & Norvig, P. (2002). *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition.

Shoemake, K. (1985). Animating rotation with quaternion curves. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques* (pp. 245–254). New York, NY, USA: ACM.

Tanenbaum, A. S. (2003). *Computer Networks, Fourth Edition*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Taylor, I. J. & Harrison, A. (2009). *From P2P and Grids to Services on the Web: Evolving Distributed Communities*. London: Springer, 2nd edition.

Thrun, S., Burgard, W., & Fox, D. (2005a). *Probabilistic Robotics*. Cambridge, MA: MIT Press.

Thrun, S., Burgard, W., & Fox, D. (2005b). Probabilistic robotics – mapping with known poses. Companion slides to the Probabilistic Robotics book by the same authors. http://robots.stanford.edu/probabilistic-robotics/ppt/mapping-occupancy.ppt. Last checked: 20091110.

Thrun, S., Diel, M., & Hähnel, D. (2003). Scan alignment and 3D surface modeling with a helicopter platform. In *Proceedings of the International Conference on Field and Service Robotics* Lake Yamanaka, Japan. http://robots.stanford.edu/papers/thrun.heli-mapping03.pdf.

Thrun, S., Montemerlo, M., & Aron, A. (2006a). Probabilistic terrain analysis for high-speed desert driving. In G. Sukhatme, S. Schaal, W. Burgard, & D. Fox (Eds.), *Proceedings of the Robotics Science and Systems Conference* Philadelphia, PA. http://robots.stanford.edu/papers/thrun.mapping-Stanley.pdf.

Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C., Palatucci, M., Pratt, V., Stang, P., Strohband, S., Dupont, C., Jendrossek, L.-E., Koelen, C., Markey, C., Rummel, C., van Niekerk, J., Jensen, E., Alessandrini, P., Bradski, G., Davies, B., Ettinger, S., Kaehler, A., Nefian, A., & Mahoney, P. (2006b). Stanley: The robot that won the DARPA Grand Challenge. *Journal of Field Robotics*, 23(9). http://robots.stanford.edu/papers/thrun.stanley05.pdf.

Trepagnier, P., Nagel, J., Kinney, P., Koutsougeras, C., & Dooner, M. (2007). *The 2005 DARPA Grand Challenge*, chapter KAT-5: Robust Systems for Autonomous Vehicle Navigation in Challenging and Unknown Terrain, (pp. 103–128). Springer: Berlin.

United States Government Accountability Office (2009). Global positioning system – significant challenges in sustaining and upgrading widely used capabilities. Testimony Before the Subcommittee on National Security and Foreign Affairs, Committee on Oversight and Government Reform, House of Representatives. GAO-09-670T. http://www.gao.gov/new.items/d09670t.pdf. Retrieved: 2009-12-10.

van der Merwe, R. (2004). *Sigma-Point Kalman Filters for Probabilistic Inference in Dynamic State-Space Models*. PhD thesis, OGI School of Science & Engineering, Oregon Health & Science University, Portland, OR, USA. http://www.cse.ogi.edu/~rudmerwe/pubs/pdf/rvdmerwe_phd_thesis.pdf.

van der Merwe, R. & Wan, E. (2001a). Efficient derivative-free Kalman filters for online learning. In *Proceedings of the European Symposium on Artificial Neural Networks (ESANN)* Bruges, Belgium. http://cslu.cse.ogi.edu/publications/ps/merwe01.ps.gz.

van der Merwe, R. & Wan, E. (2001b). The square-root unscented Kalman filter for state and parameter estimation. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)* Salt Lake City, Utah: IEEE. http://cslu.cse.ogi.edu/publications/ps/merwe01a.ps.gz.

van der Merwe, R. & Wan, E. (2003). Sigma-point Kalman filters for probabilistic inference in dynamic state-space models. In *Proceedings of the Workshop on Advances in Machine Learning* Montreal, Canada. http://cslu.cse.ogi.edu/publications/ps/merwe03a.ps.gz.

van der Merwe, R., Wan, E., & Julier, S. (2004). Sigma-point Kalman filters for nonlinear estimation and sensor-fusion: Applications to integrated navigation. In *AIAA Guidance, Navigation, and Control Conference and Exhibit* Providence, Rhode Island. http://www.cse.ogi.edu/~rudmerwe/pubs/pdf/vanderMerwe_GNC2004.pdf.

Vincenty, T. (1975). Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations. *Survey Review*, XXII(176), 88–93. http://www.ngs.noaa.gov/PUBS_LIB/inverse.pdf.

Wurm, K. M., Stachniss, C., Grisetti, G., & Burgard, W. (2007). Improved simultaneous localization and mapping using a dual representation of the environment. In *Proc. of the European Conference on Mobile Robots (ECMR)* Freiburg, Germany. http://www.informatik.uni-freiburg.de/~wurm/wurm07ecmr.pdf.

XSens Technologies B.V. (2008a). *Magnetic Field Mapper MT Manager Add-on User Manual*. Revision D, April 1, 2008.

XSens Technologies B.V. (2008b). *MTi-G User Manual and Technical Documentation*. Revision B, April 1, 2008.

Xu, G. (2007). *GPS – Theory, Algorithms and Applications*. Berlin: Springer, 2nd edition.