Design, Implementation and Localization of a Mobile Robot for Urban Site Modeling

Atanas Georgiev

Submitted in partial fulfillment of the

requirements for the degree

of Doctor of Philosophy

in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2003

ABSTRACT

Design, Implementation and Localization of a Mobile Robot for Urban Site Modeling

Atanas Georgiev

This thesis presents a systematic and practical approach to mobile robot localization in urban environments. It reflects work on both system and algorithmic problems. The methods and ideas presented are generally applicable to mobile robots operating in urban environments.

On the system level, I have designed and built a functioning autonomous mobile robot. The design extended an existing robotic vehicle with a carefully chosen sensor suite of a digital compass with an integrated inclinometer, a global positioning unit, and a camera mounted on a pan-tilt head. The robot has been equipped with wireless networking.

I have also designed and implemented a distributed software architecture for mobile robot navigation. It addresses important issues like computation distribution, autonomy, flexibility and extensibility. A motion control component controls the robot pose and drives the robot to its destination. A graphical user interface

allows for remote control and monitoring.

On the algorithmic level, I have developed a localization system that employs two methods. The first method uses odometry, the compass module and the global positioning sensor. To reduce the effect of systematic odometry errors, I have extended an existing calibration procedure. An extended Kalman filter integrates the sensor data and keeps track of the uncertainty associated with it. When global positioning data is reliable, the method is used as the only localization method. When the data deteriorates, the method detects it and and seeks additional data by invoking the second localization method.

The second method is based on visual pose estimation. It is heavier computationally but is only used when it is needed. When invoked, it stops the robot, chooses a nearby building to use and takes an image of it. The pose estimation is done by matching linear features in the image with a simple and compact model of the building. A database of the models is stored on the on-board computer. No environmental modifications are required.

I have demonstrated the functionality of the robot and the localization methods with real-world experiments.

Contents

List of Figures **v**

List of Tables **viii**

Acknowledgments **ix**

Chapter 1 Introduction **1**

1.1 Dealing with the Real World is Hard 1

1.2 The Localization Problem is Difficult 4

1.3 Localization is Essential 5

1.4 The Problem Has Not Yet Been Solved 7

1.5 Scope of This Work 8

Chapter 2 Related Work **12**

2.1 Dead Reckoning 13

2.2 Global Positioning 15

2.3 Imaging 16

2.4 Sensor Integration and Uncertainty Handling 20

2.5 Other Approaches 23

2.6 Comparison with My Approach 25

Chapter 3 System Design and Implementation 28

3.1 The AVENUE Project 28

3.2 Hardware 31

3.2.1 Choice of Sensors 33

3.2.2 Other Hardware Considerations 36

3.3 System Architecture 38

3.3.1 Background 38

3.3.2 Issues 39

3.3.3 Design 42

3.4 Motion Control 48

3.5 User Interface 50

3.6 Summary 52

Chapter 4 Localization in Open Space 54

4.1 Odometry 54

4.1.1 Systematic Errors 58

4.1.2 Non-systematic Errors 62

4.2 Attitude Sensor 65

4.2.1 Pose Estimation 65

4.2.2 Kalman Filters 67

4.2.3 Integration with Odometry and Uncertainty Handling 69

4.3 Global Positioning Systems 72

4.3.1 Background 73

4.3.2 Coordinate Systems and Calibration 76
4.3.3 Fusing GPS with Odometry and the Attitude Sensor 79
4.4 Summary 81

Chapter 5 Visual Localization **83**

5.1 Environmental Model 85
5.2 Choosing a Model to Use 88
5.3 Pose Estimation 91
5.3.1 Preparation 93
5.3.2 Sampling and Pose Candidate Computation 95
5.3.3 Pose Candidate Refinement 97
5.3.4 Pose Candidate Evaluation 98
5.3.5 Number of Iterations and Speedups 102
5.4 Summary 104

Chapter 6 Experiments **105**

6.1 Localization in Open Space 105
6.2 Localization with Vision 108
6.3 Localization Using All Sensors 124
6.4 Summary 128

Chapter 7 Conclusion and Future Work **130**

7.1 Limitations 132
7.2 Discussion and Future Work 135
7.2.1 View Planning 135
7.2.2 Feature Matching 136

7.2.3 Partial Visual Localization 137
7.2.4 Visual Pose Integration 139
7.2.5 Obtaining the Model 140

Appendix A Implementation Details of the System Architecture 143

A.1 Overview 143
A.2 Description of the Hardware Servers 146
A.2.1 ATRV2Server 146
A.2.2 PTUServer 146
A.2.3 AttitudeServer 147
A.2.4 GPSServer 147
A.2.5 VideoServer 148
A.2.6 ScanServer 148
A.3 Description of NavServer 149
A.3.1 Localizer 149
A.3.2 Controller 149
A.3.3 Navigator 150

List of Figures

3.1 The modeling process. 30

3.2 The mobile platform used in this thesis. 32

3.3 GPS problems in urban areas: Not enough visible satellites in urban "canyons" (left) and signal reflections and multipath (right). . . . 35

3.4 The system architecture 43

3.5 The robot reference frame and a control target point. 49

3.6 The user interface. 51

4.1 The ATRV kinematics 55

4.2 The resulting distribution of the robot position after moving forward by a certain distance. 63

4.3 A diagram of the extended Kalman filter configuration 68

4.4 The differential GPS technique. 75

4.5 Cartesian and ellipsoidal coordinates. 77

4.6 The campus coordinate system 79

5.1 A sample model (right) of a building facade (left) 86

5.2 An example for choosing a model. 88

5.3 Criteria for choosing a model: distance and viewing angle 89
5.4 Preprocessing of the 3-D line segments: the original lines from the model (left) and the result after merging the collinear subsets (right). 94
5.5 Preprocessing of the 2-D line segments: the extracted edge lines (left) and the result after merging the collinear subsets and removing the short ones (right). 94
5.6 Error metric used for pose estimation 96
5.7 Coverage of a pair of matching lines. 99

6.1 The first test run in open space 106
6.2 A second test run returning to the starting point 107
6.3 3-D models used for localization shown on a 2-D map of the test area. 109
6.4 A map of the area where the experiments were conducted showing approximate camera locations and orientations. 110
6.5 Visual localization test: Location 1 111
6.6 Visual localization test: Location 2 111
6.7 Visual localization test: Location 3 112
6.8 Visual localization test: Location 4 112
6.9 Visual localization test: Location 5 113
6.10 Visual localization test: Location 6 113
6.11 Visual localization test: Location 7 114
6.12 Visual localization test: Location 8 114
6.13 Visual localization test: Location 9 115
6.14 Visual localization test: Location 10 115
6.15 Visual localization test: Location 11 116

6.16 Visual localization test: Location 12 116
6.17 Visual localization test: Location 13 117
6.18 Visual localization test: Location 14 117
6.19 Visual localization test: Location 15 118
6.20 Visual localization test: Location 16 118
6.21 Initial and final alignments in the pose estimation tests with a pair of images taken from the same location. 122
6.22 Initial and final alignments in the pose estimation tests with a pair of images taken from the same location. 123
6.23 A map of the area showing the robot trajectory (dotted line) and the locations where the robot used visual localization. Notice location 3 which is directly underneath a building extension. 125
6.24 Integration test: Location 1 126
6.25 Integration test: Location 2 126
6.26 Integration test: Location 3 127
6.27 Integration test: Location 4 127

7.1 A building facade exhibiting repetitive feature patterns 139
7.2 Model acquisition and simplification 141

A.1 An example for software component hierarchy 144

List of Tables

6.1 Computed and actual robot position for each of the locations where the visual localization test was performed. The last column shows the total error. Measurements are in meters. 120

6.2 Robot orientation as computed by the localization method (first three columns) and by Tsai's method (middle three columns). The last three columns show the discrepancies between the two estimates. Measurements are in degrees. 121

6.3 Robot position and error estimated by GPS, attitude sensor and odometry along with the corresponding improved position estimate and error after performing visual localization. Measurements are in meters. 128

Acknowledgments

I would like to begin by expressing my gratitude to my advisor, Professor Peter Allen, for his support, patience, and encouragements. He has had a profound influence on my way of thinking — both about academic matters and in general. He taught me how to pay attention to the "big picture" and prioritize accordingly. Whenever hardware broke — which happened much too often — he was always understanding and pragmatic. His insight into the American culture and the time spent with him outside of school made my student life a lot more enjoyable. For all that, and so much more: Thank you, Peter!

I would like to thank the members of my defense committee. Professor John Kender, who has kindly agreed to serve as the committee chair, has helped me improve my teaching skills and has given me a lot of food for thought with the interesting questions he always seems to come up with. I value Professor Shree Nayar's help with my research as well as his sharing of his opinion and ideas about academic research in general. I appreciate Professor Andrew Laine's and Professor Camillo Taylor's interest in my work and their desire to share their feedback.

I would also like to thank the members of the Robotics Laboratory, with whom I had the pleasure to share ideas and spend time together: Ioannis Stamos,

Paul Blaer, Ethan Gold, Andrew Miller, Paul Oh, Michael Reed, and Ben Smith. Special thanks go to Ioannis, who provided me with code and ideas; Paul Blaer, for writing the path planner, modifying various other programs, and helping me out with the robot; and Ethan, for his excellent work on the user interface and the fun he brought to the lab.

I am grateful to all my previous teachers and mentors who influenced my career. I thank my secondary-school teacher Hopteriev, who first sparked my interest in mathematics; Prof. Ivan Blyantov and Ivan Badev, for the wonderful years of extracurricular study of mathematics and the chance to participate in the competitions; my high-school teachers, Ognyan Gavrailov and Kosta Garov, for turning me to Computer Science and all the soccer we played; Prof. Krassimir Manev and Prof. Pavel Azalov, whom I had the pleasure to teach me at Sofia University; and my former advisor, Christo Dichev, who introduced me to scientific research and directed my work on my Master's thesis.

I would like to thank all my friends who helped me with ideas, support, and encouragement. I am especially thankful to Stanya Milanova, who got up early one morning to come to Columbia and help me with an experiment.

Finally, my deepest thanks go to my parents, Georgi and Lilyana. They have always encouraged my curiosity and have been extremely supportive in my strive for better education. Throughout my years at Columbia, they had to live with the thought of my absence for too long, being too far away, and seeing me at most once a year for a couple of weeks. Mom and dad: we went through this together.

To my mom, Lilyana, and my dad, Georgi

Chapter 1

Introduction

Mobile robot localization can be generally defined as the process of determining and keeping track of a mobile robot's location in its operating environment. It was adequately summarized by Cox in 1988 with the question "Where am I?" [Cox, 1988]. There are a few lessons that the brief history of Mobile Robotics has already taught us about localization.

1.1 Dealing with the Real World is Hard

In 1970, scientists from the Stanford Research Institute introduced Shakey, the first mobile robot that "reasons" about its actions. It was able to find a specific box in various rooms and move it to a designated position. To do so, among other things, it had to perform some kind of localization — it was equipped with wheel encoders, a laser range finder and a video camera which it used to detect its position with respect to the box and the surrounding obstacles.

The performance of Shakey turned out to be anything but satisfactory: It

took hours for the robot to find and slowly push a block. While the planning system generated good plans, the sensory interpretation of the world was very primitive. The environment had to be tailored to the limitations of the system: it could only recognize simple flat-faced uniformly-colored objects and the rooms had to be specially constructed. Although the main purpose of the project was to demonstrate the application of logical reasoning to problems from the physical world, the difficulty in implementing it was by far the most important lesson learned.

What makes it so difficult to interact with the real world? Most of the reasons can be traced back to a mobile robot having to rely on physical sensors and actuators that operate in a physical environment. Being part of the real world themselves, the sensors and actuators are prone to various imperfections which introduces a lot of uncertainty and limitations:

- **Measurement inaccuracy:** Each sensor measurement is accurate only to a certain extent which is typified by the the nominal accuracy of the sensor. Even if the theoretical accuracy is acceptable for the application, it may still not be achieved in practice due to factors like noise or a lack of good calibration.

- **Measurement uncertainty:** Beyond the accuracy of a measurement, a big question is even whether it reflects what we think it does. This is not necessarily the case. For example, wheel encoders on mobile robots are typically used to measure the distance traveled by the wheel but what they measure in practice is how much the wheel has turned. The two may differ significantly if the wheel slips. Another typical example is a sonar facing a corner. Due to multiple reflections of the ultrasonic sound from the walls, the sonar might

"see" a lot more free space ahead than there really is.

- **Measurement inconsistency:** To reduce the effect of inaccuracies and uncertainties, redundancy is typically sought in the form of additional sensors that measure the same quantity, preferably in a different way. However, this introduces other difficult problems: How does one efficiently register and fuse different kinds of data? And how does one resolve inconsistencies between multiple measurements? The latter problem can be encountered even with a single sensor if it takes multiple different measurements of the same static quantity.

- **Data interpretation:** Some sensors, like a camera or a laser range scanner, capture a lot of information about the environment. This is certainly desirable but is only useful to the extent the information can be recovered from the data. Unfortunately, this is not easy. In fact, the entire field of Computer Vision exists to explore ways to interpret visual data. Processing range scans is not much easier either, because of the sheer amount of data per scan and the fact that it only represents a discrete sample of the continuous physical world.

- **Various sensor limitations:** In addition to the above, each kind of sensor contributes with its own peculiarities and limitations. Sonars and range finders have limited range. Compasses need special calibration to undo the influence of the Earth's magnetic field. Global positioning systems (GPS) require a line of sight to a minimum number of satellites. Inertial sensors drift. All sensors have a limited resolution/sensitivity threshold.

- **Actuator accuracy and limitations:** Actuators, like sensors, have limited accuracy and other specific limitations. A step motor might turn 29.9 degrees instead of 30, which would cause the robot to arrive a millimeter away from the target. Motors have also limits on their speed, acceleration, torque, etc. To account for that, sensors must be used to detect discrepancies, which introduces all of the sensor-related problems.

- **Complexity:** Last but not least, the real world is simply too complex, dynamic and unpredictable to describe in a concise and precise algorithmic or descriptive form — the kind that is required by the computing technology today. The best one can hope for is a practically good approximation.

1.2 The Localization Problem is Difficult

To find out where it is, a mobile robot must rely on sensors. It can use **exteroceptive sensors** to detect cues about its position from the surroundings, **proprioceptive sensors** to compute the position from measurements of its own motion, or a combination of these. Regardless of the type of sensors used, the need for such introduces all of the issues with their physical nature.

Localization is even further complicated by the effects sensor-related problems have on the task. A typical example for this is odometry. Theoretically, knowing the radii of the robot wheels, if at any given moment the rotation of each wheel can be measured, it would be straight-forward to compute the robot position in a flat 2-D world. In practice, however, minute errors in the measurements by the wheel encoders accumulate with the distance traveled and at some point result in a

total error exceeding any reasonable limit. It only gets worse with inertial sensors, like gyros and accelerometers, which introduce their errors in the first or second derivative of what one really wants to measure.

Another typical example is the time constraint. For reliability reasons, the robot needs to update its position estimate frequently. Each iteration of the localization loop should be fast enough to allow for this to happen and to spare sufficient CPU time for the other processes to run. This places an upper limit on the time each position estimation step can take. On the other hand, sensors like a camera or a laser range finder, provide a lot of data that requires sophisticated algorithms and a long time to process reliably. Clearly, there is a trade-off and a balance has to be struck.

The time constraint is also important for a reason not directly related to sensor imperfections. Many position estimation algorithms are based on matching acquired sensor data to a model of the environment using distinguishing features for clues. Even with ideal sensors, robust feature detection and matching is not a trivial problem. In particular, if feature identities are not available, matching the sensor data with the model can easily become intractable for a relatively small number of features.

1.3 Localization is Essential

There are many ways in which mobile robots can be useful and few people doubt that the future holds an important place for them in our lives. They are expected to help us alleviate, and possibly completely take over tedious, monotonous and painstaking tasks. Examples include data acquisition (scanning, surveying), envi-

ronmental mapping and site modeling, material transportation and delivery, security (area patrolling), fire safety, and recovery from disasters. Some of the earliest adopters were the military, who have been researching uses such as hazard detection and elimination, reconnaissance, assistance and others. Commercial entities are trying to bring mobile robots to our homes in the forms of autonomous mowers and vacuum cleaners. Some level of autonomy is envisioned in our cars and the wheelchairs of disabled people.

What is the role of localization in this vast sea of potential applications?

In 1991, Leonard and Durrant-Whyte — two of the prominent figures in Mobile Robotics — summarized the problem of mobile robot navigation with three fundamental questions: "Where am I?", "Where am I going?" and "How should I get there?" They came to the conclusion that solving robustly and reliably the localization problem ("Where am I?") is essential to answering the remaining two questions.

Indeed, localization is both fundamental and essential for a mobile robot. It is fundamental because it provides the basis on which other common tasks — on both the users' and the developers' ends of the spectrum — are relying. From a developer's perspective, one uses the current robot position as a starting point in planning a path to the target or adjusting the robot's motion to follow that path. From a user's standpoint, applications, such as surveying, environmental modeling and hazard detection, tag their data with the location where it was acquired.

Localization is also essential since it is a prerequisite for performing almost any useful task. Without an estimate of its own location, a robot would not be able to find its way around an obstacle or will not know if it follows the right path.

In fact, it will get lost and will have no idea where it needs to go. Path planning becomes meaningless, as do all the data acquisition applications.

Being fundamental and essential makes localization one of the most important problems in Mobile Robotics. The significance of the problem has been further demonstrated along with a comprehensive study of the state of the art by Borenstein et al [Borenstein and Feng, 1996b].

1.4 The Problem Has Not Yet Been Solved

Today, more than three decades after Shakey's introduction, accurate localization is still the focus of most of the mobile robotics research as evident by the enormous number of publications on the subject and the special interest it receives at Robotics conventions.

Certain progress has been made and some robots have gone out of the labs. An excellent example for a successful system was demonstrated in the summer of 1998 [Thrun *et al.*, 1999]. For a period of two weeks, an autonomous robot, *Minerva*, acted as a museum guide and entertained visitors to the Smithsonian's National Museum of American History. During this time the robot traversed a total of more than 44 km having to constantly address issues like safe navigation in a populated and dynamic environment.

More recently, commercial companies have started experimenting with various autonomous devices that people can use in their homes. The company *Friendly Robotics* manufactures and sells *Robomower* — a fully automatic lawn mower [Robomower, 2000]. The UK-based *Dyson* sells a robotic vacuum cleaner with 50 sensors and 3 computers on board [Dyson, 2002]. Another European company, *Eureka*, is

in the process of testing their version of an autonomous vacuum cleaner, the *Robo Vac* [Eureka, 2001].

Unfortunately, these examples are rare exceptions, not the rule. Most mobile robots of today still belong to the research labs and one of the major reasons for that is the lack of robust localization methods in common, everyday environments. Upon closer inspection, one will find that all commercial examples above solve the localization problem by... avoiding it — the devices simply move in a preprogrammed or random patterns. And while *Minerva* did an excellent job at not getting lost, it relied on a number of assumptions about a closed indoor type of environment.

The progress of outdoor mobile robots is slower. As it turns out, methods that succeeded indoors are not easily converted to outdoor operation. Many of the simplifying assumptions, commonly accepted indoors, do not hold outside. Long and narrow corridors, ceiling lights, constant illumination, the lack of large open spaces are just a few examples. Popular sensors, such as sonars and most range finders, have a very limited operating range. An extremely detailed model, such as the evidence grids used indoors, of a reasonably large site is prohibitive, since computers would not be able to store and process it. Most of all, indoor environments are human controllable to a large extent; outdoor environments are not.

1.5 Scope of This Work

The work presented here is a systematic approach to mobile robot localization in urban environments. It consists of both system and algorithmic components which are equally important.

On the system level, it describes the efforts and considerations that were put to materialize the idea of a mobile robot capable of navigating autonomously in urban settings. A functional prototype has been built, equipped with odometry, an attitude sensor, a GPS unit, and a camera mounted on a pan-tilt unit. A distributed modular software architecture has been designed to address issues like computational and storage resource limitations and provide the user with ease and flexibility in monitoring and controlling the robot.

Key features of the localization method are the careful choice of sensors to use, their successful integration into a single reliable system, and the intelligent exploitation of important characteristics specific to urban sites. An important advantage is the fact that no environmental modifications are needed. To help reduce the complexity and ambiguity of processing the rich visual information, a simple and compact model of the operating environment is utilized. For the purpose of this work, the model is assumed to be given *a priori*, however, its automated construction is possible and is being addressed in the closely-related AVENUE project, described below.

Urban environments pose their own unique set of challenges. They are more complex, dynamic and unpredictable than the well-structured and more human-controlled indoor ones. A typical urban site can be orders of magnitude larger. Global positioning systems have problems with the obstruction of the direct line of sight to satellites by nearby buildings, causing multipath and degraded accuracy.

On the other hand, urban settings have unique characteristics that can be used to facilitate the estimation of the robot location. Of all outdoor environments, urban areas seem to possess the most structure in the form of buildings. The laws

of Physics dictate common architectural design principles according to which the horizontal and vertical directions play an essential role and parallel line features are abundant. The system presented here takes advantage of these characteristics.

The choice of urban environments is not arbitrary. A good solution to the localization problem in urban settings would be a key factor for the development of quality site-modeling applications. Site models are used in a number of areas, such as city planning, urban design, fire and police planning, military applications, virtual and augmented reality, and geographic information systems (GIS). This modeling is currently done primarily by hand and, owing to the complexity of these environments, are extremely painstaking. The models built are often incomplete and updating them can be a serious problem.

My work for this thesis is a part of the larger project AVENUE[1], which addresses the above needs by developing an autonomous mobile robot system for data acquisition and creation of 3-D photo-realistic geometrically accurate models of urban scenes. The development of such a system involves work on both hardware and software as well as on making them work together. The prototype system described in the thesis serves as the mobile scanning platform for the AVENUE project.

I see the contributions of my work in the following areas:

- The design and creation of a fully operational mobile robot platform for urban site modeling

- The design and implementation of a distributed software architecture for au-

[1]AVENUE stands for an Autonomous Vehicle for Exploration and Navigation in Urban Environments

tonomous mobile robot monitoring and control

- A localization method for outdoor robot localization in open areas using odometry, an attitude sensor, and a global positioning sensor
- A novel method for mobile robot localization in urban areas using odometry, an attitude sensor, global positioning and computer vision.

The rest of this document is organized as follows: Chapter 2 contains a discussion of the state of the art in the area and how existing work compares to the method presented here. Chapter 3 describes the robot along with the hardware and system work that was done to make the real-world implementation and testing of this method possible. Chapter 4 describes the first stage of the localization system that functions in open areas and the following Chapter 5 focuses on the second stage which involves the vision component. Following are Chapter 6 which presents the experimental results and Chapter 7 which outlines the limitations of the method along with ideas for its improvement and future extensions.

Chapter 2

Related Work

There has been a tremendous amount of work on mobile robot localization in the past twenty years. Most of the research effort has been focused on indoor environments, mainly because they are simpler, more structured and easier to control. An increasing number of researchers, however, are undertaking the challenge to take the robots out of their labs.

Sensors and methods for indoor localization have been comprehensively reviewed in two books [Borenstein and Feng, 1996b, Everett, 1995]. Another excellent book presents case studies of successful mobile robot systems [Kortenkamp *et al.*, 1998]. In this chapter, I will discuss existing localization methods from the perspective of outdoor applications. The next three sections review existing approaches according to the sensing techniques used. They are followed by a section on sensor integration. Various other approaches are summarized after that and the last section of this chapter presents a comparison of my method with related ones.

2.1 Dead Reckoning

Dead reckoning is a localization method that involves the use of a simple mathematical procedure to compute the current location of a vehicle, based on a known previous location and measured heading and velocity information throughout the time of travel. A typical example is odometry, which deduces the vehicle position by monitoring the rotation of the wheels.

Because of its simplicity and low-cost, odometry is implemented on almost every mobile robot. This also means that its inherent error accumulation problem is omnipresent and, consequently, a lot of effort is put in making it work better. Two orthogonal directions have been pursued: better calibration and integration of odometry with other dead reckoning sensors.

Borenstein and Feng have extensively studied the sources of odometry errors as well as benchmarking methods for quantifying the errors. They have classified odometry errors into systematic, which are caused by kinematic imperfections of the vehicle, and non-systematic [Borenstein and Feng, 1995]. They have, further, developed a calibration method, that can reduce the effects of two major sources of systematic errors. The method consists of having the robot traverse the same square path a number of times both clockwise and counterclockwise. The bias of the odometry system is computed based on the centroids of the clusters of arrival points and is used for software compensation.

Roy and Thrun have later shown how odometry calibration can be performed automatically by using an incremental maximum likelihood estimator [Roy and Thrun, 1999]. Their experiments showed an 83% error reduction rate along trajectories 741 m and 269 m long.

Gyroscopes have been frequently used to complement odometry. Two examples for this have been suggested by Borenstein and Feng [Borenstein and Feng, 1996a] and Maeyama *et al* [Maeyama *et al.*, 1996]. Both of them rely on the fact that the readings of the two sensors agree most of the time, except for when the robot encounters irregularities, such as bumps. During such an encounter, Borenstein and Feng simply rely on the gyroscope, while Maeyama *et al* use a more comprehensive model allowing the estimation of the gyro drift and detection of sensor failures.

Two other methods have added a magnetic compass to the above sensor configuration. Hardt *et al* model the vehicle dynamics and use an extended Kalman filter to fuse the redundant information from a compass and a gyro effectively reducing the effects of drifts and magnetic field perturbations [von der Hardt *et al.*, 1996]. A Kalman filter has also been used by Roumeliotis *et al* to estimate the robot attitude [Roumeliotis *et al.*, 1999]. They have considered both the 2-D and the 3-D case. However, they point out that modeling the vehicle dynamics has serious drawbacks when used with a Kalman filter and work around that by implementing the indirect formulation instead.

An early approach to combining odometry with gyros and accelerometers, has been developed by Fuke and Krotkov [Fuke and Krotkov, 1996]. The authors consider a lunar robot vehicle and address the problem of odometry position overestimation due to the effects of uneven terrain. They formulate an error-state extended Kalman filter that efficiently integrates the three sensors for improved attitude estimation.

A method for the alignment of an inertial measurement unit has been devised by Dissanayake *et al* [Dissanayake *et al.*, 1999]. They exploit the non-holonomic con-

strains of their vehicle and an assumption about sufficient excitation in all degrees of freedom to compute the roll and pitch angles of the sensor. Their method is attractive in that it works on the fly and does not require any external observations.

Regardless of the improvement they achieve, all researchers agree that dead reckoning alone is insufficient as a localization method. External references are needed to cope with the error accumulation problem.

2.2 Global Positioning

With the rapid development of technology, GPS receivers are quickly becoming the sensor of choice for outdoor localization. They are typically used in combination with dead reckoning and these are the approaches that I will describe in this section. Methods for integration of GPS with other sensors will be presented in section 2.4.

Cooper and Durrant-Whyte were among the first to advocate the combination of global positioning and inertial sensors on autonomous land vehicles [Cooper and Durrant-Whyte, 1994]. They started with a standard eight-state extended Kalman filter model of the vehicle position, velocity and clock drift and showed that this model is inadequate because of time correlation in the assumed white noise sequence. They solved the problem by adding six additional states and demonstrated how to correctly tune the filter. However, their experimental results showed an error in the order of tens of meters, perhaps because they used a standalone civilian-band receiver at times when Selective Availability was still active. The authors' conclusion was that GPS requires additional external sensors in order to be usable.

Schleppe has investigated the use of GPS for real-time attitude estimation [Schleppe, 1996]. He has built a platform on which he mounted an array of

four GPS antennae forming a square. The attitude is estimated using interferometric techniques: The predictions of the platform attitude by an extended Kalman filter are corrected by double differenced observations between pairs of antennae and pairs satellites. Static field test revealed accuracies of less than two degrees for each of the Euler angles.

Aono *et al* have investigated the problem of localizing an autonomous mower on undulating ground [Aono *et al.*, 1998]. Their sensor suite consists of a real-time kinematic GPS, a fiber optic gyro and a roll-pitch sensor. The integration is performed by an extended Kalman filter. To test their system, the authors simulated various levels of noise into the otherwise accurate to $2\,cm$ GPS readings, and compared the results of the sensor integration algorithm with the uncorrupted GPS data. When a noise with standard deviation of $1\,m$ was used, their system produced estimates with an error of $0.2\,m$.

2.3 Imaging

Imaging sensors, such as CCD cameras and especially laser range finders, have become very popular components for indoor localization and navigation applications. Small line-based laser range finders have proven particularly attractive because of their speed and simplicity — coordinates of the target points are easily obtained, without the need for sophisticated calculations. Additionally, cameras are used with considerable success, since a lot of the complexities involved in understanding the scene are simplified by the structure of these environments.

Conversely, due to the enormous complexity and dynamics of outdoor scenes, fewer assumptions can be made about them. 2-D laser scanners no longer depict

the environment accurately and the 3-D ones are either large, heavy, slow or noisy. More computation and advanced vision techniques are needed to compensate for that.

According to their need for *a priori* information, the image-based methods can be classified as landmark-based, model-based, and ones that do not require any information. While the distinction between landmark-based and model-based methods is not clear, generally, a model is a more comprehensive description than a set of landmarks.

The landmarks that are used could be artificial or natural. The localization process normally consists of three steps: **detecting** landmarks on an acquired image, **matching** the detected landmarks with stored data, and **computing** the position based on the matches.

The methods using artificial landmarks solve the detection step by choosing easily distinguishable marks. A good example is the work of Lin and Tummala [Lin and Tummala, 1997], who use a special circular pattern and develop a technique called *The Modified Elliptical Hough Transformation* to detect its location and orientation on an image. Their method seems to be very stable for viewing angles of less than 20° and occlusions of less than 25%.

The main difficulty in the artificial landmark approaches is the matching step. Since their landmarks are usually identical a many-to-many matching procedure is involved, which can lead to exponential complexity. Additional drawbacks are the need for environmental customization. To address these issues, many researchers are trying to use landmark features that are already present in the environment.

Nayar *et al* [Nayar *et al.*, 1994] have started a trend of localization ideas by using principal component analysis to perform real-time pose estimation and tracking of objects. They have demonstrated the efficiency of their approach on an indoor robot manipulator. Their work has been extended to mobile robots by Krose [Krose and Bunschoten, 1999]. A similar approach has been taken by Thrun who trains a neural net to estimate the pose of the camera based on a maximum-likelihood criterion [Thrun, 1998]. Sim and Dudek note that the above approaches require an extensive training set and suggest a method that automatically detects landmarks in images, based on local extrema of edge density [Sim and Dudek, 1998]. Their method greatly reduces the data base size, since it stores only the principal components of small windows around the landmarks.

None of the above methods has actually been tested outdoors although they are in principle not limited to indoor usage. They require extensive training and their landmarks data base is inappropriately large. They are also very sensitive to occlusions, landmark misidentification and mismatching. The latter are better handled by model-based approaches because of their more detailed information about the environment.

One typical model-based approach has been demonstrated on the robot FINALE [Kosaka and Kak, 1992]. The robot stores a detailed CAD model of its operating environment. At each step, the estimated robot pose uncertainty is used to generate an expectation of what the camera would see. When an image is taken, the expectation map is used to apply selective feature extraction. The predicted and extracted features are matched and an extended Kalman filter is used to update the robot pose and its uncertainty.

Zheng and Tsuji have described a method that they have actually implemented and tested in an urban area [Zheng and Tsuji, 1992]. They propose a model of the environment around the route of the robot which they call *panoramic representation*. The model is built out of vertical slices from images taken sideways as the robot is taught its route. The localization during an autonomous task is performed by matching the panoramic view generated so far with the memorized version. Although tested outdoors, this method has strong limitations. It is only applicable to localization along a single path and is sensitive to how closely this path is followed. The model is very rich and has significant storage requirements. Matching is also very slow: the authors report a 30 second processing time frame.

Finally, I want to mention two methods that do not require any *a priori* information about the environment [Adam *et al.*, 1999, Basri *et al.*, 1999]. Basri *et al* have been able to navigate the robot to a position and orientation from which a specified image was taken [Basri *et al.*, 1999]. This approach is known as localization by homing. Its major drawback is that it requires a significant overlap between the initial and target images. It is also strongly dependent on a good robot control system. Adam *et al* use fixation on an automatically selected landmark to improve the navigation accuracy [Adam *et al.*, 1999]. Their method is attractive because it requires only one landmark to be tracked, which greatly simplifies the computation. On the other hand, tracking a landmark is a very unstable and sensitive to occlusions process.

2.4 Sensor Integration and Uncertainty Handling

The issues of sensor integration and uncertainty handling are tightly related because the representation of uncertainty very often dictates the way sensor data is fused. Three dominant ways to address this issue have been established:

1. **Kalman filters.** Under this scenario, both the motion and measurement models are approximated by Gaussians. This is almost certainly used in combination with a Kalman filter. There are multiple advantages to this approach: First, a Gaussian distribution is completely characterized by its first two moments, hence, the uncertainty is represented simply by the state vector and a variance-covariance matrix. Second, manipulating these is relatively fast and easy — the entire well-understood apparatus of linear estimation theory can be taken advantage of. Further, the Kalman filter provides an excellent framework for the fusion of various observations from multiple sensors. A drawback of this approach is that it can not represent multi-modal distributions.

2. **Grid-based methods.** These methods represent the uncertainty of the vehicle pose explicitly by discretizing the state space into a number of cells and storing into each cell the probability of the robot being there. They are attractive in cases when multi-modal distributions need to be used. One obvious drawback is the *a priori* commitment to a particular resolution. Additionally, they have huge storage requirements and impose excessive computational burden as theoretically each of the cells needs to be updated with each new measurement.

3. **Particle-based methods.** These methods were invented to address the issues of the grid-based approaches. The idea is to describe the probability distribution of the robot pose by a number of representative samples randomly drawn from it, instead of discretizing the entire state space. This obviously solves the problem with the storage and computational requirements. In fact these methods are very easy to implement. A serious concern with them, however, is how to make sure that the sample particles remain representative of the distribution at each step. Another problem with these methods is the need for a fairly complete and detailed map of the entire working area.

Below, I present a number of examples for each category.

Kotani *et al* have integrated GPS, vision and a fiber optic gyro [Kotani *et al.*, 1998]. They fuse the data from odometry, the camera and the gyro using a standard extended Kalman filter framework. They apply a Hough transform on the images to detect curves and straight lines and match them with a map. Successful matches are used to correct odometry. An experimental test, performed on a 1179 m long trajectory, demonstrates the feasibility of this approach. However, there is no real integration between the GPS and the other sensors: GPS is only used in the beginning to obtain the initial position and orientation of the vehicle. Another limitation of this approach is the requirement for a full metric map to use vision.

Significant work on the integration of odometry, a millimeter wave radar, GPS and INS has been done at the Australian Centre for Field Robotics [Bozorg *et al.*, 1998, Sukkarieh *et al.*, 1998]. Their idea consists of implementing a decentralized navigation architecture with two independent loops: one, that employs data

from GPS and INS, and another, that fuses data from odometry and the radar which measures the distance to beacons placed in the environment. Each loop is implemented as an information filter, which is a form of the Kalman filter. The estimates of the two loops are blended by an assimilation unit which can work in both feedback and feedforward configurations.

A very elegant and general approach to sensor fusion has been published by Neira *et al* [Neira *et al.*, 1999] and explained in detail in [Castellanos and Tardos, 1999]. The authors have devised an uncertainty representation model, called the *SPmodel*, which can represent different types of geometric elements — such as points, rays, segments, and corners — in the same common framework. According to the model, the location of each element is represented by an estimation of a small perturbation from a local base, its covariance, an estimate for the base of the perturbation and a binding matrix describing the possible symmetries resulting from the specific type of the element. The model deals easily with partial and imprecise data. The authors have adapted the extended Kalman filter to the peculiarities of their representation and have demonstrated a working version of an indoor robot equipped with a camera and a range finder.

A good example for a grid-based approach is the one developed by Thrun *et al* [Thrun *et al.*, 1998]. They assume a flat world and discretize the 3-D state space consisting of the robot position (x and y) and orientation (θ). To find the robot pose, they employ a hill-climbing EM algorithm in likelihood space. The algorithm alternates two steps: *expectation*, in which the best robot pose is computed given the current map estimate, and *maximization*, in which a maximum likelihood map is estimated. To cope with the computational burden, various approximations and

optimizations have been made.

Finally, a classical particle-based method has been developed by Dellaert *et al* [Dellaert *et al.*, 1999]. They have employed the Monte Carlo technique to update the probability density representation over time. Actual tests were performed on in the National Museum of American History and the impressive average speed of $1.6\,m/s$ was achieved.

2.5 Other Approaches

This section briefly outlines approaches to localization that utilize ideas other than tracking the robot pose over time.

A very popular and successful idea is to exploit the duality between localization and modeling and address both issues in the same process. This is known as *Simultaneous Localization and Map Building (SLAM)*. Work in this direction has been done by Thrun *et al* [Thrun *et al.*, 1998], Durrant-Whyte *et al* [Durrant-Whyte *et al.*, 1999], Leonard and Feder [Leonard and Feder, 1999], and Castellanos *et al* [Castellanos *et al.*, 1998]. This method, however, may not be preferable if a map of the environment is already available and the modeling process is more computationally demanding than the localization. It should also be pointed out that models produced by SLAM applications are designed to facilitate localization and are not necessarily substitutes for the detailed photo-realistic models one would expect from a quality site-modeling application.

Fundamentally different from the above is the idea of qualitative localization, where the goal is to determine the robot location on a coordinate-free topological model of the environment. Levitt *et al* describe a landmark-based formal theory for

multi-level spatial representation of environmental locations encoding perceptual knowledge. Their model is topological but also allows for smooth integration of metric information. Localization is done by recognizing landmarks in the vicinity and associating them with a "visual landmark memory" [Levitt *et al.*, 1987].

Park and Kender suggest the use of topological directions for navigation in large-scale environments [Park and Kender, 1995]. The directions are a form of an environmental model tailored to the perceptual capabilities of the mobile robot. The robot localizes itself by watching for events that are characterized by spatial properties of the environment. The authors also consider the problem of error handling by analyzing the types of errors that are detectable and recoverable.

A similar approach is taken by Taylor and Kriegman [Taylor and Kriegman, 1998]. They consider the problem of systematically exploring an unknown environment in search of one or more recognizable targets. Their exploration algorithm is based on landmark visibility. It constructs a representation of the environment, called *boundary place graph*, which records the set of recognizable objects visible from the boundary of each configuration space obstacle. The authors provide a necessary and sufficient condition under which the algorithm is guaranteed to discover all landmarks. The robot maintains a notion of its location by measuring *a relative range* — any quantity that is monotonically related to the distance to a landmark. No absolute reference or measurements are needed.

An example for a somewhat mixed approach between topological and quantitative methods is the autonomous road following or highway navigation [Thorpe, 1990]. The main goal in this case is not to estimate the position of the vehicle but to simply keep it on the road. However, due to the inertial properties of most such

vehicles, estimates of its velocity and relative position to obstacles or other vehicles is strictly necessary.

2.6 Comparison with My Approach

The localization method described in this thesis is one that integrates odometry, an attitude sensor, GPS and vision. I have implemented an extended Kalman filter framework for sensor integration and represent the uncertainty in the robot pose by a Gaussian distribution.

I have chosen the Kalman filter over the particle-based methods since the availability of a complete map of the working area can not be guaranteed and there is no clear way to sample the probability distribution of the robot pose over the entire working area. A grid-base approach would clearly be prohibitive for such large-scale environments.

The qualitative approaches are very appropriate for outdoor usage in that they address the problem of limited computational resources. A topological description of the environment could be extremely compact and is best suited for "high-level" path planning and navigation. In my case, however, the robot needs to have a good quantitative estimate of its location as this is essential for the site modeling application.

I am aware of two other projects specifically addressing the localization of mobile robots in urban environments. Chen and Shibasaki have also noticed that GPS accuracy becomes poor for navigation in many areas [Chen and Shibasaki, 1999]. They have addressed the problem with the addition of a camera and a gyro to achieve a better and more stable performance. While their work does not discuss

how the sensor integration is done, it is clear that the camera and an environmental model obtained from a geodetic information system (GIS) play key roles in their method. The authors have considered situations where GPS, GIS or both data are unavailable and have implemented both absolute and relative to previous pose localization. They seem to have consistently been able to compute locations with a sub-meter accuracy.

One of the drawbacks of their research is the requirement for a geodetic information system with accurate data. The authors also seem to only process vertical edges of the building structures very few of which may be visible from a large number of places.

The second approach to vehicle localization in urban areas is one proposed by Nayak [Nayak, 2000]. His sensor suite consists of four GPS antennae and a low-cost inertial measurement unit. The sensor integration is done by an indirect Kalman filter estimating the error of the inertial measurement unit, similar to my case. Since the sensors are mounted on a car moving at sufficiently high velocities (greater than $10\,m/s$), he is also using the velocity and heading readings from the GPS sensors. Tests were performed using simulated GPS outages of 20 second duration. The resulting error in the estimate was around $9\,m$.

An obvious drawback to this approach is the requirement for four GPS receivers mounted on a relatively large platform. The resulting error, while sufficient for many applications, is not acceptable for mobile robot navigation.

A closely related work is the MIT City Scanning project, whose goal is similar to AVENUE's — the creation of accurate three-dimensional models of urban landscapes. Antone and Teller address the camera pose estimation problem from

multiple images of urban scenes. They use a mobile platform equipped with various instrumentation to acquire images at various locations in the operating site. A number of images are acquired at each location and used to create a data base of hemispherical mosaic nodes. Each node is annotated with the approximate pose derived from sensors like odometry and GPS. The data base is then processed to align the images and compute the camera orientations [Antone and Teller, 2000b] and positions [Antone and Teller, 2000a].

The main difference between their approach and mine is that Antone and Teller focus on the modeling as opposed to the localization problem. They split the process in two steps and recover the camera poses after all images are acquired. They do not address the problem of an autonomous platform performing the acquisition step. Their method is inapplicable to the task of robot localization as this task requires on-line pose estimate computation. Further, their method precludes sensor placement planning which is an integrable part of the modeling system used in AVENUE.

Finally, another related work in that it employs model-based pose estimation is the Facade project [Taylor *et al.*, 1996]. The focus of this project is again on the modeling not the localization aspect. It provides an interactive tool with which a user with some manual interaction is able to construct 3-D models of buildings from multiple images of them while simultaneously recovering the camera poses.

Chapter 3

System Design and Implementation

In this chapter, I will discus the work that was done to design and build a functional mobile robot for urban site modeling. This is the robot that was used as a test platform for the localization methods described in the following chapters. While the localization methods are general and independent of the modeling part, an explicit goal of the design effort was to tailor the robot to the task of urban site modeling as part of the AVENUE project [Allen *et al.*, 2001]. Thus, I will begin the chapter with a brief description of AVENUE in order to provide a better understanding of the context in which the various design decisions were made.

3.1 The AVENUE Project

The AVENUE project targets the automation of the urban site modeling process. The main goal is to build not only realistically looking but also geometrically accu-

rate and photometrically correct models of complex outdoor urban environments. These environments are typified by large 3-D structures that encompass a wide range of geometric shapes and a very large scope of photometric properties. The models are needed in a variety of applications, such as city planning, urban design, fire and police planning, and many others. However, they are typically created by hand which is extremely slow and error prone. AVENUE addresses these problems by building a mobile system that will autonomously navigate around a site and create a model with minimum human interaction if any.

The entire task is complex and requires the solution of a number of fundamental problems:

- In order to create a single coherent model, scans and images of the scene taken from multiple viewpoints need to be registered and merged.
- To provide full coverage, there must be a way to tell what part of the scene is not yet in the model and where to go to ensure a good view of it. Better yet, it is desirable to plan the data acquisition so as to minimize the cost (such as time, price or total travel distance).
- Given a desired acquisition location, a safe path must be determined that will take the sensors there from their current position.
- To acquire the desired data, the sensors need to be physically moved and accurately positioned at the target locations.
- The user must be able to monitor and control the process at any given stage.

The first two problems have already been successfully addressed by Stamos [Stamos, 2001]. He developed a comprehensive system for the automated

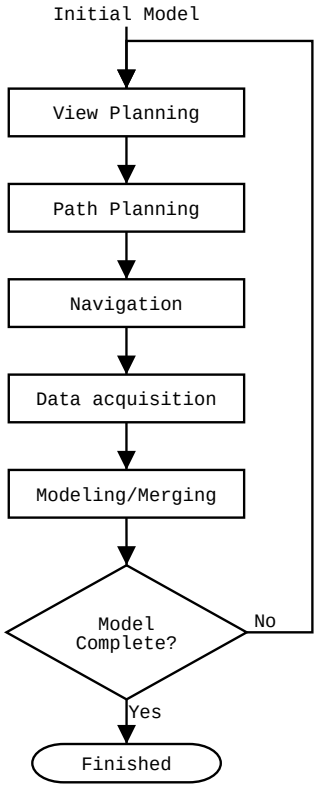


Figure 3.1: The modeling process.

generation of accurate 3-D models of large-scale environments using a laser range finder and a camera and provided a framework for sensor planning utilities. He demonstrated the feasibility of his approach by creating models of various buildings. Additionally, a safe trajectory generator based on Voronoi diagrams has been developed in our lab by Paul Blaer [Gueorguiev *et al.*, 2000].

Here, I am addressing the remaining two items in the list by introducing an autonomous mobile platform operating according to the following scenario (Figure 3.1): Its task will be to go to desired locations and acquire requested 3-D scans and images of selected buildings. The locations will be determined by the sensor planning (or view planning) system and will be used by the path planning system

to generate reliable trajectories which the robot will follow. When the rover arrives at the target location, it will use the sensors to acquire the scans and images and will forward them to the modeling system. The modeling system will register and incorporate the new data into the existing partial model of the site (which in the beginning could be empty) according to Stamos's method. After that, the view planning system will decide upon the next best data acquisition location and repeat the above steps. The process will start from a certain location and gradually expand the area it has covered until a complete model of the site is obtained.

3.2 Hardware

The design of a mobile robot for urban site modelling involves a myriad of issues related to both its safe navigation (including motion control, path planning, and localization) and the necessary data acquisition (range scans and images). The most important issues are: the choice of a robotic vehicle, the choice of sensors, the sensor placement, and the means of remote communication with the robot.

Ideally, all of these items are considered at the design stage, before moving on to implementation. In practice, some of the design decisions are already made by virtue of existing hardware or other practical considerations. This was the case for me with the mobile platform and the scanning equipment.

The mobile robot used as a test platform for this thesis is an ATRV-2 model manufactured by Real World Interface, Inc (Figure 3.2). It has built-in odometry and twelve sonars. It has a regular PC on-board and provides additional $24 - 45\,W$ of $+5\,V$ and $+12\,V$ power supply which amounts to about three hours of standalone operation. Its payload of $118\,kg$ is enough to accommodate the scanner,

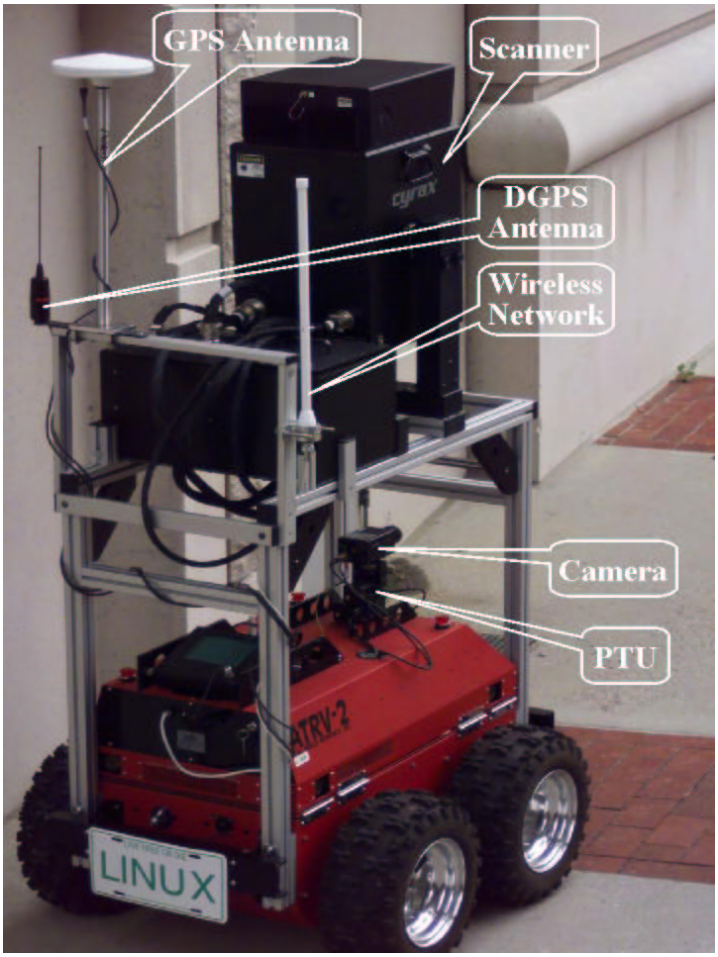


Figure 3.2: The mobile platform used in this thesis.

its electronics box including the batteries, and the additional periphery I have installed. The robot can move as fast as $2\,m/s$ and handle slopes of up to 45 degrees. An additional benefit is that it is a *holonomic* vehicle, that is it has zero turning radius.

The choice of sensors is an important topic that I would like to discuss in more detail in the next subsection. The rest of the hardware-related work is described in the following subsection.

3.2.1 Choice of Sensors

The sensor suite on the robot has to provide enough data to allow for safe navigation and accurate modeling. The modeling method already developed dictate the use of a 3-D range finder and a color camera. Both are provided by a Cyrax 2500 laser scanner that has a nominal accuracy of $6\,mm$ and range of up to $100\,m$.

In terms of navigation, the robot needs to be able to detect unexpected obstacles and its position in the environment. The sonars on the robot provide for a very good solution to obstacle detection. For localization, however, the answer is more complicated, as there are many factors to consider.

An essential requirement for localization is availability at all times, that is at any given moment the robot has to have an idea of its location. This is best accomplished by *proprioceptive sensors* — ones that deduce the robot pose by relying only on measurements from its kinematics, such as wheel revolutions and joint angles and velocities. Odometry (wheel encoders) is the typical choice as it is provided in almost every mobile robot (including the ATRV-2), requires little computation, and works in real-time.

The use of odometry, however, is not sufficient if a 3-D position is needed and the ground plane assumption does not hold (that is, the ground surface is not flat as it is mostly the case outdoors). In such cases, odometry is often complemented with inertial sensors such as fiber optic gyros and accelerometers. The problem with these sensors is that they measure the first or second derivative of the magnitude that we need. This, combined with the fact that they tend to drift, causes error accumulation and makes them problematic for long-range operations.

A better choice of a complementary sensor is one that directly measures the

desired quantity and thus produces an error bounded by a known nominal constant. In my case, this is done by a digital compass with an integrated roll-pitch sensor (Honeywell HMR3000), which provides fast update rates of up to $20\,Hz$. Its roll-pitch accuracy is $0.6°$ and its heading accuracy is better than $1.5°$ RMS. The heading accuracy also depends on the presence of strong magnetic sources other than the Earth, however, the module provides a way to detect such magnetic influences.

An inherent limitation of the proprioceptive sensors is that error accumulation with them is inevitable. This can only be alleviated by the addition of an *exteroceptive sensor* — one that makes measurements relative to the robot environment. One typical requirement of such sensors is an installed infrastructure, such as radio transmitters, beacons, or fiducial marks. This is not always a plausible option. Fortunately, there are two existing and available infrastructures, GPS and GLONASS, that are designed to address exactly this problem anywhere on or near the Earth. State-of-the-art Real Time Kinematic (RTK) receivers achieve accuracies in the order of centimeters and this makes them the primary candidates for inclusion on a mobile robot. Our robot is equipped with an Ashtech GG24C GPS+GLONASS unit that is accurate down to $1\,cm$ in an RTK mode.

It is important to note that a GPS unit can not act as a substitute for a proprioceptive sensor, since it has stringent requirements to work (e.g. line of site visibility to enough satellites, good satellite configuration, acceptable signal-to-noise ratio) and is not real-time. Thus, GPS can only act as a complement to the real-time proprioceptive sensors.

The combination of a proprioceptive sensor and GPS has been proven very beneficial. GPS is known to exhibit an unstable high-frequency behavior mani-

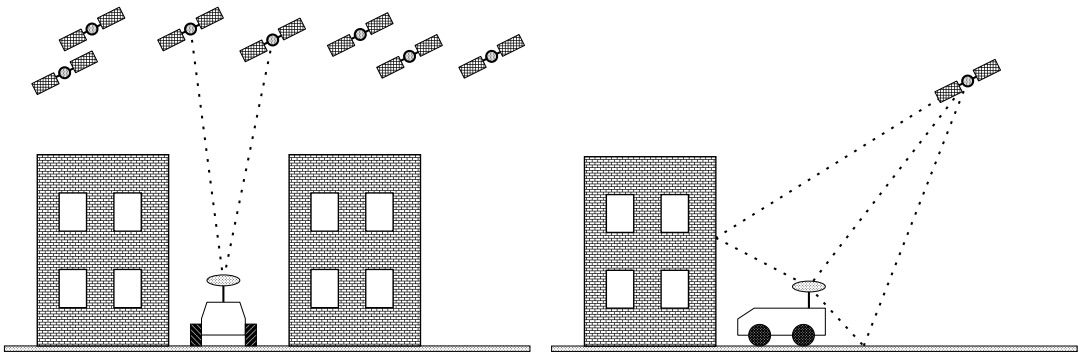


Figure 3.3: GPS problems in urban areas: Not enough visible satellites in urban "canyons" (left) and signal reflections and multipath (right).

fested by sudden "jumps" of the position estimates when the satellite configuration changes or a signal reflection takes place and confuses the receiver. It is fairly reliable, however, over a longer period of time when enough data is collected and errors are "averaged out". On the other hand, proprioceptive sensors drift gradually and rarely suffer the sudden jump problem. Thus, an intelligent integration of these two types of sensors could be done to greatly reduce the overall error.

The above combination can perform quite well for localization in open areas, however, it often fails in urban sites. Tall buildings in the vicinity tend to obstruct the clear view to the satellites and the signals of fewer satellites reach the receiver (Figure 3.3, left). The signal-to-noise ratio could be attenuated by trees or large structures standing in the way. Very difficult to deal with are signal reflections and multipath (Figure 3.3, right). The result is unstable, wrong, or even no position fixes at all in some areas. In such areas, additional sensors are needed.

Due to the nature of urban sites and the overall goal of AVENUE, it is mostly around buildings that degradation in GPS performance is likely to occur.

This knowledge can be utilized by exploiting typical urban characteristics, such as abundance of linear features, parallel lines, and horizontal and vertical principal directions. These are properties that are easily captured by a camera and, thus, a CCD camera is added to make up for the limitations of the above sensors. The camera is mounted on a pan-tilt unit (PTU) which provides two independent of the robot degrees of freedom for its orientation.

I would like to point out that range finders could also be a good choice for complementing GPS and proprioceptive sensors, since they also capture many of the typical urban features. Most range finders, however, demand a tradeoff between speed, accuracy, and range. Additional non-trivial problems are caused by manufacturers who refuse to release information on how to control a scanner remotely. We have settled on a model featuring high range, high accuracy and low noise characteristics and have sought a solution to the localization problem with a camera. Since the localization method is general and cameras are cheaper and more widespread, this makes it more applicable to other situations.

The resulting set — odometry, a digital-compass with a roll-pitch sensor, a GPS unit and a color CCD camera mounted on a pan-tilt unit — forms a well-chosen and sufficient sensor suite for localization in urban environments.

3.2.2 Other Hardware Considerations

A considerable amount of work was needed to physically install and connect the sensors and the various additional periphery to the robot. Additional effort was spent on helping set up the GPS and wireless network infrastructures on the Columbia Morningside Campus. An RTK GPS base station was installed on the roof of the

Mudd building to transmit RTK corrections to the rover. Wireless access points were installed and configured to provide outdoor network coverage.

The sensor placement decision was mainly affected by the size and the weight of the laser range finder. It consists of two parts: a scanning box and a controller electronics box. Given its size, two placement options existed: to mount it on a trailer behind the robot or to attach it to a custom-designed extension of the vehicle.

Placing the scanner on a trailer has the benefit of not adding its entire weight to the burden of the robot motors. However, the existence of a non-rigid link between the robot and the trailer entails non-trivial modifications to both the path planning and the motion control algorithms: The path planner needs to take into account a non-zero turning radius factor in narrow passages and a fine-grained control of the scanner orientation via the non-rigid link becomes a serious challenge. An additional problem arises with keeping track of the relative pose of the two rigid bodies.

Instead, I have designed and built an aluminum item support frame that I rigidly affixed to the robot chassis (Figure 3.2). The range finder was mounted on the frame such that the resulting center of mass is on the same vertical line as the center of odometry. This eliminated the problems with the introduction of a non-rigid link. The rest of the sensors, were placed under the obvious constraints that the GPS antenna needs to be high and the camera needs to face the direction in which the robot is moving.

Communication with the robot is done through a wireless network. Two $11\,Mb/s$ wireless IEEE 802.11 interfaces have been added. The first one was de-

signed to take advantage of a wireless network infrastructure, such as the one installed throughout Columbia Morningside Campus. This has the benefits that the robot can distribute its computational work across multiple machines and can be monitored by multiple users at different places. Since network coverage is not always guaranteed, the second wireless interface is configured to work in an "ad hoc" mode in which it will connect directly to a portable computer without the need for installed network infrastructure.

3.3 System Architecture

3.3.1 Background

The problem of designing a software architecture for mobile robots is complex and multifaceted. It has tradionally been addressed by variations of two main approaches which are sometimes refered to as *horizontal* and *vertical*.

The horizontal approach, also known as the sense-plan-act paradigm, decomposes the robot control software into a horizontal sequence of vertical slices (modules) according to the function they perform. Typically, roboticists have sliced the problem into some subset of sensing, mapping sensors data into a world representation, planning, task execution and control. Examples include works of Crowley [Crowley, 1985], Moravec [Moravec, 1983], and Nilsson [Nilsson, 1984].

The vertical approach was first introduced by Brooks in 1986 [Brooks, 1986] with his subsumption architecture. The idea is to decompose the problem vertically into horizontal slices based on task achieving behaviors. He defined eight levels of robot competence, starting with avoiding contact with objects at level 0 and

expanding up to level 7 which is reasoning about the behavior of other objects in the world and modifying plans accordingly. Each successive level includes as a subset the previous ones.

Other approaches also exist, the most popular of which is probably the one introduced by Arkin [Arkin, 1987]. He draws from extensive knowledge of human psychology, neurology and brain theory along with robotics experience to create a software model of the action-perception cycle. The model is based on perceptual and motor schemas which may be combined in various ways and which collectively determine the overall behavior of the system.

So far, no single approach has proved universal. The behavior-based decomposition has helped robots achieve certain skills (e.g. dealing with moving objects) that have been traditionally hard in the sense-plan-act framework. However, this decomposition is difficult to implement for more complex behaviors than simple obstacle avoidance or road following. Because of this, and due to the vast differences in scope and focus of the research projects, many roboticists have opted for their own specialized architectures that serve the specific purpose better [Simmons, 1994, Sukkarieh *et al.*, 1998].

3.3.2 Issues

The first step to designing a mobile robot architecture is to identify and prioritize the key issues that need to be addressed. For the tasks of mobile robot navigation and urban site modeling, the following considerations are very important and need to be taken into account:

- **Distribution of computation**

  A major problem when designing a mobile robot architecture is the distribution of computation. As with battery power, payload, communication range, and others, computational resources are limited and usually insufficient. This is especially true when it comes to the processing of images or large-scale environmental models. It is impractical for a robot to spend its limitted processing power on application-level work when it is needed on the system level. For computationally heavy applications, it might make more sense to delegate the large computational tasks to off-board machines. This would be a highly scalable solution as one could take advantage of the recent progress in clustering technologies.

- **Communication link independence**

  If a robot is to be truly autonomous, it needs to be able to carry on the execution of its tasks regardless of whether a communication link is available to the control stations or other software or user agents. This should hold for both planned loss of communication (passing through a region known to be out of range) or unexpected loss of communication (signal jamming, excessive noise). Both the robot and its peer should handle communication loss gracefully and resume their dialog after communication is reestablished. This requirement is very often in conflict with the goal of distributing the computation as any distributed system relies on the communication link.

- **Flexibility**

  The architecture should be able to easily adapt to serve different applications or different implementations of the core system functionality. This is

especially important for research projects in their initial phase, as they frequently involve experimentation with a number of different prototypes before stabilizing the development.

- **Extensibility**

  The ease of adding new functionality to an already existing system is essential for the viability of that system. This, again, is very imporatant for research projects since they normally start with a minimal functionality and expand as the project makes progress.

- **Remote monitoring and control**

  The architecture has to allow for convenient monitoring and control of the robot from a remote control station. This is important for several reasons: Being able to monitor the progress of the mission gives you the ability to plan for tasks that depend on it. If the robot stalls or gets confused for some reason, a certain needed functionality is not yet implemented, or the results of a particular task are found to be unsatisfactory, the user should be able to manually intervene and help by giving information, repeating a task with different parameters or manually executing the missing/failing function. Further, the ease of troubleshooting and debugging is a necessity for large and complex projects.

- **Data storage and utilization**

  Of the limitted on-board resources, storage capacity is probably the least relevant issue given the prices and capacities of state-of-the-art storage devices. However, in distributed settings, access to the data may not be a trivial prob-

lem because the data may be acquired on one host and needed on another. When the possibility of a lack of permanent communication link between the two is factored in, this becomes a serious consideration and has to be addressed appropriately.

3.3.3 Design

I have designed the distributed modular architecture shown in Figure 3.4. Its main building blocks are concurrently executing distributed software components. Each component is a software abstraction of a hardware device or a specific functionality. For example, *Odo* is an abstraction of the robot odometric device, *Drive* is an abstraction of the robot actuators, and *Localizer* computes the robot pose.

Components can communicate (via IPC — Interprocess Communication) with one another within the same process, across processes and even across physical hosts. The main communication channels for data exchange are called *streams*. A data stream is started by a component, called a *source*, which generates the data or reads it from a hardware device. Streams end with a *sink* — a component which is the final recipient of the data and usually sends it directly to an actuator. Interested components can supply data to a sink or register with a source to receive updates every time they become available. A component may also provide an additional interface with a set of specific commands that it can understand and execute.

Components performing related tasks are grouped into servers. A server is a multi-threaded program that handles an entire aspect of the system, such as navigation control or robot interfacing. Each server has a well-defined interface that allows clients to send commands, check its status or obtain data.

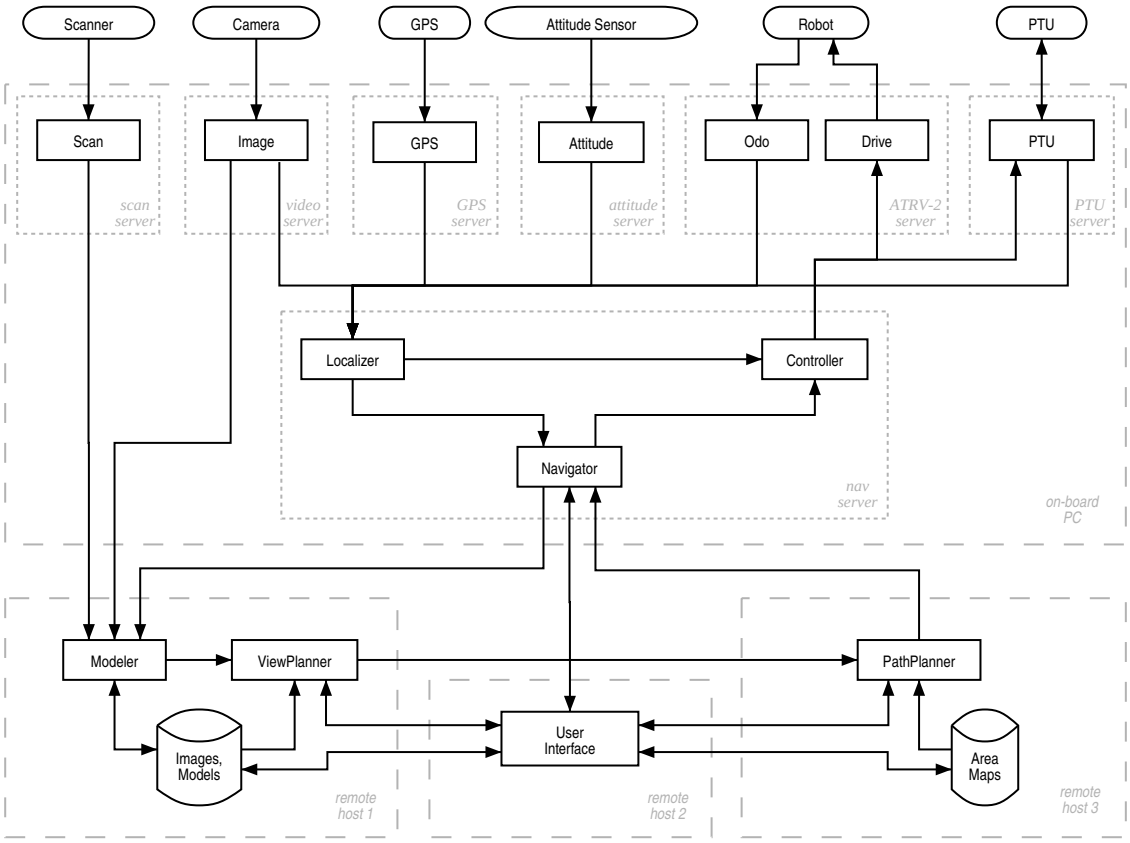


Figure 3.4: The system architecture. Solid rectangles represent components, dotted rectangles are processes, and dashed rectangles group processes running on the same machine. The arrows show the data flow between components.

To ensure maximum flexibility, each hardware device is controlled by its own server. The hardware servers are usually simple and serve three purposes:

1. Insulate the other components from the low-level hardware details, such as interface and measurement units.

2. Provide multiple, including user-defined, views of the data coming from the device. For example, a server may distribute its position data with respect to different coordinate systems.

3. Control the volume of data flow; for example, the rate at which images will be grabbed.

The hardware is accessed and controlled by seven servers, that perform some or all of the tasks above (Figure 3.4). The *NavServer* (beneath the hardware servers) builds on top of the hardware servers and provides a higher-level interface to the robot. A set of more intuitive commands, such as "go there", "establish a local coordinate system here", and "execute this trajectory", are composed out of the low-level hardware control instructions or data. The server also provides feedback on the progress of the current tasks. It consists of three components:

1. The *Localizer* is the part of the system that is the main focus of the next two chapters. It reads the streams coming from the odometry, the attitude sensor, the GPS, and the camera, registers their data with respect to the same coordinate system, and produces an overall estimate of the robot pose and velocity according to the methods described in the following chapters.

2. The *Controller* is a motion control component that brings the robot to a desired pose. It executes commands of the type **goto** and **turnto**. Based on its target and the updates from the *Localizer*, it produces pairs of desired rotational and angular velocities that it feeds to the *Drive* component of the *ATRV2Server*.

3. The *Navigator* monitors the work of the *Localizer* and the *Controller*, and handles most of the communication with the user interface and other remote components. It accepts commands for execution and reports the overall progress of the mission. It is optimized for network traffic: it filters out

the unimportant information from the low-level components and provides a compact view of the current system state to registered remote components.

A mission consists of commands that are carried out sequentially. The *Navigator* itself does not execute most of the commands — it simply stores them and resends them to the *Controller*, one at a time. It monitors the progress of the current command and, if it completes successfully, starts the next one. Additionally, a small group of emergency commands exists, such as **stop, pause**, and **resume**, that are processed immediately.

The commands stored in the *Navigator* are accessible to other components. This is useful in two ways. First, it allows users who have just connected to the robot to see what it is trying to achieve and how much it has accomplished. Second, it allows the robot to continue its mission, even if the network connectivity is temporarily lost. Moreover, this is the only way to accomplish a mission that requires passing through a region not covered by the wireless network.

The mission command sequence is usually composed by the *Path Planner* which converts the target sensor acquisition pose obtained by the *View Planner* into a sequence of navigation and data acquisition instructions according to a 2-D map of the area. The *View Planner* generates a new target after the *Modeler* updates the model with new scans and images reported by the *ScanServer*.

The architecture presented here addresses the issues pointed out in section 3.3.2. The computation needs to be distributed because of the heavy requirements of the modeling application. The underlying framework and the wireless network interface make it possible for a server to run on a computer not physically residing on the robot.

The distributed nature of the system along with the communication link independence requirement brings an interesting question of how to resolve the conflict between distributing the computational load and the need for autonomy. Obviously, some components, which I call *core components*, need to be run on-board as they are hardware related (such as interfaces to sensors or actuators) or mission-critical (for example, the low-level control system). Components, which are too computationally intense (*heavy components*) need to be run on other machines.

Normally, the core components are lightweight in that they do not consume a lot of resources and can all be run on-board. This is the case with the architecture presented here. Core servers are the ones shown in the upper portion of the block diagram. The top row consists of hardware-related servers, while the *NavServer* implements the control system and is, therefore, mission-critical.

The heavy components are the *Modeler* and the *View Planner*, however, they are not critical for the mission and do not pose problems running remotely. Acquired data is supplied to them when a network connection becomes available and once a possible resulting new mission is uploaded to the *Navigator*, the connection is no longer needed.

Note that in the general case, it is possible for a component to be both core and heavy, that is, it is essential for the mission but is more computationally intense that the on-board machine can handle. In such cases, I see the solution in a variation of Brooks' subsumption architecture. Two versions of that component will be provided: A heavy version implementing the full computationally intense functionality will be run remotely and a stripped-down version will be run locally on-board. When connection is available, the result of the heavy version will be

used, overriding the local one. If connection is lost, the local version will act as an emergency substitute.

The standardized way of communication between components makes servers easily reconfigurable and replaceable and provides high flexibility. For example, when I want to test a particular behavior of the control system indoors (where GPS data, of course, is not available), all I need to do is run a program *GPSSimulator* instead of the *GPSServer*. Similarly, one can test the navigation system on a *Pioneer I* robot by simply running a *PioneerServer* instead of the *ATRV2Server*.

For the same reasons, the architecture is also highly extensible. Adding new functionality on the application level could be as simple as writing another software component which connects to existing ones. On the system level, the addition of a new sensor, for example, might require writing a new hardware server to interface that sensor and updating the *Localizer* to integrate the sensor with the others.

Due to the distributed nature of the system, the status of the robot and each of its subsystems can be accessed and controlled remotely. This can be done either autonomously by a client program or by the user via the user interface described further in this chapter.

The architecture implementation is based on the Common Object Request Broker Architecture (CORBA), which provides to a great degree platform, operating system, and programming language independence. This important practical aspect has been demonstrated by running various pieces of AVENUE on both conventional personal computers and portable digital assistants under different operating systems, such as Linux and Windows CE. A description of the implementation details of the architecture is presented in Appendix A.

3.4 Motion Control

To provide a fully functional implementation of the software architecture, I have written a motion control component. The task of the component is to position the robot at a given target location or orientation. Each time the *Localizer* produces a new estimate, or a new target is supplied by the *Navigator*, the *Controller* updates its output to the *Drive* component. The controlled quantities are the translational velocity v and the angular velocity ω.

Since the kinematics of the robot is essentially two-dimensional, the control is done in 2-D. The reference plane for this is the O-x-y plane of the robot coordinate system. I have adopted a standard robot-fixed coordinate frame. It is centered at the center of odometry at ground level. The x axis points forward, the y axis points to the left, and the z axis points up (Figure 3.5).

The target is expressed in 3-D but due to the 2-D constraints, only its projection onto the reference plane is pursued. Thus, if the target is a new position $\mathbf{p} = [x, y, z]^T$, its Cartesian coordinates $\mathbf{p'} = [x', y', z']^T$ with respect to the robot-fixed frame are computed, using the current robot orientation $\mathbf{R_r}$ (a 3 x 3 matrix) and location $\mathbf{T_r}$ (a 3 x 1 vector):

$$\mathbf{p'} = \mathbf{R_r^T}(\mathbf{p} - \mathbf{T_r}). \tag{3.1}$$

Similarly, if the target is a new orientation expressed as the 3 x 3 matrix $\mathbf{R}$, then a vector pointing in that direction is considered relative to the robot frame:

$$\mathbf{p'} = \mathbf{R_r^T R} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}. \tag{3.2}$$

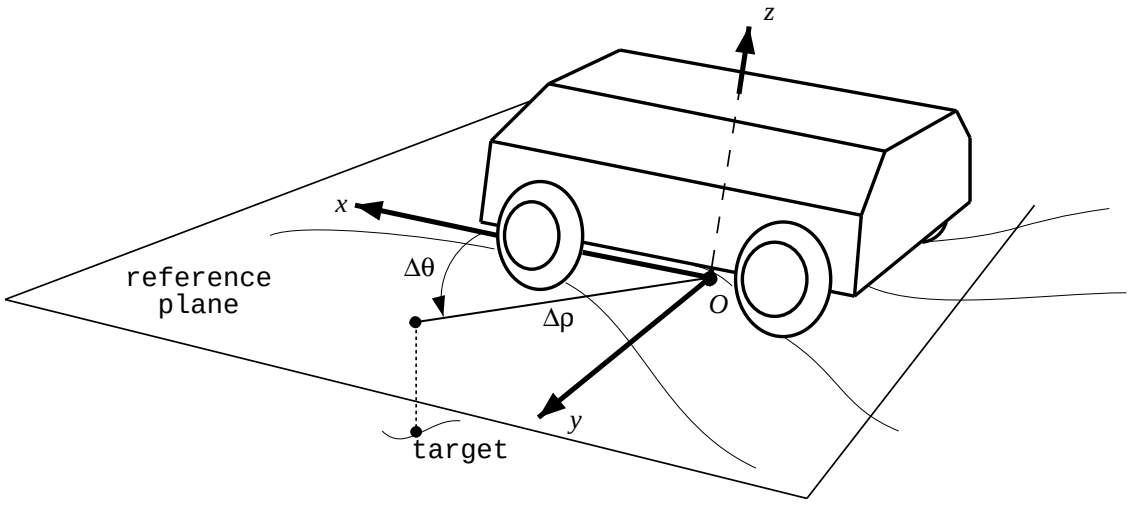


Figure 3.5: The robot reference frame and a control target point.

After that, the target is projected on the reference plane by dropping the z coordinate from consideration and the result is expressed in polar coordinates:

$$\Delta\rho = \sqrt{x'^2 + y'^2} \tag{3.3}$$

$$\Delta\theta = \arctan\left(\frac{y'}{x'}\right). \tag{3.4}$$

The *Controller* implements the proportional control law. The polar coordinates $\Delta\rho$ and $\Delta\theta$, which are current translational and rotational displacements from the target, are multiplied by a pair of experimentally determined gains to obtain the new velocities:

$$\nu_{new} = k_\nu \, \Delta\rho \tag{3.5}$$

$$\omega_{new} = k_\omega \, \Delta\theta. \tag{3.6}$$

Before applying these velocities certain limits are imposed. First, high accelerations are suppressed:

$$\nu'_{new} = \mathrm{sgn}(\nu_{new}) \, \min\{|\nu_{new}|, \, |\nu| + a_{max}\Delta t\} \tag{3.7}$$

$$\omega'_{new} = \text{sgn}(\omega_{new}) \min\{|\omega_{new}|, |\omega| + \alpha_{max}\Delta t\}. \quad (3.8)$$

where a_{max} and α_{max} are the maximum translational and rotational accelerations allowed, and Δt is the sampling period.

Next, velocities are restricted within an acceptable interval:

$$\nu''_{new} = \text{sgn}(\nu'_{new}) \min\{|\nu'_{new}|, \nu_{max}\} \quad (3.9)$$

$$\omega''_{new} = \text{sgn}(\omega'_{new}) \min\{|\omega'_{new}|, \omega_{max}\}. \quad (3.10)$$

Finally, the resulting velocities are sent to the *Drive* component and the control cycle repeats.

3.5 User Interface

The user may monitor and control the robot at any time via the *User Interface* (Figure 3.6) from a remote machine. Connection to the robot is established via its wireless networking interface.

The user interface provides a comprehensive view of the robot location and activities within its environment. The main window shows a perspective view of the objects and the environment. The pose of the virtual camera can be controlled by the user. Fixed virtual camera positions can be defined, including ones that are referenced with respect to the robot and change as it moves.

The user interface contains software components that connect to the *Navigator* and keep track of its status along with the data coming from the *Localizer* and the state of the *Controller*. The robot is shown on the screen at the position reported by the *Localizer*. As it moves, it leaves a trail on the screen to illustrate its most recent trajectory.

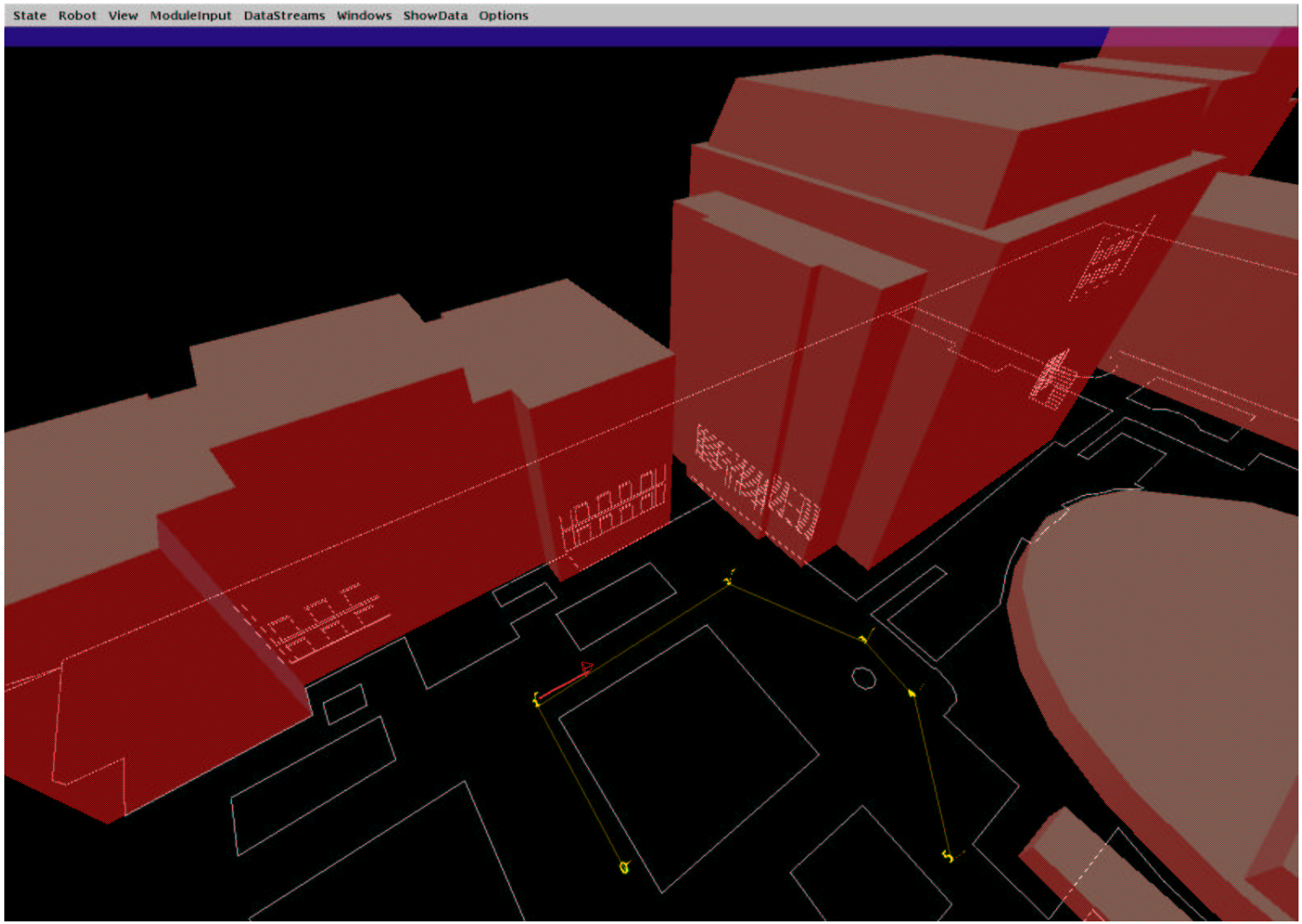


Figure 3.6: The user interface. The robot is shown as a red pyramid leaving a trail behind it as it moves. The planned trajectory is shown in yellow with each target marked by a flag and a number. The white lines outline obstacles from the 2-D map used by the path planner. Simple 3-D models of the buildings are shown semitransparent with localization models overlaid on their facades.

The user interface interacts with the planning components to allow the user to exercise control and manually create or modify existing missions. A mission is illustrated by the desired trajectory of the robot drawn on the screen with each important target marked by a flag. A separate command list editor is available to allow manual editing of the commands each target or the list of targets itself.

The environment may be presented in various independent ways. A two-dimensional map outlines the obstacles at the site. The detailed 3-D models of the buildings can be shown on top of the map. The models used for visual localization can be also shown or hidden.

The user can simply monitor the progress of the mission or take manual control and teleoperate the robot. The interface provides a "visual joystick" type of direct control of the robot. This is typically used with the visual feedback from the camera or the sonars.

The user interface is highly modular, portable and extensible. It provides a standardized module programmer's interface through which additional modules can be automatically detected using introspection techniques. Debugging and troubleshooting are easy, as the user interface allows access to the debug information generated by remote components. Multiple robots are supported.

The graphical user interface was designed and written in cooperation with Ethan Gold.

3.6 Summary

I have designed and implemented a functioning autonomous mobile robot platform for urban site modeling which is being used as the prototype platform for the AV-

ENUE project. The design extended an existing robotic vehicle with a carefully chosen sensor suite: a digital compass with an integrated inclinometer, a global positioning unit, and a camera mounted on a pan-tilt head. A large support frame was built to accommodate the size of the range finder. Necessary wireless networking hardware was also installed on the robot.

I have also designed a distributed modular software architecture for mobile robot localization and navigation. This architecture has been demonstrated to be computer platform and operating system independent. Proper attention has been paid to important issues such as the distribution of computation and wireless network coverage.

I have implemented the essential part of the software architecture including:

- software components to interface with the installed sensors and other hardware, convert hardware-specific data to an easily used format and make it available to the rest of the system
- a robot motion control component to drive the robot
- a software component that serves as an interface to the robot from external machines
- a graphical user interface to control and monitor the robot remotely (joint work with Ethan Gold)

Chapter 4

Localization in Open Space

This chapter describes the part of the localization system that uses the built-in wheel encoders and the installed attitude sensor and GPS infrastructure. This subsystem provides updates of the robot pose in 3-D together with an estimate of the uncertainty in those updates.

It is a fully functional localization system on its own, i.e it does not depend on the visual component to work properly. In fact, this is the main localization component, in the sense that it is used in a stand-alone mode during most of the time — whenever the robot traverses open areas.

The sections that follow describe the building of the subsystem from the ground up, with the addition of each sensor one at a time.

4.1 Odometry

Odometry is probably the most widely used localization method. This is mainly because it comes integrated in almost every wheeled mobile robot, but also because

it is very simple to use and is guaranteed to provide pose estimates at all times.

There are multiple implementation variants but the principle of operation is common and quite simple: A sensor is placed on each wheel (or shaft) of the robot which periodically measures the amount of rotation or the rotational velocity of the wheel. Based on this information and the parameters of the robot (wheel radii, wheel baseline, etc), it is easy to compute the robot translational and rotational displacements and velocities.

The ATRV-2 robot has two independently controlled wheels on each side in a differential-drive skid-steering configuration (Figure 4.1). The robot moves forward/backward by driving all four wheels in the appropriate direction. It turns by driving the wheels on one side forward and the wheels on the other side backward. It is also capable of making gradual turns by controlling the speeds of the individual wheels. The center of odometry is the center of the rectangle formed by the four wheels. As is standard for mobile robots, I have adopted a body-fixed coordinate frame, whose origin is the center of odometry with the x axis facing forward, y axis perpendicular to the left, and z axis pointing up.

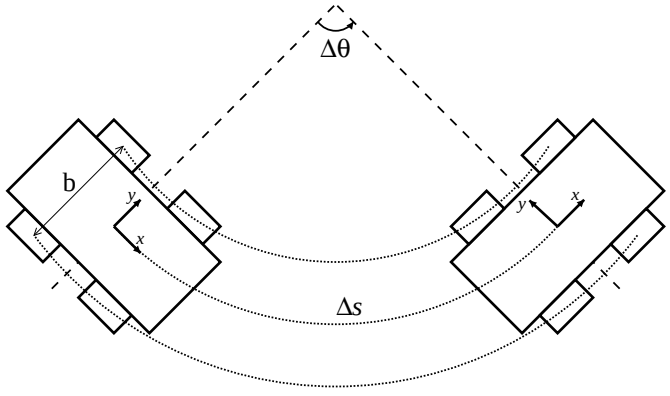


Figure 4.1: The ATRV kinematics

The robot odometry is based on encoders placed in the motors that generate and count pulses as the motors spin. The firmware samples these counts at 10-15 Hz. Each such pulse count represents the amount of rotation of the corresponding wheel during the sampling period. If we denote the wheel baseline by b, the wheel radius by r and the average angular displacements of the left and right wheels by $\Delta\varphi_l$ and $\Delta\varphi_r$ respectively, then the total robot angular displacement and travel distance during the sampling interval are

$$\Delta\theta = \frac{\Delta\varphi_r - \Delta\varphi_l}{b}\, r \tag{4.1}$$

$$\Delta s = \frac{\Delta\varphi_r + \Delta\varphi_l}{2}\, r. \tag{4.2}$$

The robot pose $\mathbf{x} = [x, y, \theta]^T$ is then computed according to

$$\theta_k = \theta_0 + \sum_{i=0}^{k} \Delta\theta_k \tag{4.3}$$

$$\mathbf{p_k} = \mathbf{p_0} + \sum_{i=0}^{k} \Delta s_i \, Rot(\theta_i)\, \mathbf{e_x}, \tag{4.4}$$

which is a discretized and linearized approximation of the general continuous model

$$\theta(t) = \theta(0) + \int_0^t \omega(\tau)\, d\tau \tag{4.5}$$

$$\mathbf{p(t)} = \mathbf{p(0)} + \int_0^t v(\tau)\, Rot[\theta(\tau)]\, \mathbf{e_x}\, d\tau. \tag{4.6}$$

Here, $\mathbf{p(t)} = [x(t), y(t)]^T$, $\theta(t)$, $v(t)$ and $\omega(t)$ are the robot position, orientation, linear velocity and angular velocity at time t. The suffixes in $\mathbf{p_k}$ and θ_k denote values at time $t_k = t_0 + k\,\Delta t$. $Rot(\varphi)$ is the $2\,\mathrm{x}\,2$ rotation matrix by an angle φ, and $\mathbf{e_x} = [1, 0]^T$.

The above equations reveal two major problems with using odometry as a localization tool. First, they obviously assume a planar motion of the robot and thus

estimate the pose only in 2-D. It is not possible to compute a 3-D estimate based solely on rotational and translational velocities as there is not sufficient information to separate horizontal from vertical motion. To do that, additional sensors are needed, such as the digital compass module described in the next section.

The second fundamental problem with odometry is apparent from equations (4.5) and (4.6). The values of $\omega(\tau)$ and $v(\tau)$ can never be computed exactly because of discrete sampling and measurement errors. For example, in equation (4.3), due to discretization of (4.5), $\omega(t)$ is approximated by the average rotational velocity $\omega_k = \Delta\theta_k/\Delta t$ for each interval $t \in [t_k, t_{k+1})$. In general, minute errors $\delta\omega(\tau)$ introduced in $\omega(\tau)$ will cause the computation of an approximation of $\theta(t)$

$$\tilde{\theta}(t) = \theta(0) + \int_0^t [\omega(\tau) + \delta\omega(\tau)]\, d\tau = \theta(t) + \int_0^t \delta\omega(\tau)\, d\tau. \tag{4.7}$$

No matter how small $\delta\omega(\tau)$ is, it accumulates as the robot moves and the resulting error in orientation

$$\delta\theta(\tau) = \tilde{\theta}(t) - \theta(t) = \int_0^t \delta\omega(\tau)\, d\tau \tag{4.8}$$

grows unbounded. Worse, $\delta\theta(\tau)$ is integrated once more in equation (4.6) resulting in even larger error for the position.

Note that this is a fundamental problem with the odometry localization model in general — it is not particular to the discretized version (4.3)–(4.4), although it is certainly exaggerated by it. In fact, the unbounded error accumulation is an inherent problem to all proprioceptive sensors. Typically, a robot can accurately traverse a few meters but for longer distances relying on such sensors becomes impractical. The error can not be bounded without the use of an exteroceptive sensor (which in my case is the GPS unit, described later in this chapter).

It is possible, nonetheless, to address this problem by looking at the sources of the errors. Borenstein and Feng have classified the errors in two categories: systematic and non-systematic [Borenstein and Feng, 1995]. Systematic errors are defined as ones that are directly caused by the kinematic imperfections of the vehicle, while the rest of them are classified as non-systematic. Examples for systematic errors are ones caused by mismodeling of the wheel baseline, the wheel radii, the gear transmission ratio, and others. Examples for causes of non-systematic errors include slippery spots, uneven terrain, surface irregularities and over-acceleration. Both systematic and non-systematic errors can be addressed.

4.1.1 Systematic Errors

According to Borenstein and Feng, two major sources of systematic errors in a differential-drive robot are unequal wheel radii and a misestimated wheel baseline. Both can be estimated and compensated for by accurate calibration. Borenstein and Feng devised a method called *the UMBmark* to do so [Borenstein and Feng, 1995], however, it assumes certain preliminary calibration. Below, I describe my adaptation of the UMBmark method to the peculiarities of the ATRV-2 kinematics and its extension into a complete calibration procedure.

As I mentioned earlier, the wheel rotations are measured with the help of encoders mounted on the drive motors. During each short sampling interval, the encoders show a pulse increment corresponding to the amount of wheel rotation. These pulse increments are converted to angular displacements by the following formula:

$$\Delta\varphi = \frac{2\pi}{nC_e}N = c_m N \tag{4.9}$$

where

$\Delta\varphi$ - incremental rotation of the wheel

n - gear ratio of the reduction gear between the motor and the drive wheel

C_e - encoder resolution in pulses per revolution

N - measured incremental pulse count

c_m - conversion factor that translates encoder pulses into angular wheel displacement.

The pair of measurements for the wheels on each side of the robot are averaged and used to compute the robot rotational displacement and distance traveled increment according to (4.1) and (4.2). If we take into account that the wheels may have different radii, these equations become

$$\Delta\theta = \frac{\Delta\varphi_r\, r_r - \Delta\varphi_l\, r_l}{b} = c_m\, \frac{r_r N_r - r_l N_l}{b} \tag{4.10}$$

$$\Delta s = \frac{\Delta\varphi_r\, r_r + \Delta\varphi_l\, r_l}{2} = c_m\, \frac{r_r N_r + r_l N_l}{2}, \tag{4.11}$$

where the suffix r denotes the average of the values for the front and the rear right wheels and the suffix l denotes the average for the two left wheels.

The UMBmark procedure helps derive two coefficients related to errors in the orientation of the robot: $E_d = r_r/r_l$, which accounts for the difference in the wheel radii, and $E_b = b/b_0$, which is the ratio between the actual wheel baseline b and the nominal one b_0 reported by the manufacturer. The robot is directed to traverse a square of size L x L, n times clockwise and n more times counterclockwise. Ideally, the robot should return to its original pose but in practice it is slightly off. The arriving locations are recorded and the errors between actual locations and the

ones reported by the odometry are averaged to produce the centers of two clusters: $[\bar{x}_{cw}, \bar{y}_{cw}]^T$ for the clockwise runs, and $[\bar{x}_{ccw}, \bar{y}_{ccw}]^T$ for the counterclockwise runs. Then, the coefficients E_b and E_d are computed by

$$\alpha = \frac{\bar{x}_{cw} + \bar{x}_{ccw}}{-4L}, \quad \beta = \frac{\bar{x}_{cw} - \bar{x}_{ccw}}{-4L}, \quad R = \frac{L/2}{\sin(\beta/2)} \tag{4.12}$$

$$E_b = \frac{\pi/2}{\pi/2 - \alpha}, \quad E_d = \frac{R + b_0/2}{R - b_0/2} \tag{4.13}$$

Note that the procedure can not recover r_r, r_l and b directly unless the nominal wheelbase and at least one of r_r and r_l are known. The procedure also does not correct for translational errors. In fact, it requires that the odometry has been accurately calibrated for translation, since otherwise the actual traversed distance by the robot will be significantly different than $4L$ which is assumed in the derivations of α, β, and R. Thus, I had to devise a preliminary calibration procedure to obtain the necessary information.

This is done in two steps: one that estimates the translation coefficient $c_T = c_m r_l$, and another that recovers the rotation coefficient $c_R = c_m r_l / b$. The computation in both steps requires the ratio E_d of the wheel radii to be known. In the beginning, however, it is not because it is the UMBmark that computes it. This is not a problem: the procedure can be bootstrapped with a good initial estimate and then the value refined in the next iteration. In almost all cases, a good initial choice for E_d is to set it to one as the wheel radii are most likely to be the same.

First, the translation conversion coefficient is determined by having the robot move forward by a certain distance (I used $20\,m$) and the actual distance traveled is measured. Denote the latter by L_f. Then:

$$L_f = \Delta s = c_m \frac{r_r N_r + r_l N_l}{2} = c_m \frac{E_d r_l N_r + r_l N_l}{2} \quad \Rightarrow \quad c_T = \frac{2L_f}{E_d N_r + N_l}. \tag{4.14}$$

Next, the rotational conversion coefficient is determined by using pure rotation. The robot is manually controlled (via a joystick or a simple program) to make m (I used $m = 5$) revolutions in place. Then:

$$2m\pi = \Delta\theta = c_m \frac{r_r N_r - r_l N_l}{b} = c_m \frac{E_d r_l N_r - r_l N_l}{b} \quad \Rightarrow \quad c_R = \frac{2m\pi}{E_d N_r - N_l}. \tag{4.15}$$

From the last two equation, we recover the true wheel baseline

$$b = \frac{c_T}{c_R}. \tag{4.16}$$

At this point, a sufficiently accurate odometry model based on the average wheel radius is available, and the UMBmark procedure can be performed. Note that even after the estimate for E_d is obtained, the values of r_r and r_l may still not be recovered. However, they are not really needed, since the following coefficients are sufficient

$$c_l = c_T \tag{4.17}$$

$$c_r = E_d\, c_T \tag{4.18}$$

and the final computation of the robot motion becomes

$$\Delta\theta = \frac{c_r N_r - c_l N_l}{b} \tag{4.19}$$

$$\Delta s = \frac{c_r N_r + c_l N_l}{2}. \tag{4.20}$$

If the result of UMBmark is not satisfactory, its estimate for E_d can be used to perform another iteration of the two preliminary steps and further refine the conversion coefficients. Generally, more than two such iterations should not be needed as the effect of the systematic errors will be reduced to negligible compared to the non-systematic ones.

4.1.2 Non-systematic Errors

Non-systematic errors can not be compensated for since they are by definition random and unpredictable. However, we can try to model their accumulation so that we keep track of the uncertainty in the robot pose estimates. This helps in the fusion of these estimates with the ones of other sensors.

This can be done formally in a probabilistic fashion. Two models are created: one of the robot kinematics and another of the robot odometry. The first one is used to estimate the error in the robot pose introduced in the execution phase by the actuators. The second model represents the error of the odometry due to inaccurate measurements, discretization, etc.

Due to uncertainty in the execution phase, the robot pose can be modeled by a conditional probability density function $P_K(\mathbf{x}'|\mathbf{x}, \mathbf{d})$ which determines the probability of the robot pose being $\mathbf{x}'$ if it was previously at $\mathbf{x}$ and executed the control input $\mathbf{d}$. This function imposes constraints on the robot pose. If there is a prior probabilistic estimate $P(\mathbf{x})$ of the pose, then the probability of the robot being at $\mathbf{x}'$ after the execution of $\mathbf{d}$ is

$$P(\mathbf{x}') = \int P_K(\mathbf{x}'|\mathbf{x}, \mathbf{d})\, P(\mathbf{x})\, d\mathbf{x} \tag{4.21}$$

The function $P_K(\mathbf{x}'|\mathbf{x}, \mathbf{d})$ can be determined by running statistical experiments for different control inputs $\mathbf{d}$ and creating a lookup table for the resulting distribution or approximating it with an analytical expression. For example, if the robot is directed to move forward by a certain distance, the result (2-D position only) might look like the one shown in Figure 4.2(top).

Grid-based localization methods typically approximate this by adding Gaus-

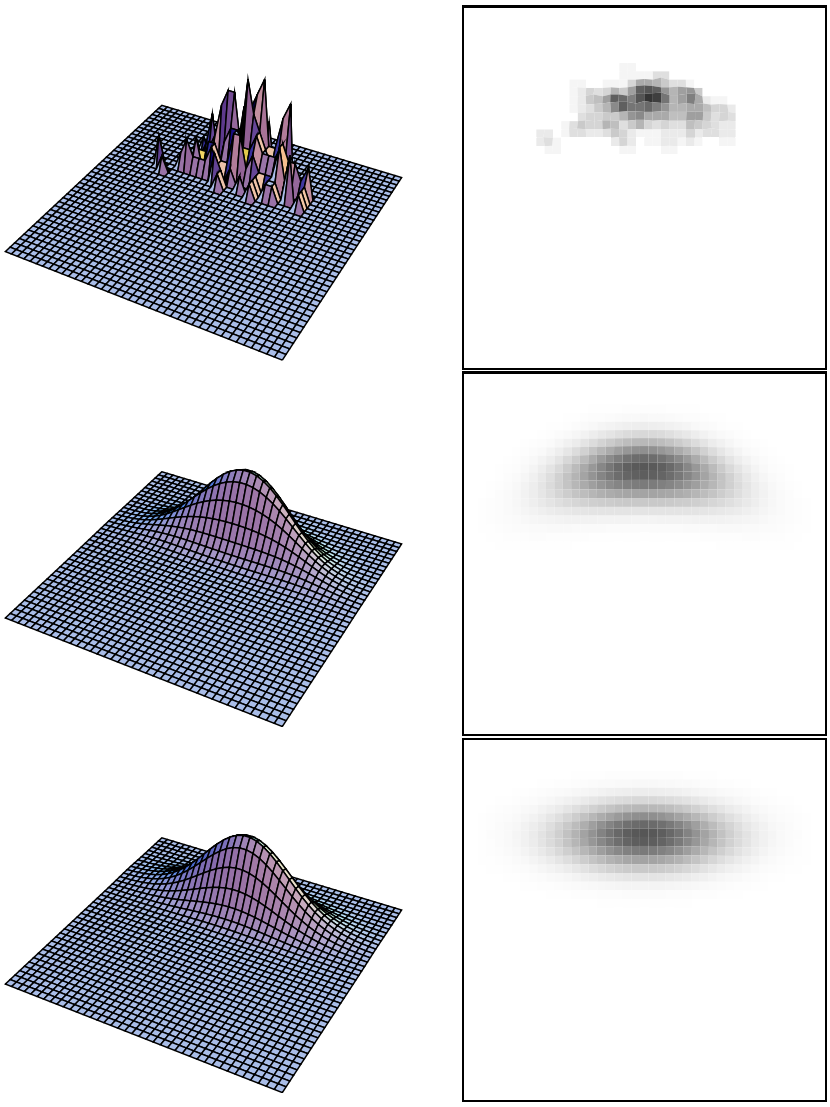


Figure 4.2: The resulting distribution of the robot position after moving forward by a certain distance. The plots on the left show a perspective 3-D view of the distribution denoted by height of the surface. The robot starts from the center of the near edge and moves away. The plots on the right show a top view of the same distribution, denoting higher probabilities with darker color. The robot starts from the center of the bottom edge and moves up: **(top row)** a typical result of a statistical experiment; **(middle row)** an accurate model; **(bottom row)** a close approximation of the model.

sian white noise to both the expected final orientation and distance traveled. The model for the example above would look as in Figure 4.2(middle). The problem with this kinematic model is that the probability density function of the robot pose becomes very difficult to represent analytically in a simple and compact form after a few iterations of (4.21). It is also a practical problem to perform the integration efficiently. These issues are addressed by discretizing the state space and representing $P(\mathbf{x})$ for each state cell explicitly which necessarily confines the applicability of this approach to a small-scale 2-D world.

In large-scale 3-D environments, $P_K(\mathbf{x}'|\mathbf{x}, \mathbf{d})$ needs a more manageable form. Thus, I have chosen another popular approximation which described as a 2-D Gaussian noise centered at the expected destination pose of the robot and with standard deviation determined as above. If the robot is moving according to the plant model $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{d}, t)$, then the pose at time t is

$$\mathbf{x}(t) = \mathbf{x}(t_0) + \int_{t_0}^{t} \mathbf{f}(\mathbf{x}, \mathbf{d}, \tau)\, d\tau + \mathbf{u}(t)$$

where $\mathbf{u}(t)$ is a zero-mean white Gaussian noise with standard deviation proportional to the distance traveled.

A clear benefit of this model is that it causes equation (4.21) to always produce a Gaussian distributed pose. Furthermore, since a Gaussian is completely characterized by its first two moments, the integral in (4.21) can be replaced by a simple matrix multiplication, as it will be shown in the next section. For comparison, Figure 4.2(bottom) shows this model under the same conditions as before.

The odometry measurements are modeled in a similar fashion producing a probability density function $P_M(\mathbf{x}'|\mathbf{x}, \mathbf{d})$ representing the probability of the odometry estimate being $\mathbf{x}'$ if the robot started at $\mathbf{x}$ and the robot actuators executed

the control input **d** precisely. This can also be estimated statistically by measuring the difference between the actual robot trajectory and the one reported by the odometry.

4.2 Attitude Sensor

In many cases, mostly for indoor applications, the assumption for a planar ground is reasonable and working with 2-D pose estimates makes sense. In a typical large-scale outdoor environment however it is more often than not necessary to utilize all three dimensions. I have added an attitude sensor to augment odometry to 3-D.

The Honeywell HMR3000 unit installed on the robot consists of a two-axis tilt module that makes absolute measurements of the roll and pitch angles of the platform and an integrated digital magnetic compass that measures the azimuth angle relative to the magnetic North. The tilt accuracy of the sensor is $\pm 0.6^\circ$ and the nominal heading error is less than 1.5° RMS.

4.2.1 Pose Estimation

The combination of an attitude sensor and odometry helps estimate the robot motion in 3-D by approximating it with a piece-wise planar trajectory. Denote by $\mathbf{p} = [x, y, z]^T$ the coordinates of the robot, $\phi = [\varphi, \theta, \psi]^T$ the vector of the three Euler angles and let $R(\phi)$ be the roll-pitch-yaw matrix

$$R(\phi) = Rot_z(\psi)\, Rot_y(\theta)\, Rot_x(\varphi), \tag{4.22}$$

where $Rot_a(\varphi)$ is the 3 x 3 rotation matrix about axis a by an angle φ. Suppose that at time t_k, the robot is at position $\mathbf{p}_k$ and the compass module provides the

attitude measurements ϕ_k. Establish a local coordinate system that coincides with the robot attitude at this time, so that a point $\mathbf{p}'$ expressed with respect to the local coordinate system is related to its world coordinates $\mathbf{p}$ by:

$$\mathbf{p} = \mathbf{p}_k + R(\phi_\mathbf{k})\,\mathbf{p}'. \tag{4.23}$$

During the interval $[t_k, t_{k+1})$, no new measurements are available and therefore the robot orientation is assumed to be changing only because of its angular velocity. Thus, the robot is assumed to be moving on a local plane, orthogonal to its current local z axis, with a constant linear velocity $v(t) = v_k$ and angular velocity $\omega(t) = \omega_k$ for $t \in [t_k, t_{k+1})$. In the local coordinate system, the motion is expressed as a trivial extension of equations (4.5) and (4.6):

$$\psi'(t) \;=\; \int_{t_k}^{t} \omega(\tau)\,d\tau \;=\; \omega_k\,(\tau - t_k) \tag{4.24}$$

$$\mathbf{p}'(t) \;=\; \int_{t_k}^{t} v(\tau)\; Rot_z[\psi'(\tau)]\,\mathbf{e_x}\,d\tau \;=\; \frac{v_k}{\omega_k}\begin{bmatrix} \sin[\omega_k(t-t_k)] \\ 1-\cos[\omega_k(t-t_k)] \\ 0 \end{bmatrix}, \tag{4.25}$$

where $\mathbf{e_x} = [1,0,0]^T$ here. Equation (4.25) has singularity at $\omega_k = 0$ which may cause numerical problems for small angular velocities. This can be worked around by using a second order Taylor's approximation for the sines and cosines in the rotation matrix:

$$\mathbf{p}'(t) \;\approx\; \begin{bmatrix} v_k(t-t_k)\cos[\omega_k(t-t_k)] - \omega_k v_k(t-t_k)^2 \sin[\omega_k(t-t_k)]/2 \\ v_k(t-t_k)\sin[\omega_k(t-t_k)] + \omega_k v_k(t-t_k)^2 \cos[\omega_k(t-t_k)]/2 \\ 0 \end{bmatrix} \tag{4.26}$$

Converting back to world coordinates, we get

$$\mathbf{p}(t) \;=\; \mathbf{p}_k + R(\phi_\mathbf{k})\,\mathbf{p}'(t) \tag{4.27}$$

$$R[\phi(t)] \;=\; R(\phi_k)\, Rot_z[w_k(t - t_k)]. \tag{4.28}$$

Equations (4.27) and (4.28) are a good approximation for the robot motion during the time interval $[t_k, t_{k+1})$. When new measurements arrive at time t_{k+1}, the robot orientation ϕ or the velocities v and ω are updated (depending on the source of the new measurements) and the process repeats.

4.2.2 Kalman Filters

The above process does not provide means for managing uncertainty and the contribution of each sensor. There is some measurement redundancy that can be exploited since, as equation (4.28) shows, odometry can also produce accurate short-term orientation estimates.

To improve on the blending of the data from the two sensors, I have employed an extended Kalman filter (EKF) framework. A benefit of this framework is that it is an optimal estimator for linear processes and a near-optimal one for non-linear processes of low-dynamics. Besides efficient estimation and uncertainty handling, this framework also provides an elegant way of adding additional sensors, such as the GPS receiver described in the next section.

As it will be shown in the next subsection, the process of the robot motion is non-linear (4.29). To utilize the EKF framework, linearization needs to take place. Before proceeding, however, I would like to point out two important decisions that have been made here.

First, a Kalman filter could be either direct or indirect. The direct Kalman filter operates on the total state of the process, which means that among its variables are the quantities that we want to estimate. The indirect formulation estimates the

error state — the differences of the quantities we are interested in from their actual values.

There are some serious drawbacks to choosing the direct formulation here. Being a linear estimator, the Kalman filter behaves well for linear systems and systems of low dynamics. Since we have a highly non-linear process, a linear model will fail to adequately represent it and will quite likely diverge. Another problem is that the filter will have troubles separating high-frequency noise from legitimate dynamic situations. Also, the amount of smoothing or low-pass filtering is directly related to the time delay of the filter output resulting in sluggish response. On the other hand, regardless of the dynamics of the process, a relatively accurate error model used in an indirect formulation will cause the errors to exhibit the low frequency behavior the filter is best suited for.

The other consideration is whether to use a linearized as opposed to an extended formulation. The linearized form uses a reference trajectory about which the linearization takes place. The error estimates from the filter are added to the reference trajectory to obtain the estimate but the reference trajectory itself is never

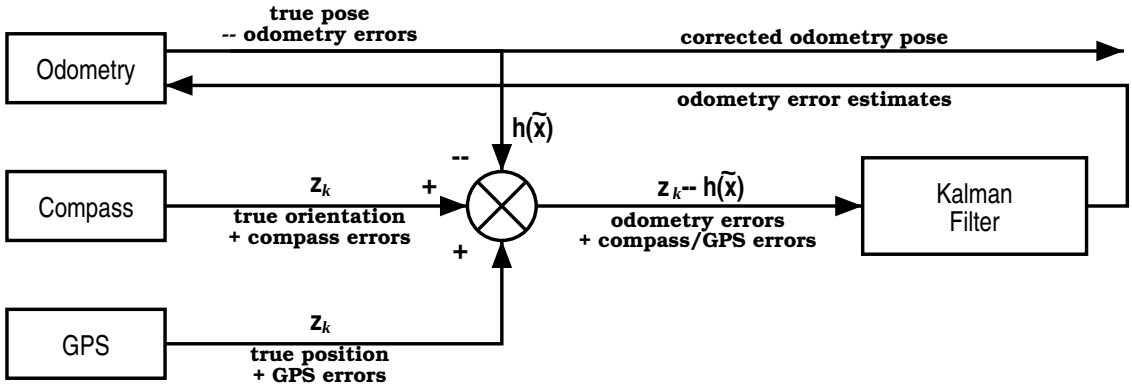


Figure 4.3: A diagram of the extended Kalman filter configuration

updated. This is known as feedforward configuration. Conversely, in a feedback configuration, the reference trajectory gets updated with each iteration of the filter and the next linearization takes place about the updated trajectory. Both versions have their strengths. In this case, however, there is no good choice of a reference trajectory, if the linearized formulation is adopted. A bad choice would most likely diverge from the actual one beyond acceptable limits. Thus, the extended version is more appropriate. A diagram of the filter configuration I have implemented is shown in Figure 4.3.

4.2.3 Integration with Odometry and Uncertainty Handling

The state vector of the process of the robot motion consists of the robot pose $\mathbf{x} = [\mathbf{p},\ \phi]^T$ which is a 6-vector of the three coordinates x, y, z and the Euler angles φ, θ, ψ. The control input to the robot is $\mathbf{d} = [v, \omega]^T$. The process differential equations are:

$$\dot{\mathbf{x}} \ = \ \mathbf{f}(\mathbf{x}, \mathbf{d}, t) \ = \ \begin{bmatrix} \mathbf{f}_p(\phi, v) \\ \mathbf{f}_\phi(\phi, \omega) \end{bmatrix} = \begin{bmatrix} v\, R(\phi)\, \mathbf{e}_x + \mathbf{u}_p \\ \omega\, M(\phi)\, \mathbf{e}_z + \mathbf{u}_\phi \end{bmatrix} \tag{4.29}$$

$$M(\phi) \ = \ \begin{bmatrix} 1 & \sin\varphi\, \tan\theta & \cos\varphi\, \tan\theta \\ 0 & \cos\varphi & -\sin\varphi \\ 0 & \sin\varphi\, \sec\theta & \cos\varphi\, \sec\theta \end{bmatrix} \tag{4.30}$$

The matrix $M(\phi)$ transforms the 3-D angular rate to the Euler angle time derivatives. The rotation, as before, is of magnitude ω about the robot z axis and $\mathbf{e}_z = [0, 0, 1]^T$. The vectors $\mathbf{u}_p$ and $\mathbf{u}_\phi$ are added zero-mean Gaussian white noise due to effects like discretization and system misrepresentation.

I have chosen to linearize about the estimated trajectory from odometry.

This is a good choice, since odometry is sampled very frequently and usually drifts slowly with the distance traveled. Denote by $\tilde{\mathbf{x}} = [\tilde{\mathbf{p}},\ \tilde{\phi}]^T$ the odometry pose estimate. For convenience, the errors will be defined as

$$\delta\mathbf{x} = \mathbf{x} - \tilde{\mathbf{x}} \tag{4.31}$$

$$\delta v = v - \tilde{v} \tag{4.32}$$

$$\delta\omega = \omega - \tilde{\omega}. \tag{4.33}$$

Then

$$\delta\dot{\mathbf{p}} = \dot{\mathbf{p}} - \dot{\tilde{\mathbf{p}}} = \mathbf{f_p}(\tilde{\phi} + \delta\phi, \tilde{v} + \delta v) - \mathbf{f_p}(\tilde{\phi}, \tilde{v}) + \mathbf{u_p} \tag{4.34}$$

$$\delta\dot{\phi} = \dot{\phi} - \dot{\tilde{\phi}} = \mathbf{f}_\phi(\tilde{\phi} + \delta\phi, \tilde{\omega} + \delta\omega) - \mathbf{f}_\phi(\tilde{\phi}, \tilde{\omega}) + \mathbf{u}_\phi \tag{4.35}$$

and applying first-order Taylor's expansion, we arrive at

$$\delta\dot{\mathbf{p}} = \mathbf{F_p}\,\delta\phi + \mathbf{G_p}\,\delta v + \mathbf{u_p} \tag{4.36}$$

$$\delta\dot{\phi} = \mathbf{F}_\phi\,\delta\phi + \mathbf{G}_\phi\,\delta\omega + \mathbf{u}_\phi, \tag{4.37}$$

which is combined into

$$\delta\dot{\mathbf{x}} = \mathbf{F}\,\delta\mathbf{x} + \mathbf{G}\,\mathbf{v} + \mathbf{u}, \tag{4.38}$$

where $\mathbf{G}$ is a 6 x 6 diagonal matrix composed of the elements of $\mathbf{G_p}$ and $\mathbf{G}_\phi$, and

$$\mathbf{F} = \left[\begin{array}{c|c} \mathbf{0} & \mathbf{F_p} \\ \hline \mathbf{0} & \mathbf{F}_\phi \end{array}\right]. \tag{4.39}$$

Equation (4.38) linearly approximates the process of error accumulation. The term $\mathbf{Gv}$ represents the contribution of the rotational and translational velocity errors, using the random-variable $\mathbf{v} \sim N(0,\ \mathbf{S_v})$, $\mathbf{S_v} = diag(\sigma_v^2, \sigma_v^2, \sigma_v^2, \sigma_\omega^2, \sigma_\omega^2, \sigma_\omega^2)$,

with σ_v^2 and σ_ω^2 being obtained empirically. The random vector $\mathbf{u} \sim N(0,\ \mathbf{S_u})$ accounts for the errors due to inaccuracies in the model such as undulation of the ground traversed during the measurement interval. Its variance-covariance matrix consists of diagonal terms proportional to the maximum slope of the terrain.

From (4.38), we obtain an approximation for the state transition matrix from time t_k to time t_{k+1}:

$$\mathbf{\Phi}(t_{k+1}, t_k) = e^{(t_{k+1}-t_k)\mathbf{F}} \approx \mathbf{I} + (t_{k+1} - t_k)\mathbf{F}. \tag{4.40}$$

Knowing $\mathbf{\Phi_k} = \mathbf{\Phi}(t_{k+1}, t_k)$, we can use the standard Kalman filter prediction equations [Brown and Hwang, 1997]:

$$\delta\hat{\mathbf{x}}_{\mathbf{k+1}}^{-} = \mathbf{\Phi_k}\,\delta\hat{\mathbf{x}}_k^{+} = \mathbf{0} \tag{4.41}$$

$$\hat{\mathbf{P}}_{\mathbf{k+1}}^{-} = \mathbf{\Phi_k}\,\hat{\mathbf{P}}_{\mathbf{k}}^{+}\,\mathbf{\Phi_k^T} + \mathbf{Q_k}, \tag{4.42}$$

In the above, the minus sign superscript denotes predicted values from the previous step, before the current measurements are processed. Conversely, the plus sign superscript denotes values updated with the current measurements. The first prediction equation results in the null vector, because in an extended Kalman filter the errors $\delta\hat{\mathbf{x}}_k^{+}$ at the end of each iteration are fed back to the odometry and the filter state variable is zeroed out. The matrix $\mathbf{Q_k}$ is computed from

$$\mathbf{Q_k} = \int_{t_k}^{t_{k+1}} \mathbf{\Phi}(t_{k+1}, \tau)\left[\mathbf{G_k}\,\mathbf{S_v}\,\mathbf{G_k^T} + \mathbf{S_u}\right]\mathbf{\Phi^T}(t_{k+1}, \tau)d\tau. \tag{4.43}$$

The observation model for the compass module is quite simple. It is already linear:

$$\mathbf{z_c} = [\varphi,\ \theta,\ \psi]^T \quad \mathbf{H_c} = \begin{bmatrix} \mathbf{0} & \mathbf{I} \end{bmatrix} \quad \mathbf{R_c} = diag(\sigma_t^2, \sigma_t^2, \sigma_h^2), \tag{4.44}$$

with the tilt and heading variances, σ_t^2 and σ_h^2, obtained from the manufacturer specifications. The compass is calibrated using publicly available magnetic deviation charts for the local area. It is setup to trigger an alarm whenever the magnetic field strength is too low or too high. In such a case, the heading information is unreliable and is ignored by zeroing the bottom right element of $\mathbf{H_c}$.

When a new measurement sample is obtained at time t_{k+1}, the filter is presented with the difference $\mathbf{z_{k+1}} - \mathbf{H_{k+1}}\,\mathbf{x}_{\mathbf{k+1}}^{-} = \mathbf{z_c} - \mathbf{H_c}\,\mathbf{x_o}$ between the actual observation and the one predicted using the current odometry estimate $\mathbf{x_o}$. The error state is updated according to the well-known EKF equations

$$\mathbf{K_{k+1}} = \hat{\mathbf{P}}_{\mathbf{k+1}}^{-}\,\mathbf{H}_{\mathbf{k+1}}^{\mathbf{T}}\,(\mathbf{H_{k+1}}\,\hat{\mathbf{P}}_{\mathbf{k+1}}^{-}\,\mathbf{H}_{\mathbf{k+1}}^{\mathbf{T}} + \mathbf{R_{k+1}})^{-1} \tag{4.45}$$

$$\delta\hat{\mathbf{x}}_{\mathbf{k+1}}^{+} = \delta\hat{\mathbf{x}}_{\mathbf{k+1}}^{-} + \mathbf{K_{k+1}}\,[\mathbf{z_{k+1}} - \mathbf{H_{k+1}}\,\mathbf{x}_{\mathbf{k+1}}^{-} - \mathbf{H_{k+1}}\,\delta\hat{\mathbf{x}}_{\mathbf{k+1}}^{-}] \tag{4.46}$$

$$\hat{\mathbf{P}}_{\mathbf{k+1}}^{+} = (\mathbf{I} - \mathbf{K_{k+1}}\,\mathbf{H_{k+1}})\,\hat{\mathbf{P}}_{\mathbf{k+1}}^{-}. \tag{4.47}$$

Finally, the error is transferred from the error state to the total state, clearing the error state variable, and the loop reiterates.

4.3 Global Positioning Systems

The abbreviation GPS typically refers to the NAVSTAR[1] Global Positioning System which is a United States satellite-based navigation system that provides instantaneous accurate information about time, position and velocity anywhere on or near the Earth. There are in fact two existing satellite-based systems for global positioning — the US NAVSTAR GPS and the Russian GLONASS[2] — with a third

[1]NAVSTAR — Navigation System with Timing and Ranging.
[2]GLONASS — Global Navigation Satellite System.

one in the works[3]. All operate according to the same principles but differ in implementational details. For the purposes of my thesis, they are indistinguishable and I will use the abbreviation GPS to refer to any or all of them.

4.3.1 Background

A satellite-based navigation system consists of three segments: a *space segment*, a *control segment* and a *user segment*. The space segment consists of a constellation of satellites orbiting the Earth at an altitude of more than 20000 km. The number of satellites and their orbits are designed such that at least the minimum number of satellites needed for positioning are visible anywhere at any time. The control segment consists of *monitor stations*, which track the satellites, a *master control station*, which processes the data from the monitoring stations to compute the exact satellite orbits, locations and clock parameters, and *ground control stations*, which are used to upload data to the satellites. Finally, the user segment consists of the GPS receivers people use in their applications.

The satellites broadcast spread spectrum radio signals which are carrier signals at predetermined frequencies modulated by various codes and *the navigation message*. The navigation message contains the health status, the clock and the *broadcast ephemerides* of the satellite, *the satellite almanac* and various other data. The ephemerides are essentially a set of data containing three records: general information, orbital parameters and clock parameters of the satellite. The parameters are used by the receiver to compute its position and velocity. They are uploaded

[3]The European Space Agency's Galileo will have its first satellite launched in 2004 and is expected to reach full operational capability in 2008. It will be compatible with both GPS and GLONASS.

to the satellite after the master control station processes the latest tracking data. The almanac is a shorter low-accuracy version of the ephemerides of all satellites. Its primary purpose is to aid the receiver in its search for other satellites or to help construct visibility charts for planning tasks.

When a GPS receiver catches a satellite signal for the first time, it decodes the navigation message and starts tracking the satellite. With each consecutive message, the receiver uses the satellite clock information to compute the difference Δt between the time the message was sent and the time it was received. From there, using the speed of light c, the *pseudorange* $\tilde{\rho}$ to the satellite is

$$\tilde{\rho} = c\,\Delta t = \rho + c\,\delta t. \tag{4.48}$$

The pseudorange differs from the actual range ρ because it does not take into account the time offset δt between the satellite and receiver clocks.

From the navigation message, the satellite position $\mathbf{x_s}$ with respect to an Earth-fixed coordinate system is also known. If we denote the receiver position by $\mathbf{x_r}$, then

$$\tilde{\rho} = ||\mathbf{x_s} - \mathbf{x_r}||. \tag{4.49}$$

Thus, if the receiver locks onto at least four satellites, it can solve the non-linear system of equations (4.49) and obtain its position. It can similarly compute its velocity by using the Doppler principle of radio signals.

Obviously, this model assumes a straight-line path of the signal from the satellite to the receiver. Thus, it is susceptible to various kinds of errors as the signal deviates from the straight line due to refractions in the troposphere and the ionosphere. Other sources of errors are the inaccuracies of the satellite orbits,

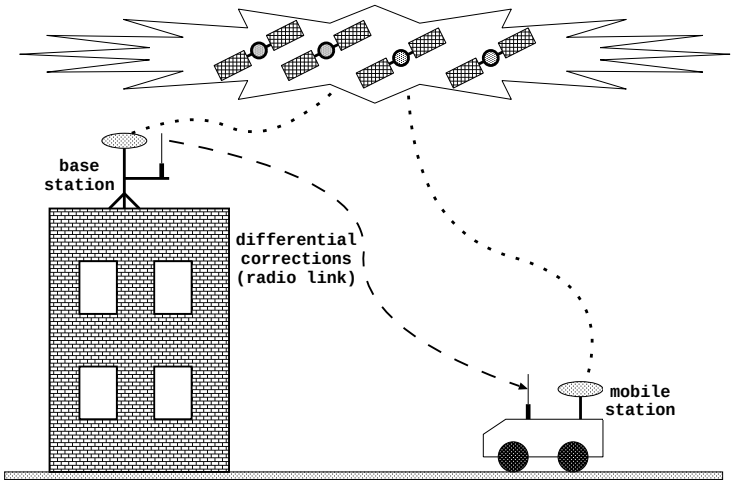


Figure 4.4: The differential GPS technique.

relativistic effects and signal reflections. These sources can typically contribute position errors of up to 30 m. A major concern until it was turned off in 2000 was also *Selective Availability* — the intentional denial of accuracy for civilian users of NAVSTAR GPS by the US Department of Defense.

It turns out, however, that the errors affecting observers standing close to each other are strongly correlated and this fact is used in the differential GPS (DGPS) technique to improve the accuracy. The basic principle of DGPS is to use two receivers (Figure 4.4). One, called a *base station* is permanently affixed at a known location while the *remote receiver* is used as intended by the user. Both receivers compute their positions as usual. However, the base station, knowing its true location, subtracts it from its measurements to obtain a *differential correction vector*. This correction is sent over a radio link to the remote receiver which applies it to its own estimate. The result is a much improved accuracy, typically below a meter for distances of a few kilometers from the base station.

With the advance of the interferometric techniques, an even more accurate

GPS method was invented. The *real-time kinematic* GPS (RTK GPS), also known as *carrier-phase differential* GPS (CPD GPS), measures the phase of the carrier signal instead of its time of flight. It does this with a precision that allows for a more accurate estimation of the pseudorange to the satellite. Since only the phase shift is measured directly, however, this leaves some ambiguity in the integer number of cycles the signal makes on its way. This ambiguity is resolved by using both the carrier and the code phase measurements. This process, known as ambiguity resolution or carrier phase initialization, takes up to several minutes and depends on the baseline length, the number of satellites in view, their configuration and required reliability. When the convergence process is complete, the receiver produces the so-called *fixed solutions* which are of centimeter-level accuracy.

4.3.2 Coordinate Systems and Calibration

The satellite coordinates broadcast in the ephemerides are with respect to an Earth-fixed coordinate system. The official reference frame for the NAVSTAR GPS, for example, is the World Geodetic System 1984 (WGS-84). For the convenience of surveyors and most other users, the receivers usually convert the computed position automatically to ellipsoidal coordinates (latitude, longitude, and altitude). In the case of robot navigation, however, a local "world" Cartesian coordinate system is preferred to act as a reference frame. The data from the receiver needs to be converted accordingly.

The first step is to convert the latitude φ, the longitude λ and the altitude

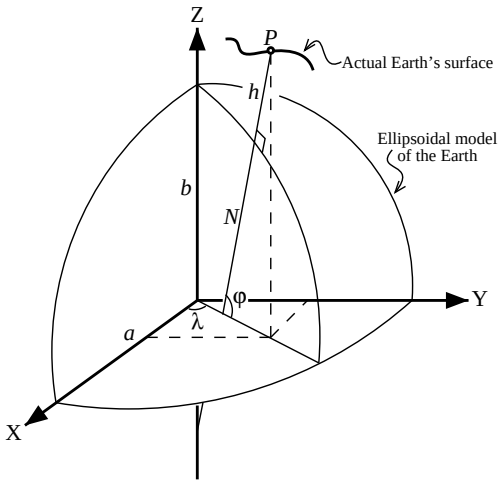


Figure 4.5: Cartesian and ellipsoidal coordinates.

h to Cartesian Earth-fixed coordinates (Figure 4.5). According to WGS-84:

$$N = \frac{a^2}{\sqrt{a^2 \cos^2 \varphi + b^2 \sin^2 \varphi}} \tag{4.50}$$

$$X = (N + h) \cos \varphi \cos \lambda \tag{4.51}$$

$$Y = (N + h) \cos \varphi \sin \lambda \tag{4.52}$$

$$Z = (\frac{b^2}{a^2} N + h) \sin \varphi, \tag{4.53}$$

where $a = 6{,}378{,}137$ m and $b = 6{,}356{,}752.31425$ m are the Earth's major and minor semi-axes.

Next, the Earth-fixed coordinates are converted to an intermediate right-handed coordinate system centered at the origin $[X_0, Y_0, Z_0]^T$ of the "world" coordinate system and having its axes in the east, north and up directions (ENU). The formula is:

$$\begin{bmatrix} x_{ENU} \\ y_{ENU} \\ z_{ENU} \end{bmatrix} = \begin{bmatrix} -\sin \lambda_0 & -\sin \varphi_0 \cos \lambda_0 & \cos \varphi_0 \cos \lambda_0 \\ \cos \lambda_0 & -\sin \varphi_0 \sin \lambda_0 & \cos \varphi_0 \sin \lambda_0 \\ 0 & \cos \varphi_0 & \sin \varphi_0 \end{bmatrix}^T \begin{bmatrix} X - X_0 \\ Y - Y_0 \\ Z - Z_0 \end{bmatrix}, \tag{4.54}$$

where φ_0 and λ_0 are the latitude and longitude of the center of the chosen world coordinate system.

Finally, a rotation is applied to adjust for the different orientation of the "world" coordinate system. Since in most cases it is desirable to have the z axis pointing up, the adjustment is simply a rotation about the z axis:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = Rot_z(\psi_0) \begin{bmatrix} x_{ENU} \\ y_{ENU} \\ z_{ENU} \end{bmatrix}. \tag{4.55}$$

For the tests on the Columbia University Morningside Campus, described in this thesis, I have chosen a world coordinate system, centered at an easily accessible location and oriented along the major directions of the campus, the map of which is shown in Figure 4.6. The center of the coordinate system, marked on the map, is precisely defined as being 2 m to the left of the vegetation planter next to it. The x axis runs approximately from west to east parallel to the streets, the y axis runs approximately from south to north parallel to the avenues, and the z axis points up.

Since the directions of this coordinate system are not precisely east-north-up, some calibration was necessary to obtain its orientation angle ψ_0. This was done by surveying two fairly distant points along the y axis. The GPS antenna was positioned at each point and left stationary for enough time to allow the convergence to a fixed solution. Then, the position fixes were recorded for a few minutes, averaged for a statistically better accuracy and converted to ENU coordinates. The orientation angle is given by:

$$\psi_0 = \arctan\left(\frac{y_2^{ENU} - y_1^{ENU}}{x_2^{ENU} - x_1^{ENU}}\right). \tag{4.56}$$

Figure 4.6: The campus coordinate system

4.3.3 Fusing GPS with Odometry and the Attitude Sensor

The EKF framework described in the previous section allows to easily integrate additional sensors. This is demonstrated here with the addition of a GPS receiver which is used to keep the position uncertainty from accumulating beyond acceptable limits.

The GPS receiver provides absolute position information via the standard GGA message and the variance of the pose computation along the main axes via the standard GST message. Every time a new update from the GPS becomes available,

it is incorporated in the EKF by augmenting the measurement vector with the expected position of the GPS antenna. Denote the coordinates of the antenna with respect to the robot by $\mathbf{p}'_g$. The measurement prediction of the absolute position $\mathbf{p}_g$ of the antenna is:

$$\mathbf{h}(\mathbf{x}) = \mathbf{p} + R(\phi)\,\mathbf{p}'_g. \tag{4.57}$$

Since this is not linear in the state variables, we need to linearize to obtain the measurement matrix

$$H_{k+1} = \left.\frac{\partial \mathbf{h}}{\partial \mathbf{p}}\right|_{\hat{\mathbf{x}}_\mathbf{k}}. \tag{4.58}$$

The variance of the measurement is converted from ENU to the world coordinate system by

$$\mathbf{R_{k+1}} = Rot_z(\psi_0) \begin{bmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_z^2 \end{bmatrix} Rot_z^T(\psi_0), \tag{4.59}$$

where σ_x^2, σ_y^2 and σ_z^2 are the reported by the receiver variances along the east, north and up directions.

The filter is presented with the difference between the actual position fix $\mathbf{z_{k+1}}$ and the predicted by odometry measurement $\mathbf{h}(\hat{\mathbf{x}}^-_{\mathbf{k+1}})$ and the computation proceeds as in equations (4.45)–(4.47), except that equation (4.46) now becomes:

$$\delta\hat{\mathbf{x}}^+_{\mathbf{k+1}} = \delta\hat{\mathbf{x}}^-_{\mathbf{k+1}} + \mathbf{K_{k+1}}\,[\mathbf{z_{k+1}} - \mathbf{h}(\hat{\mathbf{x}}^-_{\mathbf{k+1}}) - \mathbf{H_{k+1}}\,\delta\hat{\mathbf{x}}^-_{\mathbf{k+1}}]. \tag{4.60}$$

Since the GPS is the only sensor in this method that makes exteroceptive position measurements, the overall accuracy of the method depends strongly on the accuracy of the GPS fixes. If GPS quality deteriorates, the uncertainty in the pose estimates may become too large. In such cases, positioning data is needed from

other exteroceptive sensors. But in order to seek such data, there has to be a way to detect such situations.

The probability distribution of the pose estimates is a six-dimensional Gaussian represented by the 6 x 6 variance-covariance matrix $\hat{\mathbf{P}}^{+}_{\mathbf{k+1}}$. This matrix defines a generalized 6-D ellipsoid whose rotational axes are the eigenvectors of the matrix of length their corresponding eigenvalues. Each of the eigenvalues is the variance of the corresponding element (position or orientation coordinate) of the state vector.

Whenever a new GPS update is processed by the filter, a test is performed to check the uncertainty in the overall pose estimate. The eigenvalues of $\hat{\mathbf{P}}^{+}_{\mathbf{k+1}}$ are computed by using the Jacobi method [Press *et al.*, 1992]. If any of the ones associated with translation is greater than a threshold, this is considered an indication that the uncertainty is too large and an attempt is made to use the visual localization method described in the next chapter. Only the uncertainy in translation is considered because if the orientation is wrong it will quickly cause the translation error to increase too.

4.4 Summary

This chapter presented a complete mobile robot localization system for use in large-scale open environments. The system is based on three sensors: odometry, an attitude sensor, and a global positioning sensor.

I use the robot odometry as a foundation for the other sensors to build upon. Its high update rates and low-dynamics error profile make it an excellent source to fall back to when nothing else is available. Its inherent problem with error accumulation has been addressed by reducing the impact of the systematic errors

and by integration with the other sensors.

The systematic errors are reduced by careful calibration. I have designed a complete calibration procedure tailored to the peculiarities of the ATRV-2 robot. It extends the method of Borenstein and Feng and recovers the necessary parameters to undo the gross effects of kinematic mismodeling.

I have combined the odometry with an attitude sensor to form a dead reckoning system that produces estimates in 3-D. I have also added a global positioning sensor which provides the needed external reference to keep the dead reckoning error from accumulating infinitely.

The data from the three sensors is integrated efficiently by an extended Kalman filter. The system keeps track of both the pose of the robot and the uncertainty associated with it. The method is light in computation and can be performed in real-time.

The overall accuracy of the system depends on the quality of the GPS data. When reliable GPS fixes are available, as it is normally the case in open areas, the system should be able to handle the localization task on its own. When the robot enters an area of poor GPS coverage, by monitoring the variance-covariance matrix, it can detect the increased uncertainty in the estimates and decide to use additional sensors.

Chapter 5

Visual Localization

One of the most appealing characteristics of visual images is the rich information they convey. Unfortunately, processing the images to extract this information is not very easy. There is a clear trade-off between the time one is allowed to spend on processing the image and the quality and robustness of the results.

In many visual localization algorithms, the focus has been placed on the (near) real-time performance. The perceived necessity for quick processing has put an unwieldy burden on these algorithms. This has typically led to a simplistic "visual odometry" type of an implementation, where one or a set of features are tracked from frame to frame and the absolute pose of the robot is computed as the composition of the relative poses between each pair of successive frames. This approach suffers from the same problem as conventional odometry: unbounded error accumulation. An additional problem is that tracking occasionally fails as features are lost or misidentified across frames.

There is no compelling reason why visual localization has to be done as frequently and quickly as possible. Most mobile robots today are equipped with

good enough wheel encoders to be able to navigate for some time without the need for external reference. Combined with additional sensors, such as the attitude sensor and GPS in my case, a fast and light on computational resources localization system is achieved that allows for good navigation throughout most of the operating environment.

To expand the working range of such a system, it is sufficient to provide occasional "on-demand" additional updates only when the above configuration fails. Visual pose-estimation algorithms are well poised to do that. By acting less frequently and on demand, they can be allowed more time for heavy image processing operations which can be used to increase the robustness of the overall system. This also improves the efficiency in the use of limited computational resources.

This is the underlying idea in the use of my visual localization subsystem. As the robot moves, it uses the method described in Chapter 4 to keep track of its pose along with the uncertainty associated with it. As long as it is confident in these pose estimates, no attempts are made to use vision. If the confidence becomes low, then the robot is allowed to stop and take reasonable time to compute a more accurate estimate using the vision-based pose estimation method described in this chapter. Since this happens relatively rarely and because of the large-scale environments, the time the robot spends in place doing image processing is negligible to the overall travel time for a given mission.

The visual pose estimation is based on matching an image of a building taken by the camera with a model of that building. The next section describes the model in detail. The following sections explain how the appropriate building to use is chosen and how the pose is computed.

5.1 Environmental Model

To make good use of an environmental model, it needs to be carefully designed. The design depends on the application that will use it, as one model could be very appropriate for a certain application and completely unusable for a host of other applications. For the task of visual localization, the following characteristics are important:

- **Detail** — the scale of the features modeled
- **Coverage** — the extent to which the model covers the area of interest
- **Accuracy** — how closely does the model match the real environment in terms of metric information, such as width, height, length, distance, etc
- **Size** — how much space does it take on the computer
- **Ease of creation** — how easy it is to obtain the model from existing data or build one from scratch.

The environmental model for my visual localization method uses has been designed with the above properties in mind. It is in fact a data base of smaller-scale facade models. Each facade model depicts the features of a near-planar region around a building facade (Figure 5.1). The features modeled are typical and abundant in a human-made environments — they are dominant straight lines formed mainly by surface discontinuities, however, other straight lines, such as ones formed by color discontinuities, can also be used. All lines are finite segments represented by the 3-D coordinates of their endpoints in a local for the model reference coordinate system. The origin of the coordinate system, called *the reference point*, and

Figure 5.1: A sample model (right) of a building facade (left)

the orientation of the axes can be picked arbitrarily. An essential part of the facade model is the transformation between its reference frame and the world coordinate system used as a reference throughout the entire environment of operation.

In order to be useful, each facade model needs to capture features with a certain level of detail to provide sufficient information for the robot to find its pose. The level of detail varies across buildings and depends on the appearance of each facade. In some cases, the boundaries of the facade along with the largest ledges may suffice, while in others the window frames and surrounding elements need to be used. Beyond that level, however, adding more detail quickly reaches the point where it does not help the pose estimation. Hence the facade models are simple and light.

There is no particular reason to require complete coverage — in other words, there is no need to model every facade of every building. What is needed is that enough building facades are modeled to allow continuous localization throughout the area of interest. It is sufficient, that wherever the robot stops, it finds a suitable model to use for localization. Moreover, areas with proven good and stable GPS

reception may not require any modeling.

The size of the model is of concern not so much because of limited computer memory, but because of the time it would take to process. The model I use is compact since it only describes a small part of the environment with simple geometric primitives. Not every building facade is modeled and only a few prominent features of each chosen facade are picked. This allows the localization method to scale well with the size of the operating environment which is important for every outdoor application.

It is essential that the model is accurate, since the accuracy of the model determines the accuracy of the overall performance of the visual localization. Precise metric information is required about the location, orientation and length of each line segment. Additionally, the location and orientation of each modeled facade with respect to the established world coordinate system is also needed.

The model is easy to derive from available sources or build from scratch. Normally, architectural plans for the buildings contain the necessary information to create models of the building facades. If these are not available, a few key measurements with a ruler of the dimensions of the windows and the ledges around them is typically enough. The relative building positions can be obtained from 2-D maps of the area or measured using a 3-D scanner or an electronic theodolite.

In this thesis, I assume that the environmental models are already available. My ideas on how to create these models are discussed in Chapter 7.

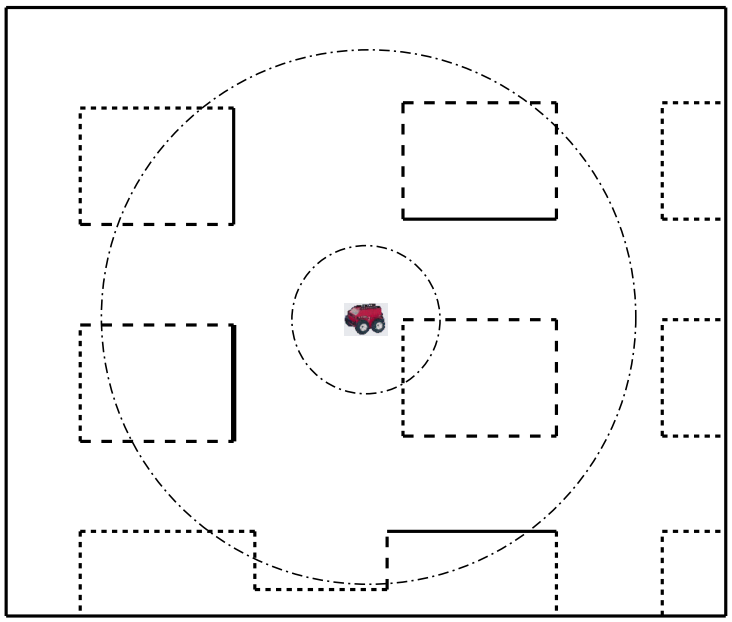


Figure 5.2: Choosing a model: A top-down view of modeled facades of buildings are shown on the map. The two circles show the minimum and maximum distance allowed. The dotted lines are models that are outside of this range. The dashed lines are models that are within the range but are viewed at a very low (or negative) angle. The solid lines are good to use. The thick one is chosen because it is closest to the robot.

5.2 Choosing a Model to Use

When visual pose estimation is attempted, a rough estimate of the robot pose is still available from the other sensors. This estimate is used to search the model data base for the most appropriate building facade to use for visual localization. This is done in two steps according to two criteria: distance and viewing angle (Figure 5.2).

The first step is to scan through the model database index to determine the facade models that are within a good distance from the robot. Both minimal and maximal limits are imposed: If a building is too close, it may not have enough

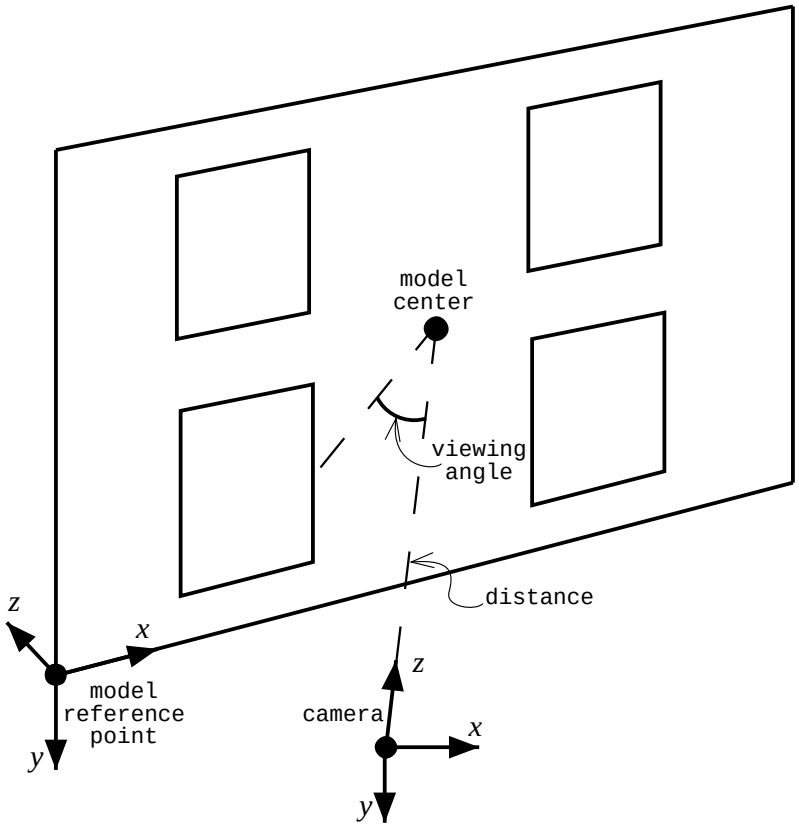


Figure 5.3: Criteria for choosing a model: distance and viewing angle

visible features on the image. Further, the edge detection may detect a lot of small-scale line features that are not really important (for example, brick boundaries). If a building is too far, the accuracy of the result becomes low because of the fixed camera resolution.

The distance (Figure 5.3) is computed between the estimated current position of the robot and the mass center of the facade model. The mass center is defined as the average of the sum of the coordinates of the endpoints of all line segments in the model. The distance computation is done very quickly, because the mass center is precomputed from the model and is immediately available.

The first step manages to quickly eliminate the majority of the models that are clearly inappropriate for use at the current robot location. This ensures the manageability of the model data base in potentially very large scale environments. The models that are within the good range are indexed for further consideration; the rest of them are disregarded.

The second step is to eliminate those facade models from the first step based on the viewing angle (Figure 5.3). The viewing angle is defined as the angle between the reference plane of the model (its O-x-y plane) and the direction from the robot to the center of the model. The angle is positive, when the robot is on the side of that plane with negative z coordinates, and negative otherwise. A right angle means that the camera is looking straight at the building; a negative angle means the facade is not visible as it is facing away from the camera.

Only models that are viewable under a larger than a positive threshold angle are considered. This eliminates both the facades that are not visible and the ones that are visible but at a very low angle. Low viewing angles are not desirable due to the foreshortening effect: many line features project very close to one another on the image and it becomes difficult to find the correct correspondence with the model.

The models that successfully pass this two-step selection process form the set of good candidate models to use. Any subset of this set can be used in the pose estimation step. As the processing time is not trivial, however, I choose only the one that is closest to the robot. Because of the finite resolution of the camera, this choice is likely to provide the most accurate result.

Finally, the camera is turned toward the chosen facade. The mass center of

the model is converted from model-based Cartesian coordinates to robot-centered spherical coordinates using the current estimate of the robot pose. This gives the pan and tilt angles the camera is expected to turn to in order to look directly at the center of the model. The pan-tilt unit is commanded to turn in this direction and an image is acquired.

In practice, the final orientation of the PTU is different from the ideal one, because the angles computed are only approximations which depend on the accuracy of the current robot pose estimate. For the distances involved, however, and the typical accuracy of the pose estimates, the resulting orientation error of the camera is usually small enough and does not effect the following computation. Further, since the camera is aimed at the center of the model, any small deviation will have minimal effect.

5.3 Pose Estimation

At this stage, a pair of an image and a model of the building facade are available and the task is to determine the pose of the robot. Since the pose of the camera is tracked by the pan-tilt unit rigidly affixed to the robot, if the pose of the camera is known, the pose of the robot can be easily derived. Thus, the focus in this section is the computation of the camera pose.

The pose computation is done by matching identical linear features in the image and the model. This is the well-known data association problem which in general is intractable because of the sheer number of possible associations between detected features on the image and the ones in the model. On the one hand, there is a set of n_2 2-D line segments extracted from the image, and on the other — n_3

3-D line segments are available from the model. Some of the lines from the model have matching lines from the image. Of course, not all lines from the model do, as well as not all 2-D lines have a match in the model. Thus, an ordered list of n matching segments needs to be determined from each set, where n is only known to be not greater than the minimum of n_2 and n_3.

A brute-force approach is not feasible because of its extreme computational requirements. If all possible combinations need to be tried, the total number of iterations k_b is:

$$k_b = \binom{n_2}{n} \binom{n_3}{n} n! \tag{5.1}$$

This is computationally prohibitive for any practical values of n_2, n_3 and n.

Instead, I have adopted a probabilistic approach following the well-known RANSAC paradigm first introduced by Fischler and Bolles [Fischler and Bolles, 1980]. The method consists of the following steps:

1. Preparation
2. Sampling
3. Pose candidate computation
4. Pose candidate refinement
5. Pose candidate evaluation

Step 1 is executed once, while the rest of the steps are repeated in a loop with a predetermined number of iterations.

5.3.1 Preparation

The purpose of the preparation step is to obtain the line segments and do some pre-processing necessary for the later steps. Depending on the task at hand, two collinear lines with distinct endpoints can be considered different lines or different representations of the same infinite line. In the latter case, the two lines are redundant and may greatly increase the computational burden. Thus, in the preparation step, two versions of each line segment set are created (Figures 5.4 and 5.5). The first one, called the *original set*, keeps the collinear lines separate for the pose evaluation step. The second set, called the *merged set*, combines all collinear segments into a single representative in order to keep the computation light in the sampling, pose computation and pose refinement steps.

The 3-D line segments are explicitly represented in the model of the facade and need little processing. They are used as the *original set* of 3-D lines (Figure 5.4, left). Their merged version is obtained by detecting the collinear subsets and replacing each one of them with a single line whose endpoints are the two outermost end points of the subset (Figure 5.4, right). Note that the merged lines are not exactly treated as infinite lines since their endpoints are used in the alignment metric below.

The image of the building is processed to obtain the 2-D line segments. A Canny edge detector is applied to locate edgels and then the incremental line fitting technique is used to connect them in straight line segments which form the *original set* (Figure 5.5, left). Next, the collinear subsets are detected and merged, replacing each such subset by a single line segment of length which is the sum of the length of the individual segments (Figure 5.5, right). Resulting segments that are shorter

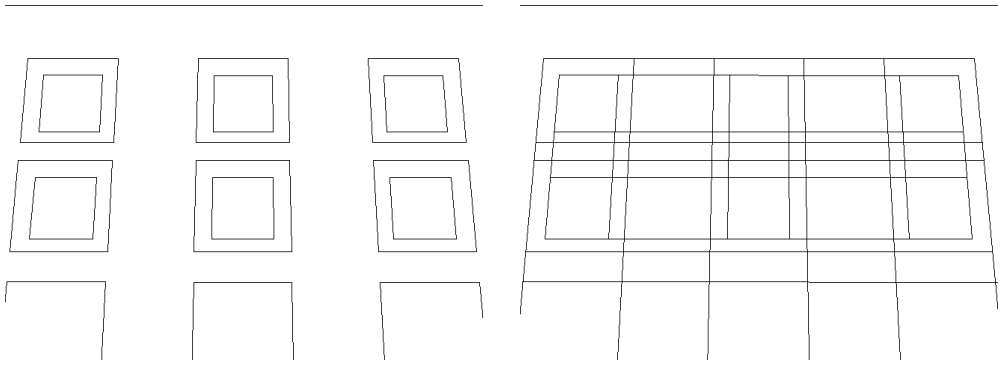


Figure 5.4: Preprocessing of the 3-D line segments: the original lines from the model (left) and the result after merging the collinear subsets (right).

Figure 5.5: Preprocessing of the 2-D line segments: the extracted edge lines (left) and the result after merging the collinear subsets and removing the short ones (right).

than a given threshold are eliminated from further consideration and the rest of them are used in the sampling and pose computation steps.

5.3.2 Sampling and Pose Candidate Computation

The idea behind RANSAC is to solve the pose estimation problem a number of times using randomly chosen matches between a minimum number of 2-D and 3-D line segments. The minimum number of matching pairs in my case is three: the problem has six unknowns (three for position and three for orientation of the camera) and each matching pair of segments provides a two-degrees-of-freedom constraint. The equations are non-linear and more than one solution is possible, however, the initial pose estimate from the other sensors is usually good enough to converge to the correct one. Thus, in the sampling step, I randomly select three pairs of lines and, based on this selection, compute an estimate for the camera pose.

The camera pose candidate is found by using the pose estimation method proposed by Kumar and Hanson [Kumar and Hanson, 1994]. A perspective camera model is used and the calibration parameters of camera are assumed to be known. An error function is composed and minimized that quantifies the misalignment of the 3-D line and its matching 2-D line from the sample. For each 2-D line l_i, consider the plane that is formed by that line and the camera center of projection (Figure 5.6). Let the normal to that plane is N_i. Suppose, l_i is matched with the 3-D line segment s_j whose endpoints $P_{j,1}$ and $P_{j,2}$ have world coordinates $\mathbf{p_{j,1}}$ and $\mathbf{p_{j,2}}$. If R and T are the rotation and translation that align the world coordinate system with the one of the camera, then

$$d_{i,j} = (N_i \cdot (R \cdot \mathbf{p_{j,1}} + T))^2 + (N_i \cdot (R \cdot \mathbf{p_{j,2}} + T))^2 \tag{5.2}$$

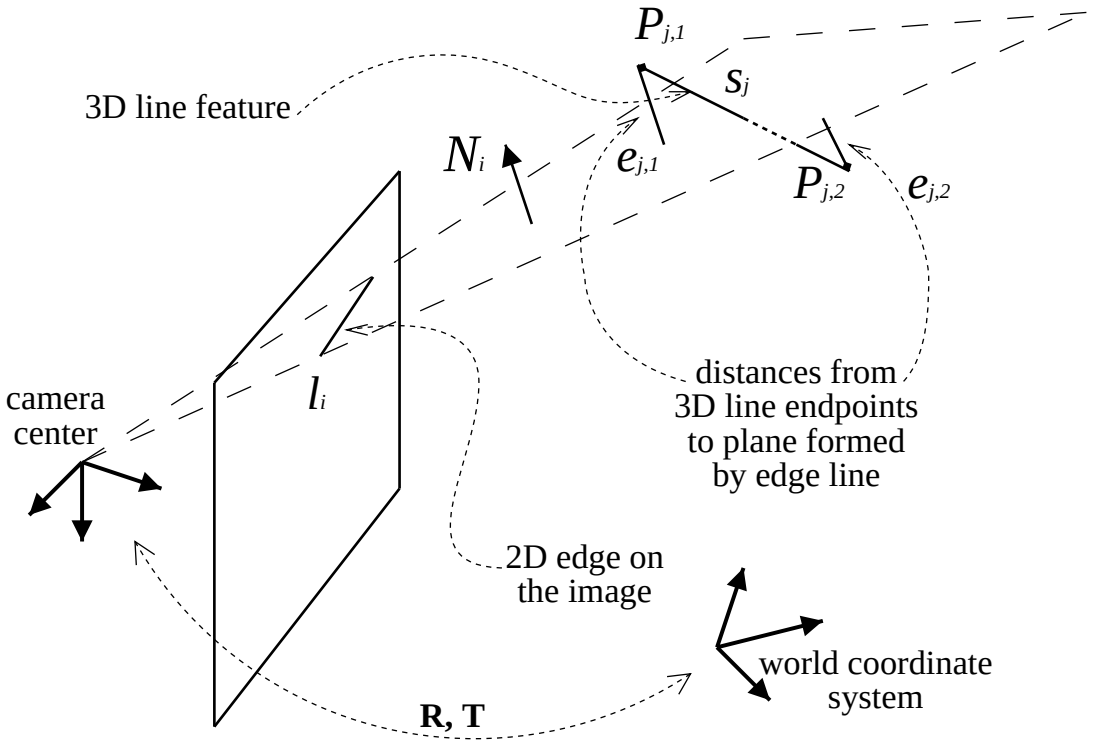


Figure 5.6: Error metric used for pose estimation

is the sum of squared distances of the endpoints of s_j to the plane formed by l_i (Figure 5.6). The error function that is minimized is the sum of $d_{i,j}$ for the three matching pairs:

$$E(R,T) = \sum_{i,j \in Matches} d_{i,j}. \tag{5.3}$$

This function is minimized with respect to the six degrees of freedom for the camera pose: three for the rotation R and three for the translation vector T. The computation follows the method proposed by Horn [Horn, 1990]. If the camera undergoes an infinitesimal incremental rotation $d\omega$ and an infinitesimal incremental translation dT, the resulting error function will become

$$E' = E(R(d\omega) \cdot R, T + dT), \tag{5.4}$$

where $R(d\omega)$ is the rotation matrix that rotates a point $\mathbf{p}$ by $d\omega$:

$$R(d\omega) \cdot \mathbf{p} = \mathbf{p} + d\omega \times \mathbf{p}. \tag{5.5}$$

Differentiating (5.4) with respect to $d\omega$ and dT and setting the result to zero gives us a linear system of six equations with six unknowns, which are the elements of $d\omega$ and dT. The solution to this system provides incremental updates to the current rotation matrix R and translation vector T. This is repeated a number of times until the error metric becomes smaller than a threshold or until a maximum number of iterations are performed.

5.3.3 Pose Candidate Refinement

The pose candidate refinement step uses the consensus set to fine tune the estimate. The consensus set is the set of all matching pairs of 2-D edge segments from the image and 3-D line segments from the model that agree with the initially computed pose candidate.

For each 3-D line segment in the model, a neighborhood of its projection on the image is searched for 2-D edges and their distance from the 3-D line segment is computed according to (5.2). The 2-D edge with the smallest distance is taken to be the match, if that distance is less than a threshold and if the 2-D line is not closer to another 3-D line. If no such 2-D edge is found, then the 3-D line segment is assumed to have no match.

The above matching scheme ensures a one-to-one match. However, recall that the lines used in this step are from the *merged set* and each such line represents a number of collinear *original* line segments. Consequently, each matching pair

of merged lines represents a number of possibly many-to-many matches between *original* line segments. The pose candidate refinement step does not attempt to establish matches between the *original* line segments, because they are not needed at this stage. This keeps the computation simple and fast.

The consensus set is used together with equation (5.3) to compute a better pose estimate. This is done iteratively a number of times starting with a large value for the consensus threshold and gradually decreasing it. The large initial value for the threshold makes sure that a roughly correct consensus set will be generated initially which will be later refined to eliminate the false positives and increase the accuracy. The result of the last iteration is the pose candidate that is evaluated in the next step.

5.3.4 Pose Candidate Evaluation

The quality of each pose candidate is judged by a metric $q(R, T)$ which quantifies the amount of support for the pose by the matches between the model and the edge. The idea is to check what portion of the model is covered by matching edge lines. The larger the coverage, the better the pose candidate. Ideally, the entire visible portion of the model should be covered by matching 2-D edge lines.

After the last refinement iteration, the consensus set contains pairs of matching 3-D lines from the model and 2-D lines from the edges. Consider one 3-D line s_j in the consensus set and its matching 2-D counterpart. Since the matching is one-to-one, neither of these lines matches any other line.

Now, recall that s_j and l_i were obtained in the preparation step by merging a number of *original* line segments. For clarity, let $S_j = \{s_{j,1}, s_{j,2}, ..., s_{j,n}\}$ be the set

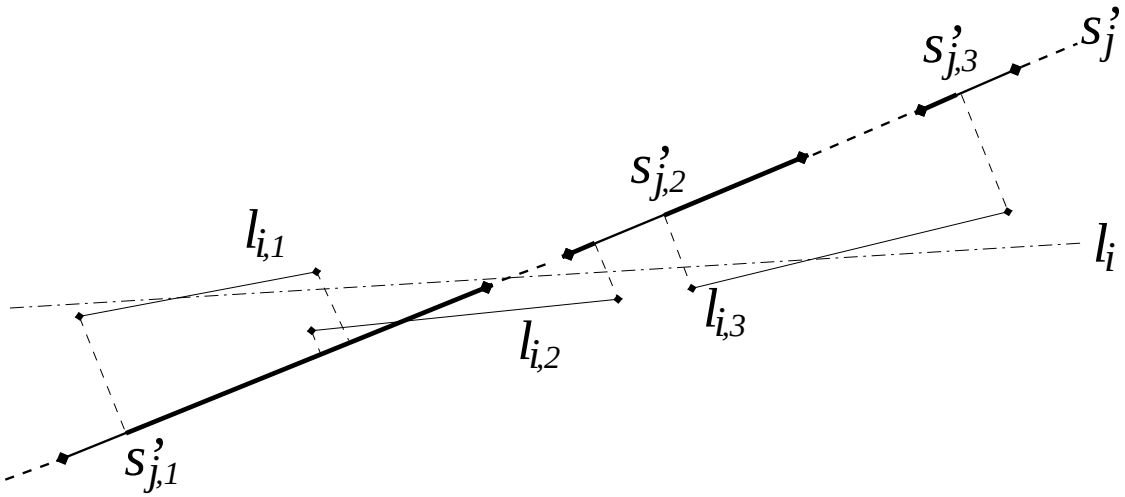


Figure 5.7: Coverage of a pair of matching lines. The light lines denote original edge line segments. The thick lines are the projections of the original 3-D lines from the model. The merged 3-D and 2-D lines are shown in dashed and dot-dashed style respectively. The covered portion of the model is shown as extra thick.

of all segments from the model that were merged into s_j and $L_i = \{l_{i,1}, l_{i,2}, ..., l_{i,m}\}$ be the set of all edge segments that were merged into l_i. Denote the projection of s_j onto the image by s'_j. Then the line segments in S_j project onto a set of segments $S'_j = \{s'_{j,1}, s'_{j,2}, ..., s'_{j,n}\}$ of s'_j because s_j was constructed as a geometric superset of the lines in S_j.

Next, project all line segments in L_i orthogonally onto s'_j (Figure 5.7). The projection defines the region of s'_j that is considered *covered* by L_i. By geometrically intersecting this projection with the line segments in S'_j, we obtain the portion of S'_j that is covered by L_i. The total length of the lines in the intersection is the amount of coverage $c(s_j)$ of s'_j. Thus the total coverage of the entire model for the pose candidate R, T is:

$$C(R,T) = \sum_{s_j \in Model} c(s_j) \tag{5.6}$$

The dependence on R and T is implicit as the consensus set and the projections s'_j

depend on the pose.

Note that the coverage is a quantity which is computed in 2-D space. As such, it depends on the scale of the model as well. If the camera moves away from the building, the visible size of the model will diminish and $C(R,T)$ will decrease even if the match is perfect. Hence normalization needs to take place.

There are two ways to normalize the coverage: divide by the total projected length of the model or divide only by the visible projected length. The former approach will tend to underrate the correct pose in cases when the model is slightly outside of the field of view. The latter approach will do fine in such cases but will overrate poses for which very little of the model is visible and the visible portion can easily match arbitrary edge lines. I choose to use the latter method and compute the pose evaluation metric as

$$q(R,T) = C(R,T) \,/\, V(R,T) \tag{5.7}$$

where $V(R,T)$ is the total length of the visible projection of the model on the image.

To avoid the pitfalls of choosing an overrated pose, a number of safety checks are applied. There are three criteria by which a given pose candidate is eliminated from consideration:

1. If the pose candidate is outside of a validation gate, it is immediately rejected as unlikely. The validation gate is determined by the total state estimate of the extended Kalman filter. It is the region of minimum volume around that estimate which contains a given probability mass under the Gaussian assumption. If $\mathbf{x}$ is the pose candidate (R and T) expressed in terms of the

total state of the filter, then it is within the validation gate, if

$$[\mathbf{x} - \mathbf{x}_k]^T \hat{\mathbf{P}}_\mathbf{k}^{-1} [\mathbf{x} - \mathbf{x}_k] \leq \gamma, \tag{5.8}$$

where $\mathbf{x}_k$ is the current EKF estimate of the pose and $\hat{\mathbf{P}}_\mathbf{k}$ is the corresponding covariance matrix. The parameter γ is a threshold controlling the volume of the region (in terms of number of sigmas) around the EKF estimate to be included in the gate. It is chosen from tables of the chi-square distribution [Bar-Shalom, 1987].

Note that in this case the expensive metric computation is not performed at all. As this is most often the case since the probability of a correct random sample is quite low, significant time savings are achieved by this step.

2. If the visible portion of the model on the image is less than a threshold, the pose is also rejected as there is no sufficient basis to evaluate it, even if it is the correct one. If this is case, the entire localization step is likely to fail, because the camera was pointed way off target. Note that in this case, the metric computation is also skipped, resulting in speed improvements.

3. If the value $q(R, T)$ for the current pose candidate is less than a threshold, the pose is also rejected as there is insufficient support for it.

Of all the pose candidates that pass the three tests, the one with the highest score after the loop is the best one and is accepted to be the correct pose. The uncertainty is set to a diagonal matrix consisting of statistically obtained values scaled by the distance to the model. The values are obtained by running the algorithm a number of times on various images for which ground truth is available,

and taking the standard deviation of the errors of all successful results normalized by the corresponding distance between the robot and the building.

If no good pose is found, the visual localization step fails. This is not fatal, however, as the robot simply moves a little further along its route and attempts another visual localization step. This is repeated until either the visual localization succeeds, or the GPS picks up a good signal and corrects it pose to reduce the uncertainty.

5.3.5 Number of Iterations and Speedups

The decision on how many iterations to perform is based on the expected number of trials k_r required to get a correct match. If the number of line segments obtained from the image is n_2, the number of line segments in the model is n_3, and n of them appear in both the model and the image, then the probability of a single sample being correct is

$$p = \left(\frac{n}{n_3}\right)\left(\frac{n-1}{n_3-1}\right)\left(\frac{n-2}{n_3-2}\right)\left(\frac{1}{n_2}\right)\left(\frac{1}{n_2-1}\right)\left(\frac{1}{n_2-2}\right). \tag{5.9}$$

The expected value $E(k_r)$ of the number of trials is then

$$E(k_r) = \sum_{i=1}^{\infty} i\, p\, (1-p)^{i-1} = \frac{1}{p}. \tag{5.10}$$

Of course, $E(k_r)$ depends on the number of matching lines segments which is impossible to know in advance. To resolve this problem, I use a fixed number of iterations that only depends on the model. This number is computed on the basis of the number of lines in the model and the average number of edge lines produced at the preparation step for the particular building facade. It can be controlled to a large degree by setting appropriately the length threshold for the merged edge lines

in the preparation step: if a large number of edge lines is generated, increasing the threshold will let only the longest (and thus most reliable) edge lines be used for sampling. Conversely, if too few edge lines are generated, decreasing the threshold will help exhaust more combinations.

It is important to note that in (5.9), n_2 and n_3 refer to the much smaller number of line segments after their collinear subsets are merged into a single line, not the ones originally extracted from the image or the model.

It is also important to note that even with the speedup resulting from merging the collinear segments, the number of iterations is too large for practical purposes. Typical values for the number of segments are $n_2 = 30$, $n_3 = 30$, $n = 20$. This results in an expected number of $86,755.8$ iterations which take significantly longer than one can comfortably allow for visual localization.

Additional decrease in the number of iteration is achieved by splitting the line segments into two disjoint groups: mostly horizontal and mostly vertical ones. This is easy to do for the lines from the model, since the information is directly available. It is also possible to do it for the edge lines, because the tilt of the robot is accurately measured by the attitude sensor and the building facade is assumed to be vertical, near planar surface. Misclassifications of edge lines are possible but extremely rare and normally do not affect the accuracy of the algorithm.

The benefit of splitting the segments into two groups is to eliminate samples that are certain to be incorrect matches such as ones that associate a horizontal line on the model with a vertical one in the image. The sampling step is modified to always produce samples having one pair mostly horizontal lines and one pair mostly vertical lines. The third pair could be from either class. If there are equal

number of horizontal lines and vertical lines this strategy can reduce the expected number of iterations by up to a factor of 8, resulting in less than 11,000 iterations.

This already is a practical number. For comparison, all visual localization tests in this thesis used a maximum of 8000 iterations, which typically took between 25 and 45 seconds on a 2GHz Pentium IV processor equipped with 1GB of RAM.

5.4 Summary

This chapter described a visual localization method for mobile robots using a single CCD camera mounted on a pan-tilt unit. The method is based on classical computer vision pose estimation techniques and the RANSAC paradigm. It uses linear features which are abundant in urban environments to establish correspondence between an image of a nearby building and a model of it and to compute the pose. A simple and compact model is used; no environmental changes are required.

The method is used only when the main localization method for the robot becomes uncertain. The robot stops and autonomously determines which of the nearby buildings to use for localization. It is able to evaluate the quality of its pose estimates and to reject them if necessary.

Chapter 6

Experiments

To demonstrate the functionality of the mobile robot, the software architecture and the software components I wrote as well as to study the performance of the localization algorithms, I performed a series of tests with a real robot in an actual outdoor environment. The tests took place on the Columbia University Morningside Campus. Three kinds of tests were performed — one that aimed to evaluate the performance of the combination of odometry, attitude sensor and GPS; another that focused only on the vision component; and a final test that used all sensors.

6.1 Localization in Open Space

The purpose of these tests were to investigate the accuracy of the open space localization method described in chapter 4.

Arbitrary trajectories were generated by the *Path Planner* or by the user with the help of the graphical interface, and were executed. The trajectories were piece-wise linear, with the robot turning to its next target in place as soon as it

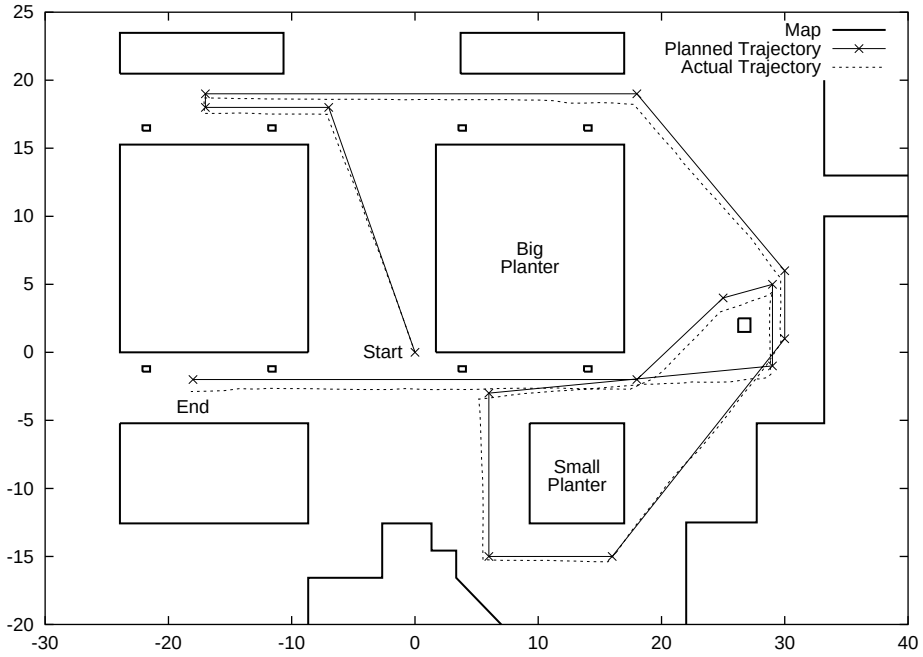


Figure 6.1: The first test run in open space

reached the current one. The maximum translational and rotational velocities were 0.5 m/s and 0.4 rad/s respectively. In all cases, the robot performed as expected with no visible deviation.

To further confirm these results, two more comprehensive test runs were set up to obtain ground truth data. A piece of chalk was attached at the center of odometry on the bottom of the vehicle so that when the robot moved it plotted its actual trajectory on the ground. After it completed the task, sample points from the actual trajectory were marked at intervals of approximately 1 m and measurements of each sample point were obtained.

First, a complex desired trajectory of 14 targets and total length of 210 m was used. (Figure 6.1 shows the planned and actual trajectories, overlaid on the map of this area of the campus. The average deviation of the robot from the

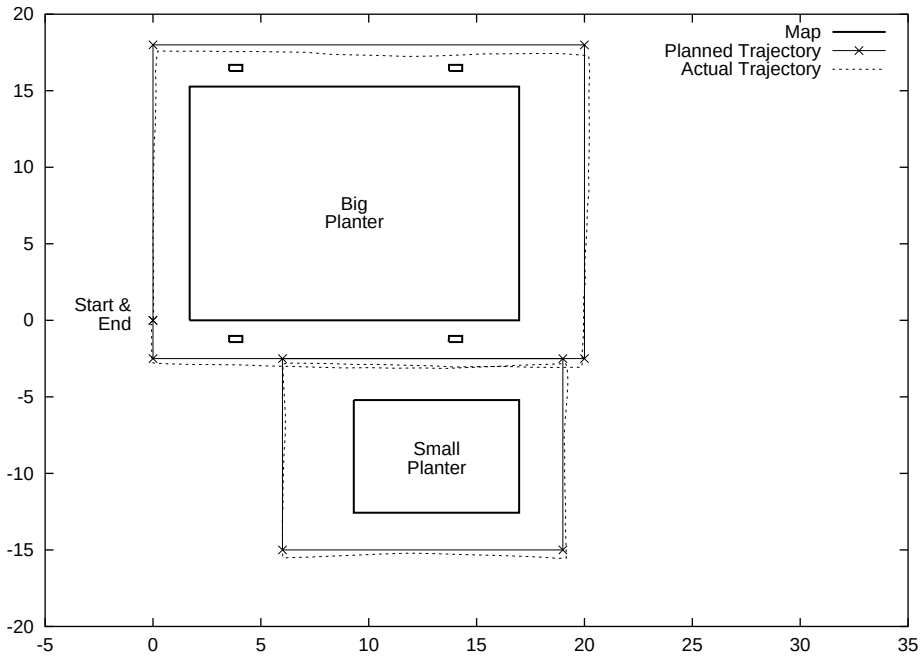


Figure 6.2: A second test run returning to the starting point

planned trajectory in this run was $0.46\,m$.

The second trajectory consisted of nine targets arranged in the shape of the digit eight around the two planters in the center of Figure 6.1. The trajectory was 132 m long and asked the robot to return to the same place where it started (Figure 6.2). The average error for this run was $0.251\,m$.

The next experiment also involved the trajectory in Figure 6.2, but this time of interest was the displacement between the starting and arrival locations. Ideally, the robot had to arrive at its starting location since this was a closed-loop trajectory. Three such runs were performed. The resulting errors were $0.08\,m$, $0.334\,m$, and $0.279\,m$.

It should be noted that the performance of the open-space localization system strongly depends on the accuracy of the GPS data. During the experiment

above, the number of satellites used were six or seven most of the time, occasionally dropping to five or increasing to eight. The GPS receiver was working in RTK float mode in which its accuracy is worse compared to when it works in RTK fixed mode. The latter mode provides accuracy to within a few centimeters, however, it is typically available when seven or more satellites provide good signal-to-noise ratio over a long period of time.

These experiments demonstrate that this localization method is sufficient for navigation in open areas with typical GPS performance and no additional sensors are needed in such cases. The location estimate errors in all of the above test runs were well within the acceptable range for urban environments.

6.2 Localization with Vision

To examine the accuracy of the visual localization method, I performed two kinds of tests: one that compares the result for each test location with ground truth data, and another, that compares the two results the algorithm produced on two different images taken from the same location.

In both kinds of tests, I wanted to measure the quality of the location estimation alone and minimize the interference from inaccuracies in the model. Thus, I took care to create accurate models of the buildings used by scanning their prominent features with a high-quality electronic theodolite with nominal accuracy of 2 mm. The features I modeled were windows, ledges and decorations — all commonly found and abundant in urban structures and easy to find using 2-D image operators (Fig. 6.3).

In the first test, the robot was driven along a long trajectory around the Uris

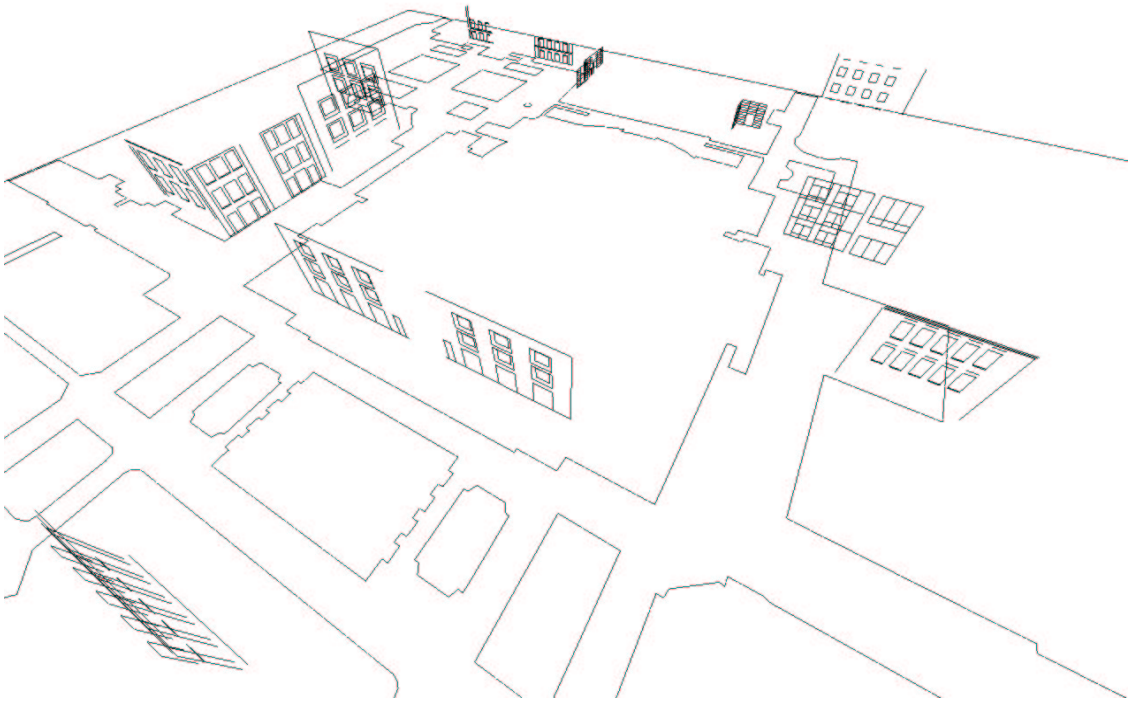


Figure 6.3: 3-D models used for localization shown on a 2-D map of the test area.

building on the campus. At 16 relatively regularly spaced locations the robot was instructed to stop and perform the visual localization routine. It used the accumulated error from odometry as an initial guess to determine the nearby buildings and choose a model of one for localization. A sketch of the test area with the test locations and directions in which the images were taken is shown in Figure 6.4.

Figures 6.5–6.20 show the results of the 16 runs. The left image in each pair shows the model used projected onto the image using the initial inaccurate estimate of the camera pose. The image to the right shows the model projected on the image after the correct camera pose was computed. In all cases the alignment of the model and the image is very accurate.

Since it is extremely difficult to determine the location of the robot with a

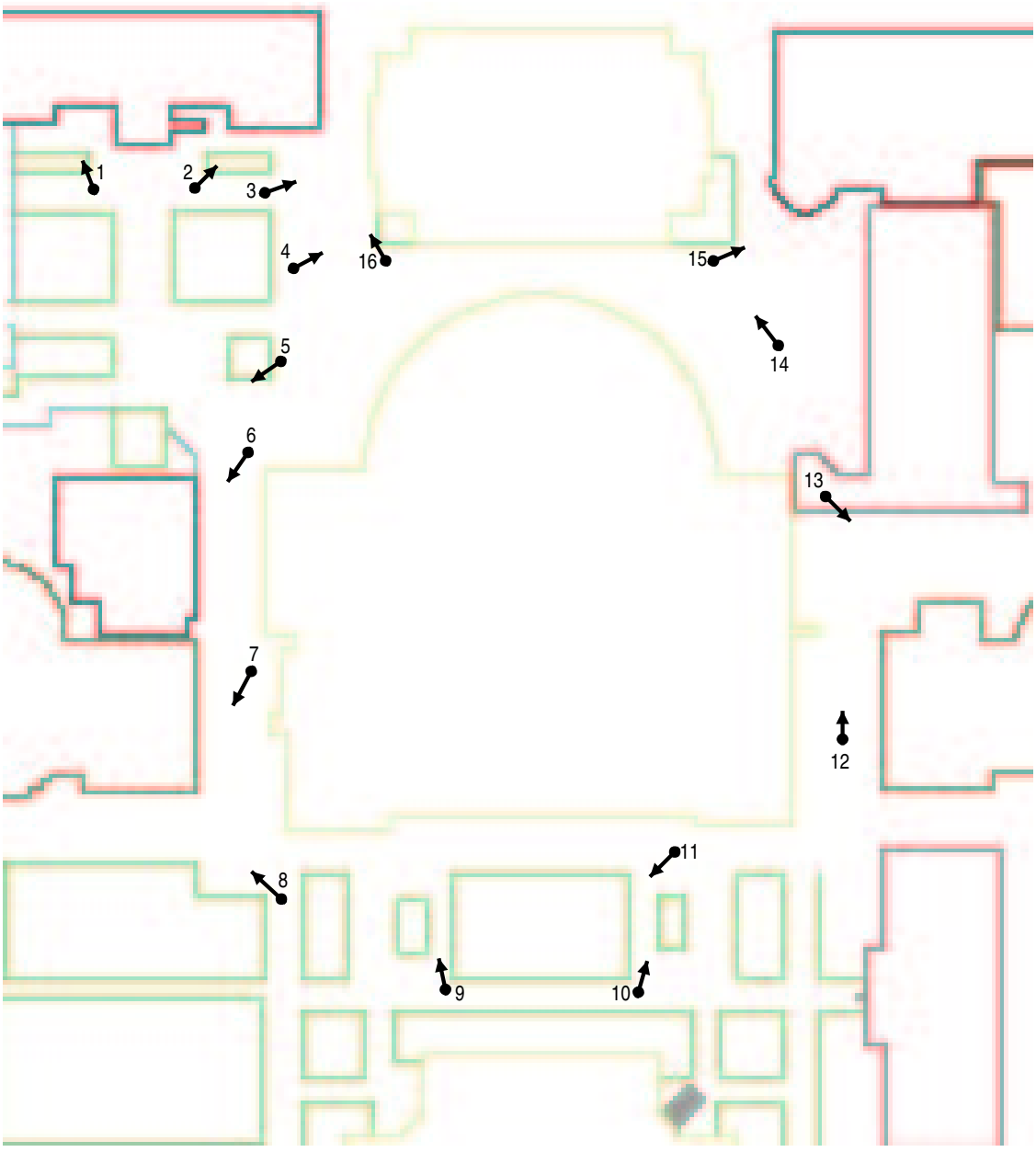


Figure 6.4: A map of the area where the experiments were conducted showing approximate camera locations and orientations.

Figure 6.5: Visual localization test: Location 1

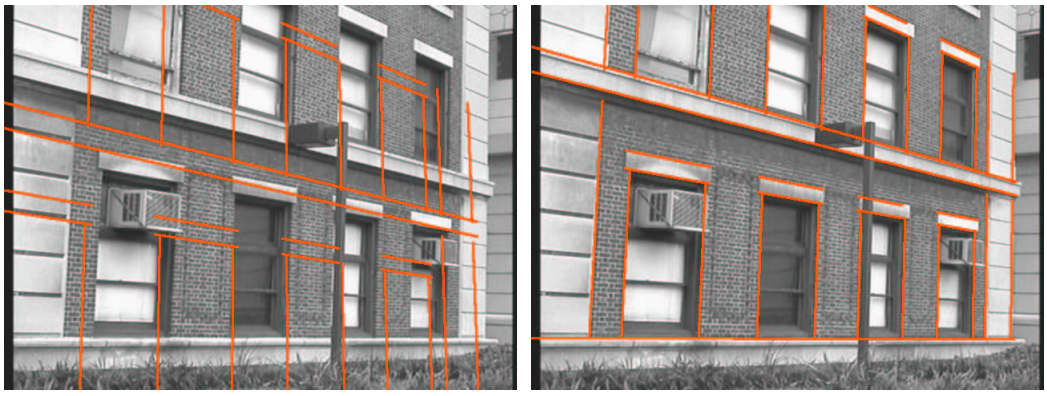


Figure 6.6: Visual localization test: Location 2

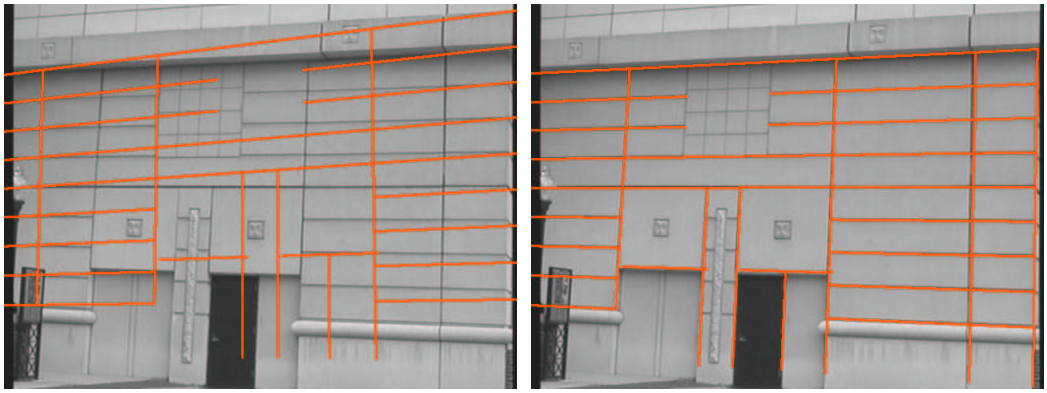


Figure 6.7: Visual localization test: Location 3

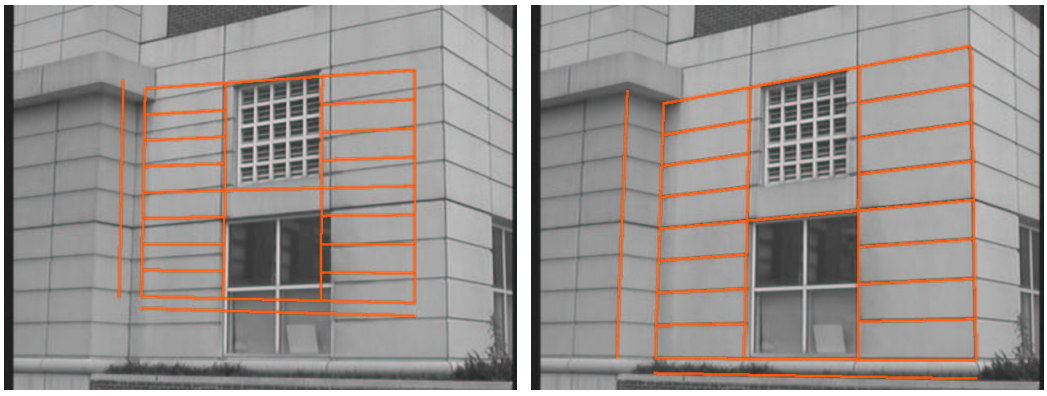


Figure 6.8: Visual localization test: Location 4

Figure 6.9: Visual localization test: Location 5

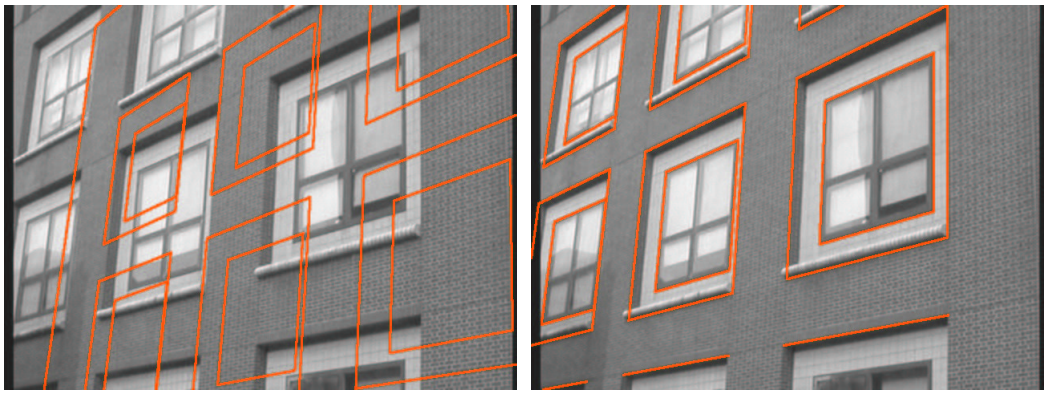


Figure 6.10: Visual localization test: Location 6

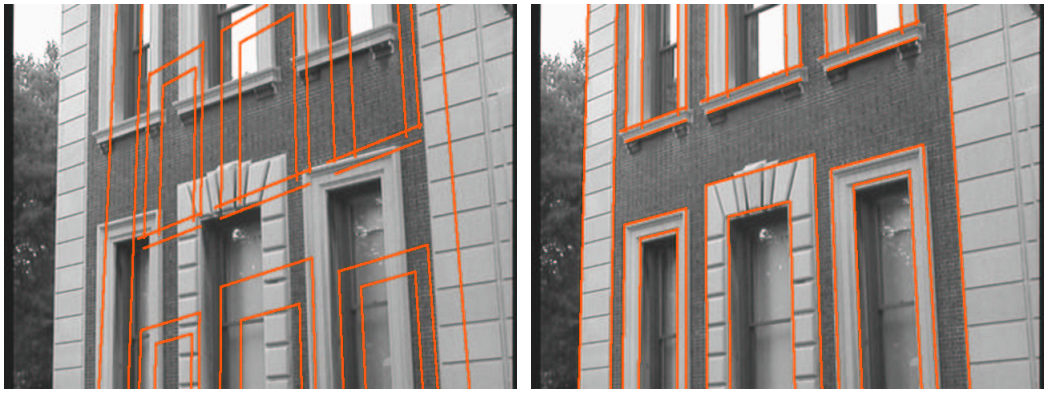


Figure 6.11: Visual localization test: Location 7

Figure 6.12: Visual localization test: Location 8

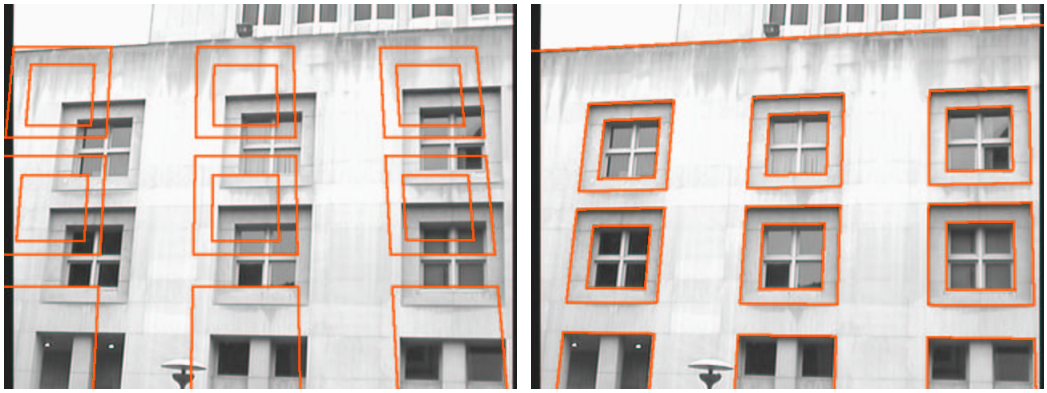


Figure 6.13: Visual localization test: Location 9

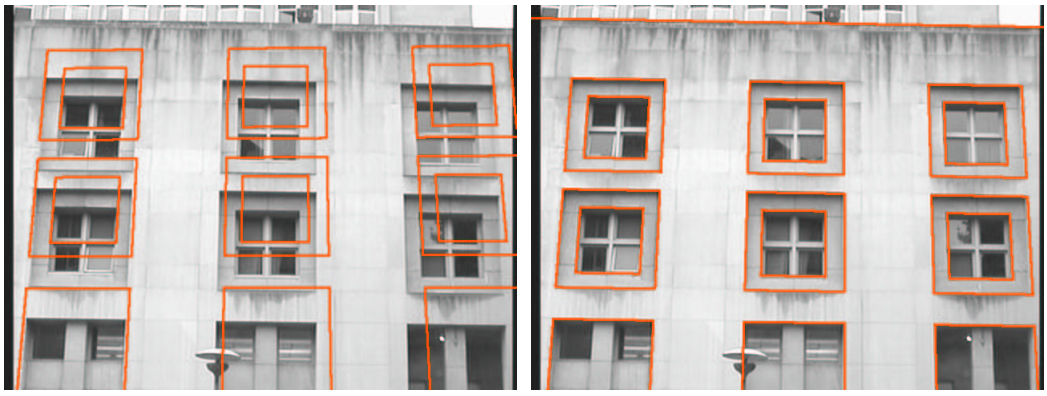


Figure 6.14: Visual localization test: Location 10

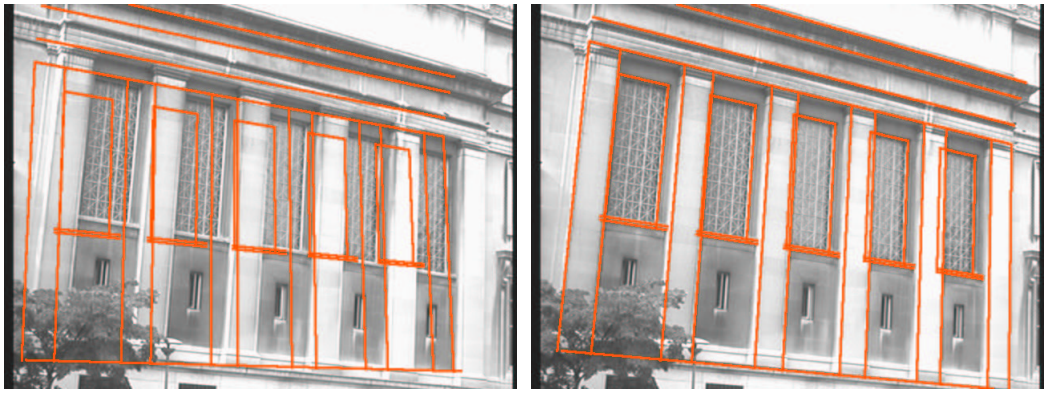


Figure 6.15: Visual localization test: Location 11

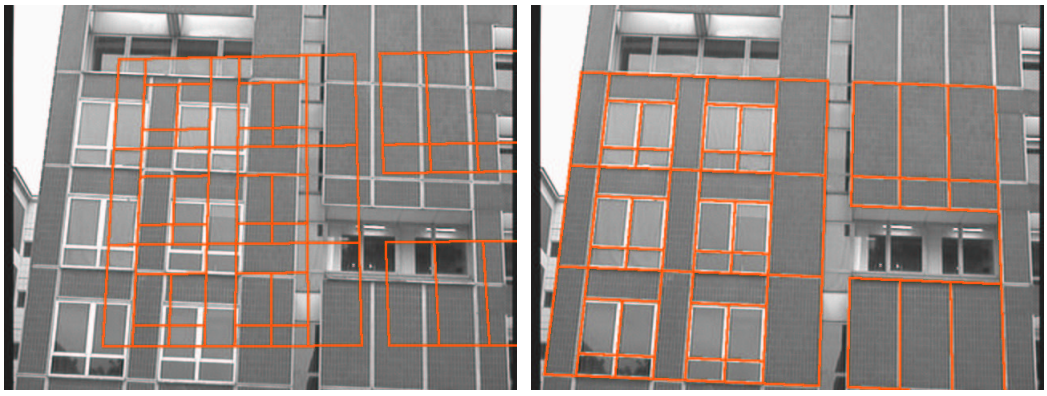


Figure 6.16: Visual localization test: Location 12

Figure 6.17: Visual localization test: Location 13

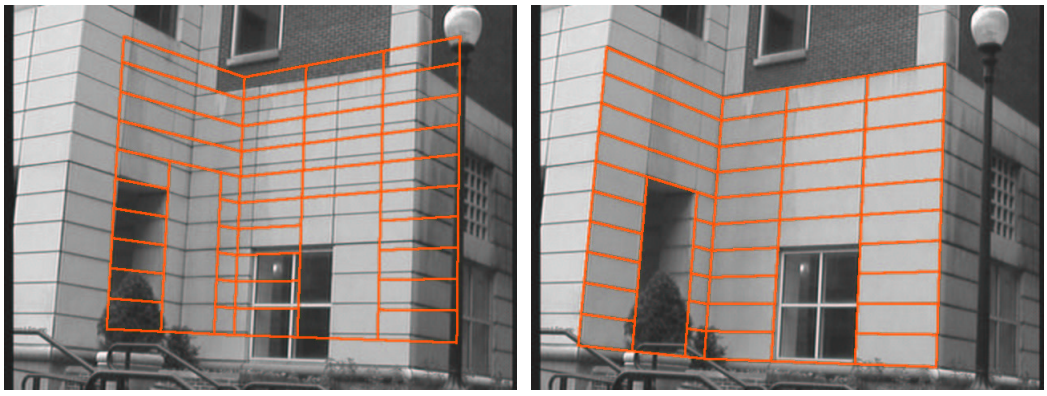


Figure 6.18: Visual localization test: Location 14

Figure 6.19: Visual localization test: Location 15

Figure 6.20: Visual localization test: Location 16

near centimeter-level accuracy, ground truth for the visual localization experiments at each location was obtained in the following manner: An electronic theodolite was placed near the robot and the building facade it was looking at. While the robot was stationary, a scan of the camera lens was taken with the theodolite. Then, a few key points of the building facade were also surveyed so that location of the camera lens could be determined with respect to the building. Finally, the expected location of the camera with respect to the building was computed based on the robot estimate of its pose and it was compared with the one obtained by the theodolite.

Because of the size of the camera lens, the error introduced by scanning its surface, rather than the focal center, was less than 2cm, which is negligible in comparison with the errors of the algorithm. The resulting errors in translation for these runs are given in Table 6.2. These errors are small and clearly demonstrate that the method generates accurate estimates that can be used for robot navigation in urban environments.

The alignment of model and image in the resulting poses suggests that the orientation is also estimated accurately. While this is obvious from Figures 6.5–6.20, I wanted to obtain a quantitative confirmation. I did this by running Tsai's method for external camera parameters estimation [Tsai, 1987] and comparing its orientation estimates with the ones from my localization algorithm. The comparison is shown in Table 6.2, where the rotation angles about each axis are given that would align the camera coordinate frame with the reference frame of the model. The last three columns show the differences in each of these angles between the two estimates. The results demonstrate that the two algorithms consistently agree to

Location Number	Estimated X	Y	Z	True X	Y	Z	Error
1	9.758	-0.456	-16.529	9.964	-0.531	-16.495	0.222
2	-7.585	-0.441	-10.219	-7.517	-0.392	-10.241	0.087
3	14.339	-0.117	-18.336	14.127	0.226	-18.459	0.422
4	13.338	0.181	-14.502	13.354	0.214	-14.630	0.133
5	-12.854	19.328	-17.954	-12.784	19.353	-18.057	0.128
6	30.446	19.192	-12.308	30.413	19.251	-12.353	0.081
7	17.394	0.464	-12.407	17.512	0.260	-12.172	0.333
8	29.417	-0.099	-17.533	29.295	0.045	-17.507	0.190
9	16.626	0.336	-29.673	16.870	0.253	-29.416	0.364
10	38.544	0.241	-30.307	38.789	0.334	-30.177	0.292
11	-10.131	2.067	-33.958	-9.828	2.268	-34.208	0.442
12	9.205	6.265	-35.902	9.181	6.432	-35.591	0.354
13	25.375	15.582	-22.703	25.256	15.798	-22.557	0.287
14	15.060	0.768	-18.029	15.241	0.622	-18.082	0.238
15	-13.372	5.406	-14.906	-13.486	5.264	-15.227	0.362
16	26.040	-0.708	-21.323	25.805	-0.673	-21.575	0.346

Table 6.1: Computed and actual robot position for each of the locations where the visual localization test was performed. The last column shows the total error. Measurements are in meters.

within a fraction of a degree.

The purpose of the second test was to confirm that the algorithm does not generate contradictory results when used on different facades from the same location. I took a pair of images of two faces of the same building at locations 4 and 5 by simply panning and tilting the camera. Both pairs of images were processed with their corresponding models (Figures 6.21–6.22) and were intentionally given initial pose estimates with large errors. The resulting errors in translation were $0.064\,m$ and $0.290\,m$.

Location Number	Localization			Tsai			Difference		
	R_x	R_y	R_z	R_x	R_y	R_z	R_x	R_y	R_z
1	7.637	-18.077	-1.302	7.429	-18.455	-1.243	-0.207	-0.378	0.058
2	3.689	51.895	-3.308	2.958	51.672	-3.862	-0.732	-0.223	-0.554
3	7.181	-19.792	-2.500	7.754	-19.654	-2.907	0.573	0.137	-0.407
4	10.050	-35.144	-0.918	9.814	-35.881	-0.843	-0.237	-0.737	0.074
5	17.254	52.109	-3.059	16.636	51.901	-3.538	-0.618	-0.208	-0.478
6	22.304	-50.205	-4.383	22.445	-50.092	-4.594	0.141	0.113	-0.211
7	12.629	-42.970	1.941	12.074	-43.429	1.995	-0.555	-0.460	0.055
8	23.918	-45.988	-3.723	23.912	-45.858	-3.706	-0.007	0.130	0.017
9	17.626	-11.995	-2.810	17.358	-12.802	-3.028	-0.268	-0.807	-0.218
10	16.108	0.625	-0.761	15.960	0.184	-0.878	-0.148	-0.441	-0.117
11	10.243	30.218	-3.850	10.594	29.306	-3.501	0.350	-0.913	0.349
12	20.236	-0.309	-2.375	20.538	0.095	-2.300	0.303	0.404	0.075
13	22.978	-33.251	-3.219	23.621	-33.492	-3.420	0.643	-0.241	-0.201
14	13.431	-34.960	-4.101	13.392	-35.549	-4.238	-0.039	-0.589	-0.138
15	18.610	56.541	-4.950	17.754	56.307	-5.089	-0.856	-0.234	-0.139
16	8.151	-42.190	-2.943	8.458	-41.745	-2.975	0.307	0.446	-0.032

Table 6.2: Robot orientation as computed by the localization method (first three columns) and by Tsai's method (middle three columns). The last three columns show the discrepancies between the two estimates. Measurements are in degrees.

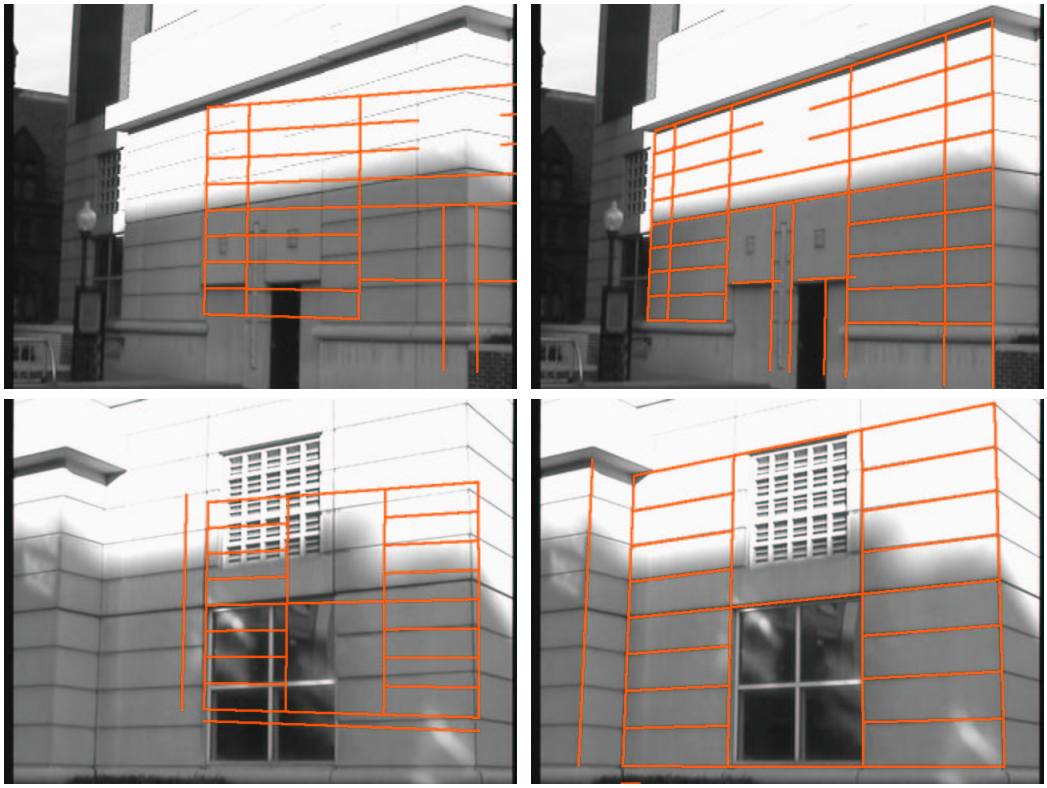


Figure 6.21: Initial and final alignments in the pose estimation tests with a pair of images taken from the same location.

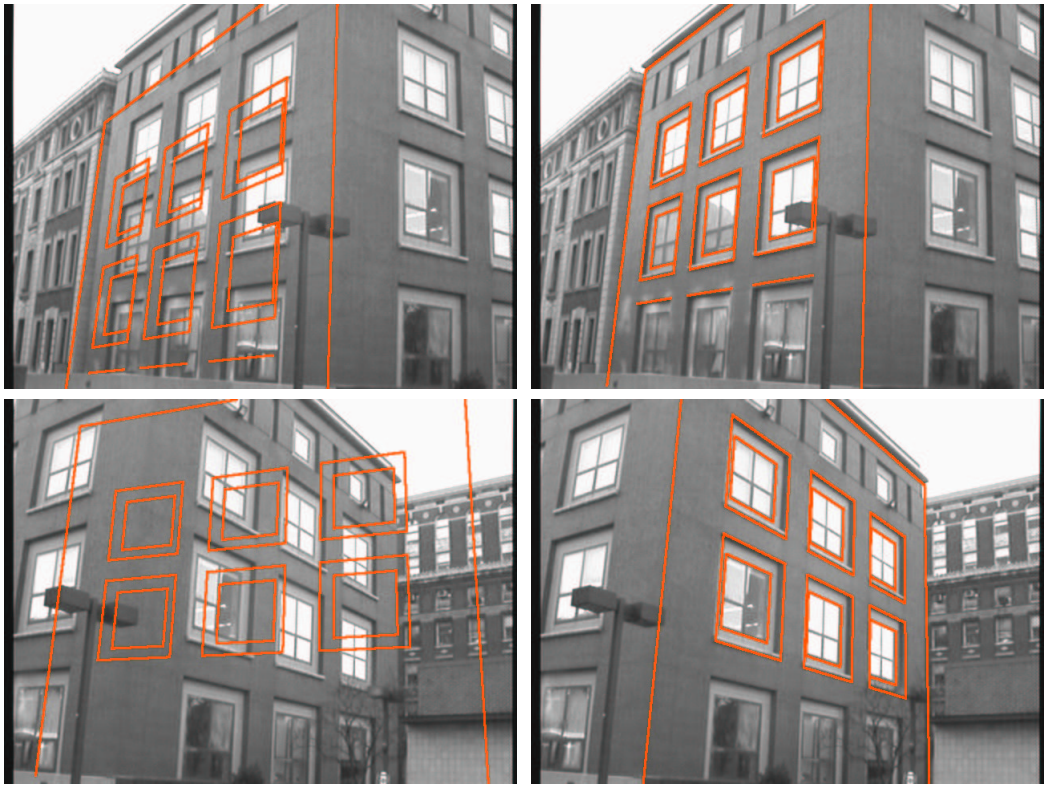


Figure 6.22: Initial and final alignments in the pose estimation tests with a pair of images taken from the same location.

6.3 Localization Using All Sensors

Finally, a test was performed to confirm that the entire localization system works well together, that is, it uses the visual localization as needed and that it actually improves the performance. A more than $330\,m$ long trajectory was composed around the Uris building on the campus (Figure 6.23). The robot was directed to follow that trajectory using all sensors, including vision as needed.

During the test run, the robot passed through both areas of good GPS visibility and poor GPS visibility. It was setup to seek visual localization whenever the standard deviation of the uncertainty of the current position exceeded $1m$. The robot was consistently able to detect the areas of poor GPS performance (marked on Figure 6.23) and supplement it with vision. Notice, that no GPS data was available at all at location 3, as the robot was directly beneath an extension of the nearby building.

It stopped at each marked location, correctly determined a nearby building to use and performed the visual localization procedure described in Chapter 5. While at rest, I scanned its camera with an electronic theodolite to obtain ground truth.

Table 6.3 below shows the estimated position $[X_e, Y_e, Z_e]^T$ of the robot at each location based on the combination of GPS, attitude sensor and odometry prior to triggering the visual procedure, followed by the actual error of that estimate. The following columns show the computed robot position $[X_c, Y_c, Z_c]^T$ by the vision algorithm along with its error. The table clearly demonstrates the improvement the visual algorithm makes to the overall system performance. The corresponding images overlaid with the model are shown in Figures 6.24 through 6.27.

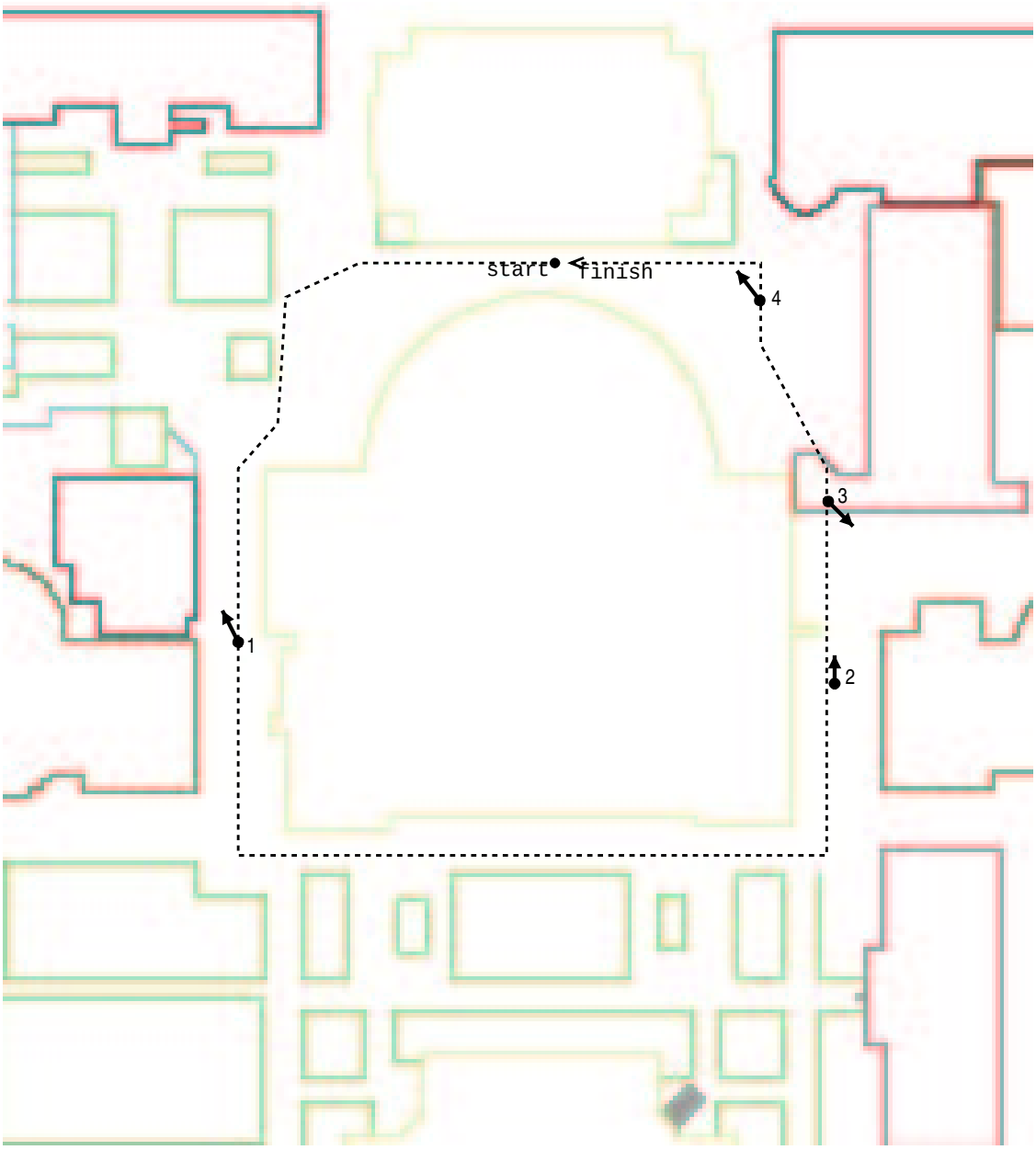


Figure 6.23: A map of the area showing the robot trajectory (dotted line) and the locations where the robot used visual localization. Notice location 3 which is directly underneath a building extension.

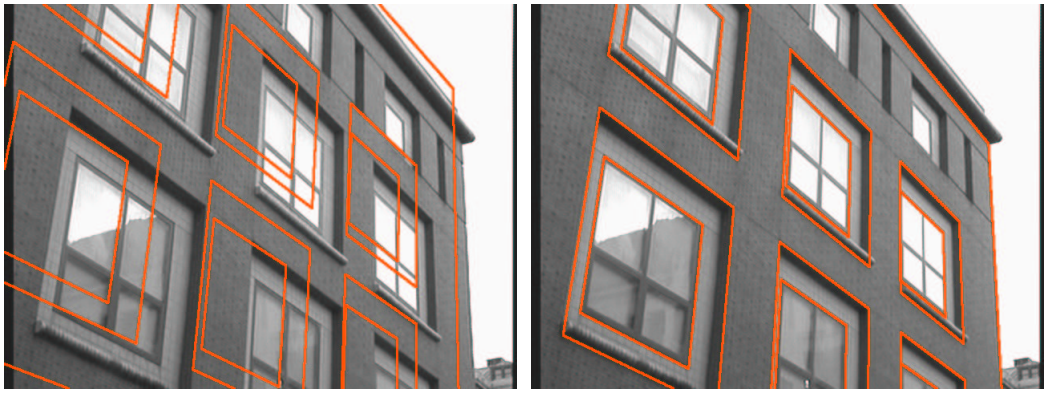


Figure 6.24: Integration test: Location 1

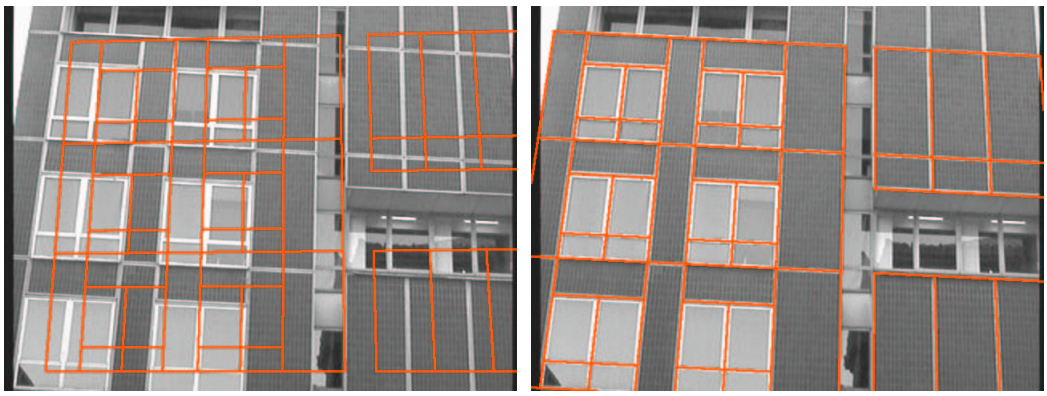


Figure 6.25: Integration test: Location 2

Figure 6.26: Integration test: Location 3

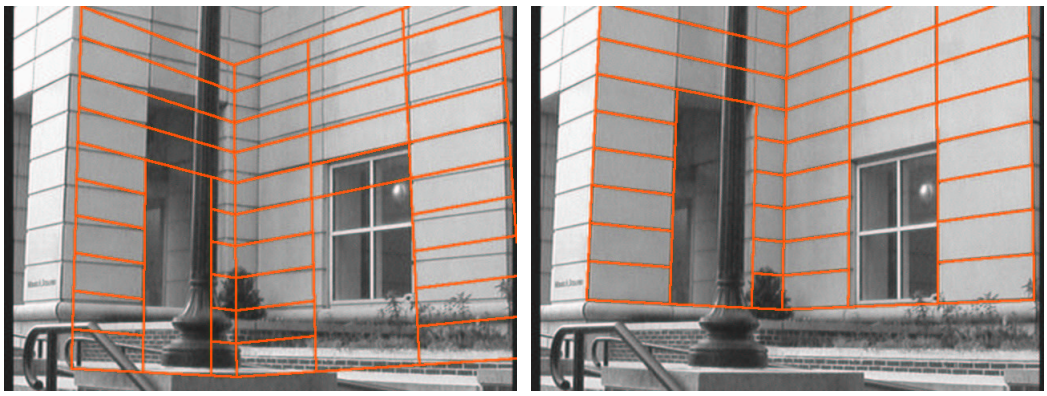


Figure 6.27: Integration test: Location 4

No	X_e	Y_e	Z_e	Error	X_c	Y_c	Z_c	Error
1	17.547	-58.079	-0.683	1.297	16.470	-58.565	0.110	0.348
2	108.521	-61.634	0.279	1.031	108.26	-62.001	1.127	0.345
3	90.66	-30.729	1.418	0.937	91.344	-31.404	0.867	0.179
4	95.095	2.577	1.713	1.212	95.363	2.881	0.85	0.274

Table 6.3: Robot position and error estimated by GPS, attitude sensor and odometry along with the corresponding improved position estimate and error after performing visual localization. Measurements are in meters.

6.4 Summary

This chapter presented the experiments I have done to demonstrate the functionality of the robot and the practicality of the localization methods I described earlier.

The first set of experiments focused on localization in open space. Two long runs were performed for which the actual robot trajectory was sampled and compared to the planned one. Another experiment in a closed-loop trajectory measured if the robot would reliably return to its starting location. In all tests, the robot demonstrated consistent behavior and small enough errors to allow for safe navigation in typical urban environments.

The second set of experiments focused on the vision-based method. An accurate and diverse data base of models of buildings throughout the Columbia University Morningside Campus was created. The accuracy of the method was verified by a comprehensive set of test runs on various buildings. An additional experiment was done to verify the consistency of the method when run on different models from the same location. The results clearly demonstrate that the method is both very accurate and consistent.

Finally, a complete autonomous run was done on the campus in which the

robot was allowed to use all sensors. The trajectory passed through both open and narrow areas. The robot was able to automatically determine when to stop and utilize vision. It chose correct building models and successfully improved its pose estimate using the vision-based method. This run demonstrated the functionality of the entire system.

Chapter 7

Conclusion and Future Work

This thesis presented a systematic and practical approach to mobile robot localization in urban environments. It reflects my work on both system and algorithmic components. The work was done as part of the AVENUE project for urban site modeling, however, the methods and ideas presented here are independent of the project and are generally applicable to mobile robots operating in urban environments.

- On the hardware level, I have designed and implemented a functioning autonomous mobile robot platform for urban site modeling which is being used as the prototype platform for the AVENUE project. The design extended an existing robotic vehicle with a carefully chosen sensor suite: a digital compass with an integrated inclinometer, a global positioning unit, and a camera mounted on a pan-tilt head. A large support frame was built to accommodate the size of the range finder. Necessary wireless networking hardware was also installed on the robot.

- I have designed a distributed modular software architecture for mobile robot localization and navigation. This architecture has been demonstrated to be computer platform and operating system independent. Proper attention has been paid to important issues such as the distribution of computation and wireless network coverage.

- I have implemented the essential part of the software architecture including:

  - software components to interface with the installed hardware, convert hardware-specific data to an easily used format and make it available to the rest of the system
  - a robot motion control component to drive the robot
  - a software component that serves as an interface to the robot from external machines
  - a graphical user interface to control and monitor the robot remotely (joint work with Ethan Gold)

- I have developed a complete calibration procedure for the odometry of the ATRV-2 robot. The method extends the existing *UMBmark* method developed by Borenstein and Feng with a procedure to determine the necessary kinematic parameters of the robot.

- I have developed a mobile robot localization method for open-space outdoor environments. The method uses odometry, an attitude sensor and GPS. It is computationally light and performed in real-time. The method estimates both the robot pose and the uncertainty associated with it. It is used in

open space areas where the quality of the GPS data is good and prevents the uncertainty from growing beyond an accepted tolerance. The method is able to detect when its estimates are likely to be of lower quality so that additional sensors can be utilized.

- To complement the open-space localization method, I have developed a second method based on visual pose estimation. It is heavier computationally but is only used when it is needed. The pose estimation is done by matching an image of a nearby building with a simple and compact model of it. A data base of the models is stored on the on-board computer. No environmental modifications are required.

- Finally, I have demonstrated the functionality of the robot and the localization methods with numerous experiments in Chapter 6. Experiments in open areas demonstrated the applicability of the main localization method. I have created an accurate and diverse data base of models of buildings throughout the northern part of Columbia Morningside Campus. Using this data base, I have shown that the visual localization method produces very accurate and consistent results. A completely autonomous run demonstrated the integrity and functionality of the entire system.

7.1 Limitations

Due to the scope of the system, the difficult nature of the problems and the limited time, there are still a number of open technical and theoretical issues that need to be addressed:

- **View Planning**

  When visual localization is about to be attempted, a decision is made on which facade model of a nearby building to use. This decision is based on a number of criteria, such as distance to the building facade and viewing angle. Obviously, there might be more than one candidate. It is also possible that some facades are occluded by others. My current implementation will choose the first model it finds to satisfy the viewing constraints. Ideally, one would want to pick the best available choice which is not known to be occluded. An interesting theoretical question then is what is the best choice and how to find it. My idea on how to approach this issue is presented in the next section.

- **Feature Matching**

  Inevitably, the visual localization occasionally fails to produce a pose estimate even if the image captured is of the correct building facade. The most common reasons for the failure are mismatches due to too few or too many edges detected on the image, which is somewhat sensitive to the current illumination conditions. Ideas on how the matching process could be improved are discussed in the Future Work section.

- **Partial Visual Localization**

  What happens when the visual localization procedure can not produce a reliable estimate of the current pose? My algorithm tries to act conservatively and rejects every pose for which there is insufficient evidence. However, there might be a way to extract some useful information from the image and improve the current pose estimate, even if the exact six-degrees-of-freedom pose

cannot be determined. This is further discussed in the next section.

- **Visual Pose Integration**

  The main localization system is based on an Extended Kalman Filter which is good at representing estimates with a Gaussian distribution of the uncertainty. This works well for odometry, the attitude sensor and GPS, however, the uncertainty in the vision-based estimates is multi-modal and non-isotropic. This raises an interesting theoretical question on how to best fuse the vision data with the rest of the system. My ideas on how to approach this are presented in the Future Work section.

- **Path Planning**

  When visual localization is to be performed, the method works with whatever views are available from the current robot location. However, some positions provide better views of the nearby buildings than others. Thus, when generating trajectories for the robot, the path planner could be made to favor ones that allow for better visual localization.

  Further, the uncertainty in the robot position is decided to be too high based on a predefined threshold — the available free space around the robot is not taken into account. This information is, however, available from the path planner and can be used. By providing means of making it accessible by the *Localizer*, a more intelligent use of the visual localization method can be made. It would become possible to resort to the time consuming effort only when the robot is approaching an obstacle and correction is really necessary.

7.2 Discussion and Future Work

In this section, I would like discuss my vision on how the current limitations of my method could be approached.

7.2.1 View Planning

The problem of finding which model to use for visual pose estimation is very interesting and challenging. My approach to addressing it would be as follows:

Consider a Gaussian sphere around the center of projection of the camera. Discretize that sphere in a two-dimensional spherical grid of a certain resolution. Determine the set of relevant facade models that have some portion closer than a maximum usable range. Project each model onto that sphere, keeping track in each cell of the line numbers, face number, average viewing angle and average distance of the portion of the model that projects onto that cell. Replace the data in the cell with the current one only if the current distance is smaller than the existing data in the cell (known as Z-buffering). Repeat this for all relevant models.

For simplicity, assume that the camera field of view is a cone with its vertex at the center of projection. More accurate representations can be investigated as necessary.

Now, the view finding problem is to determine an orientation for the cone for which its intersection with the Gaussian sphere contains the most favorable configuration of projected model lines. Note that, there is no constraint that the lines come from the same facade model. The difficult part here is to define what "most favorable" means and this is a very challenging theoretical problem.

A practical solution could be this: Intuitively, the most favorable view is the

one that maximizes both the probability of a correct match between the models and the image and the accuracy of the pose estimate for the correct match. Factors that contribute to this are: the viewing angle of the facade, the distance to it and the number of model lines in view. Thus, a linear combination of these, multiplied by weights reflecting the importance of each factor, could be a good practical metric. The weights would need to be determined through experiments or simulations.

It is likely that this computation be slower than acceptable for each visual localization run because of the exhaustive two-dimensional search over the Gaussian sphere. This could be addressed by doing the computation upfront: The working area could be discretized into a grid of cells, each of which would hold a precomputed value of the best viewing direction for that cell. These values will be computed once for the entire site and then used instantaneously during any mission. Alternatively, if the site is too large for this to be practical, the viewing directions can be precomputed at discrete points of the planned trajectory before the mission starts.

7.2.2 Feature Matching

The feature matching procedure could be improved in two orthogonal ways. First, additional clues, such as color, could be used in the matching process. A color segmentation step can be performed on the image to roughly determine sections with identical color. If similar information is available in the model, this can serve as a means to coarsely register the image with the model. To avoid problems with portions of the building that have a strongly reflective surface, such as windows, these could be represented in the model by a special color that matches every other

color. After the coarse registration, the lines can be used for fine tuning by only searching for matches in the neighborhood of the projected model lines.

The second approach, which can be used in conjunction with the first one, is topological. Matching could be performed where higher-level features, such as window frames or decorative stones, could be sought and their relative positions taken into account. This can be very beneficial on buildings since they are typically designed to exhibit a very clear topological structure on their facades.

Using topology, some small but unique features could help disambiguate repetitive patterns. Further, by focusing on the unique or rare features first, a tremendous speedup could be achieved in finding the correct match as it may obviate the need for performing a lot of iterations in hope of exhausting all or most of the possible data association variations.

7.2.3 Partial Visual Localization

The idea behind partial visual localization is that some information from the image can be used even if the pose estimation method was unable to produce a reliable pose. This is typical for situations where the camera was way off target due to large orientation error from the a-priori pose estimate. In many such cases, however, due to the nature of the environment, the camera will still be looking at a building or other man-made structure.

A common property of the urban landscape is the dominance of the horizontal and vertical directions. Most of the building facades are vertical and mostly planar. An image of such a facade would exhibit a set of two strongly supported vanishing points, corresponding to the vertical and horizontal direction. The next

best vanishing point will be very weakly supported in comparison. This can be used to both detect such situations and recover the orientation of the robot.

The extension of the algorithm would be as follows: After the pose estimation algorithm finished without a reliable correct match, the extracted lines are used to determine the top three vanishing points. This is generally done by having each pair of edge lines vote for a vanishing point at their intersection and processing the resulting clusters of votes after that. The support for each of the vanishing points is noted. If there is a clear indication of strong support for the top two vanishing points and weak support for the last one, then this is an indication that the largest portion of the image is of a planar man-made structure. A further confirmation would be if the lines connecting the camera center with the two vanishing points form an approximately right angle.

In such a case, it would be easy to tell which is the horizontal and which is the vertical vanishing point. The local vertical direction for the robot can be approximately determined by the roll and pitch angles reported by the attitude sensor. Whichever vanishing point is very close to that direction is the vertical one. The other vanishing point should closely match the direction of the horizontal lines of the model used as the image was presumably of the same facade that was chosen by the view planner.

At this point, sufficient information is available to compute the pose of the camera and the robot. There is a correspondence between two of the main axes of the model and the camera coordinate systems. The third axes are the cross products of the first two. While this correspondence would not reveal the position of the robot, it fully recovers the orientation.

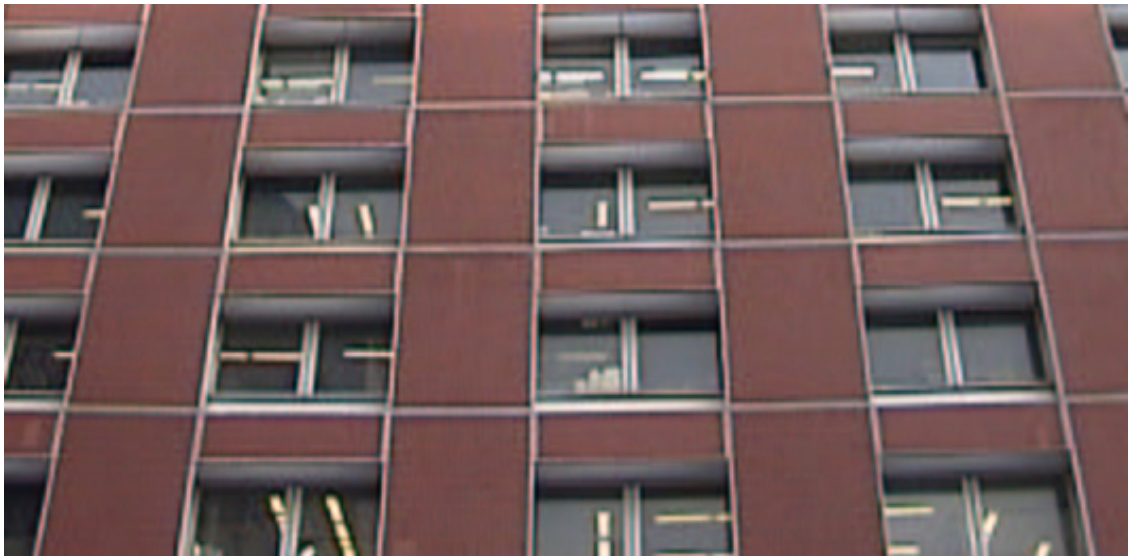


Figure 7.1: A building facade exhibiting repetitive feature patterns

Finally, if huge error in orientation was the reason for failure of the pose estimation method, now that a better orientation estimate is available, it can be repeated with higher chances of success.

7.2.4 Visual Pose Integration

The problem of the integration of the vision-based pose estimates with the rest of the sensors is the very different nature of the uncertainty associated with those estimates. While using odometry, the attitude sensor and GPS, the localization problem is to mainly track the motion of the robot as there is only one most likely hypothesis about its current pose.

This is not the case with the vision-based pose estimation. Very often multiple likely hypotheses can result from running that algorithm. This is especially true if the facade of the building being used has a large number of the same repetitive features, such as equally spaced identical windows (Figure 7.1).

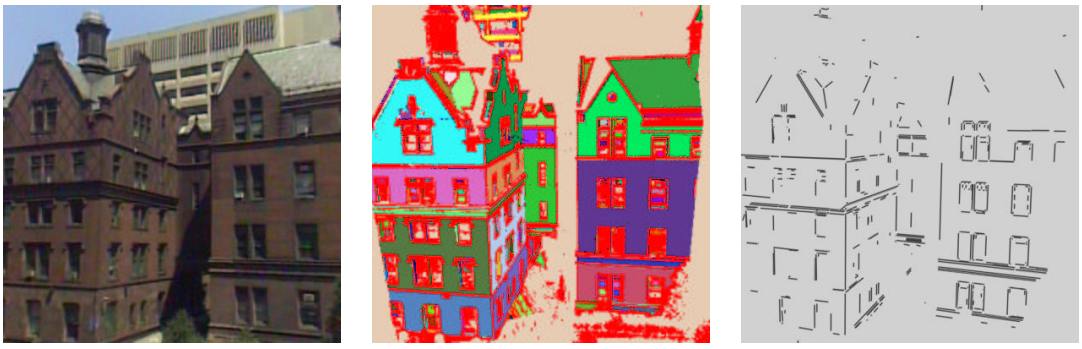
In cases like this, it would be desirable to consider all of the available hypotheses until sufficient evidence is obtained of which one is actually true. Such evidence could either be a number of runs of the visual localization algorithm from different points of view or a pose update from the other sensors the exclude all but the one true hypothesis.

Unfortunately, the Extended Kalman Filter framework does not allow for easy integration of a multi-modal probability distribution. This implies that a more general probabilistic framework is used.

When vision-based pose estimation is performed, the method could be modified as follows: The standard RANSAC loop described in chapter 5 is performed, however, the resulting pose of each iteration is used as a sample of the conditional probability distribution of the robot pose given the current image. At the end of the loop, this probability can be fused in a Bayesian fashion with the Gaussian distribution from the EKF. The outstanding peaks in the resulting distribution could be used to form hypotheses which will be tracked by their own EKF until invalidated by further evidence, such as large innovation series or negative values in the variance-covariance matrix.

7.2.5 Obtaining the Model

The idea on how to obtain the environmental model used in the visual localization method is tightly coupled to the intended use of the localization method. Recall from Chapter 3 that the work in this thesis is part of the AVENUE project whose goal is the creation of a detailed geometric and photometric 3-D model of urban site. I will refer to this detailed model as the *detailed model*, as opposed to the



(a) (b) (c)

Figure 7.2: Model acquisition and simplification: (a) an image of a building; (b) the 3D model created from the image and a range scan; (c) a reduced version of the same model consisting only of line features.

localization model used for the visual localization.

The detailed 3-D models that obtained from the range scans and images of the buildings are too large and complex since they capture a lot of detail (Figure 7.2b). The modeling process is incremental. At each stage there is a partial model of the site available. By partial, I mean that the detailed model consists of usable models of certain building facades plus various artifacts of the modeling process that correspond to yet unmodeled portions of the site. These artifacts are easily recognized as such since they are tagged as "unknown" by the modeling system.

During the scan/image registration process and their integration with the existing partial detailed model, a data simplification step is done which creates a reduced complexity model. Its generated for the purpose of the registration of the range scanner's and the camera's coordinate systems. This simplified model (Figure 7.2c) consists of 3-D line segments obtained by segmenting the range scans

into planar regions and intersecting planes to obtain lines (for details, see [Stamos and Allen, 2000]). The result of this operation is a set of lines segments — the kind that is needed for visual localization.

Thus, to create a localization model, only a little post-processing is needed of the available 3-D line features. First, the features are classified into near-planar regions. That is, for each vertical face of the scanned building a representative plane is established which defines its own coordinate system. All line features, which typically correspond to window frames, ledges or decorations, are referenced with respect to their representative plane. Then, for each plane (modeled facade), the number of features used are reduced by keeping only the longest few of them. Finally, each resulting set of line segments is stored in the localization model data base.

Note that there is no controversy here about which model comes first (the bootstrapping problem). The robot will start from a certain location, scan the buildings in view, create their partial detailed models and register them with its original pose. As a result, localization models of some of the scanned facades will be also available which will allow the robot to go further, possibly using the available so far localization models. As it obtains new scans and images and updates the detailed model, new localization models will become also available which can be used to localize the robot as it goes farther along its modeling path.

Appendix A

Implementation Details of the System Architecture

A.1 Overview

The implementation of the software architecture (Figure 3.4) is based on the Common Object Request Broker Architecture (CORBA) and Mobility — a robot integration software framework developed by the robot manufacturer, Real World Interface, Inc.

The Mobility Robot Object Model is an object-oriented approach to modeling a robot as "a collection of dynamically connected objects". It defines interfaces and provides implementations of interfaces and objects needed to build robot software as a set of distributed components.

A component is the main software building block which represents an abstraction of a hardware piece, such as a sensor or an actuator, a behavior, or a perceptual schema. It is defined by an interface written in CORBA's Interface Def-

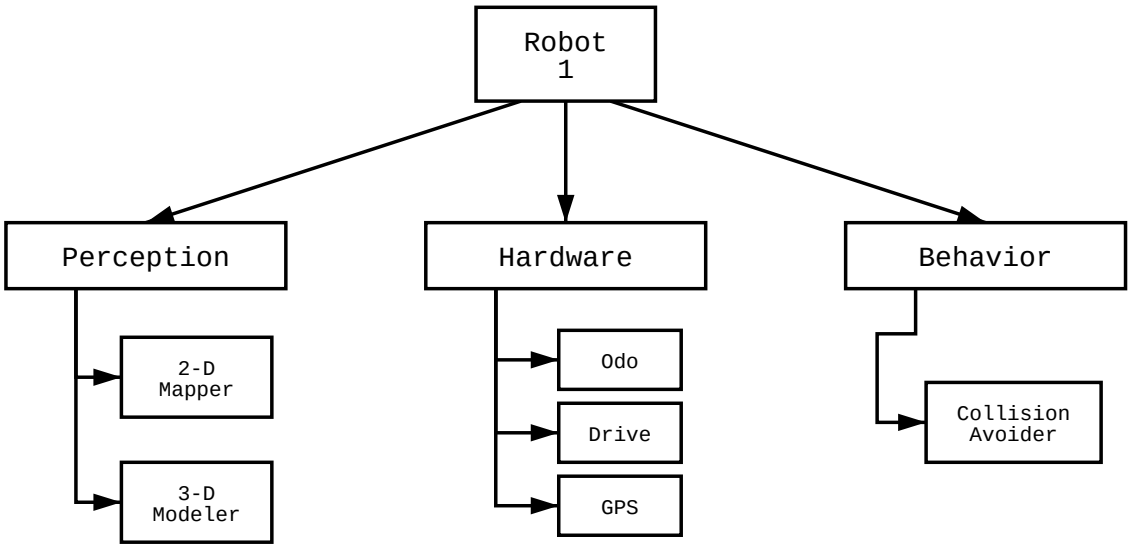


Figure A.1: An example for software component hierarchy

inition Language (IDL) and typically implemented as a class if an object-oriented language is used.

Some components, called containers, have the extra ability to contain other components. This allows for the components to be arranged in a hierarchical way. A robot is represented as a tree of components as shown in Figure A.1.

Components refer to one another by name but to actually execute a remote call CORBA needs to know the object IOR (Interoperable Object Reference). The name-to-IOR translation for each component is done by its parent container. The top-level containers are registered with a name server which is part of the standard CORBA package. The IOR of the name server is published at a configurable web URL.

To find a specified object by name, first the IOR of the name server is obtained through an http request. Then the name server is queried for the IOR

of the top-level container of the component. Finally, the container is queried for the object. To facilitate programming, this procedure is implemented in the core Mobility package.

Data stream sources are implemented by using state components. A state component stores a list (implemented as a queue) of the data records that have been passed along. Each data record is tagged with the time of its creation and the list is sorted in a chronological order. To limit the use of memory, only the last few data records are kept, however, that number is configurable.

The state components also maintain a list of the IORs of registered *observers* — components that are interested in receiving a notification when new data becomes available. The interface of the state component allows for an observer to add or remove its IOR from the list. When a new data record becomes available, the state component iterates through this list and notifies each observer via a CORBA IPC call (a server push approach).

In addition to using streams, a component may also expose an additional command interface to provide access to a set of specific commands. This is done by implementing a standard hook method which accepts as an argument a single string. When this method is invoked, the component parses the string and executes the command. This is how the emergency commands **stop** and **pause** of the *Navigator* have been implemented, for example.

The sections that follow describe the interfaces of the system components in the architecture. The hardware servers are discussed in the next section after which the higher-level navigation components are presented.

A.2 Description of the Hardware Servers

A.2.1 ATRV2Server

The *ATRV2Server* is the interface to the robot hardware and comes with the standard Mobility distribution. It consists of several components that represent its sensors and actuators or provide general robot-specific information, such as shape and dimensions. Two components of main interest are *Odo*, which provides position and velocity, and *Drive*, which drives the robot.

The interface to the *Drive* module consist of a pair of floating point numbers specifying the velocities of the two axes — translation and rotation. The first number designates the translational velocity of the robot in meters per second. The second number is the target angular velocity.

The *Odo* component is the source of a data stream consisting of triplets of floating point numbers. The triplet corresponds to current location coordinates x and y and orientation θ. This module has been modified to incorporate the systematic error corrections discussed in section 4.1.1.

A.2.2 PTUServer

The *PTUServer* controls the orientation of the camera by turning the pan-tilt unit. It provides a command sink to which three types of commands can be supplied. The command types are to pan, tilt or both pan and tilt. Each command supports two variants — one for which the target is specified in absolute angles, and another, for which the target is specified relative to the current location.

This server also provides a status object that allows components to monitor

its state. The state consists of a flag that specifies whether the PTU head is currently in motion and two numbers which are the current pan and tilt positions.

A.2.3 AttitudeServer

The *AttitudeServer* parses the data coming out of the digital compass module and provides the current orientation in both east-north-up and local x-y-z coordinates. This is done by two state components — one for each reference frame. Each of these state components have three fields which contain the coordinates with respect to the corresponding reference frame. Additional fields contain the strength of the magnetic field, and the alarm status.

The server also implements a **calibrate** command that is used to specify the orientation of the local reference frame.

A.2.4 GPSServer

The *GPSServer* parses the data from the GPS receiver and makes it available to other components.

Position fixes are provided in various formats. Two of them are absolute: longitude-latitude-altitude and X-Y-Z with respect to the WGS-84 coordinate reference frame [Hofman-Wellenhof *et al.*, 2001]. The other two are local: east-north-up and x-y-z with respect to a user-defined coordinate system (see section 4.3.2). Each of these formats is provided by a separate state component so interested observers can subscribe only to the one that they find most useful.

In addition to the three position coordinates, various other data is provided. It includes the standard deviation of the position fixes, the number of satellites

used, current RTK mode (float or fixed), current GPS time, horizontal and vertical dilution of precision.

The server also has a **calibrate** command which is used to establish the local reference frame by specifying it explicitly or referring to a file containing this information.

A.2.5 VideoServer

The images taken by the camera are grabbed and distributed by the *VideoServer*. They are used for both localization and modeling.

This server can operate in two modes: single grab and continuous. In the single grab mode, it waits for a **grab** command to be send, in order to capture an image. In continuous mode, it captures images continuously at a specified frame rate.

The images are distributed by two state components: The first one provides raw image data which is used for image processing. The second component scales the image down by sampling every second pixel and compresses the result before supplying it to the observers. This is useful for video streaming to the user interface over a slow wireless network to obtain visual feedback of the progress of the mission.

A.2.6 ScanServer

The *ScanServer* is designed to control the scanner and make the range scans and images available to the other components. Unfortunately, it is not yet implemented, because of unavailable information regarding its proprietary interface.

A.3 Description of NavServer

The *NavServer* is a program that consists of three components: *Localizer*, the *Controller* and the *Navigator*.

A.3.1 Localizer

The *Localizer* registers as an observer to the *Odo*, *Attitude*, *GPS* and *Image* components. Each time a new measurement is made, it gets notified about it, obtains it from the corresponding state component and integrates it into the current pose estimate according to the methods described in chapters 4 and 5.

The *Localizer* contains two state components that provide information about the current pose estimate. The first component supplies the current pose as a six-dimensional vector consisting of the three coordinates and the three Euler angles. The second component provides the coefficients of the variance-covariance matrix for observers interested in the uncertainty.

The coordinates provided by the *Localizer* are with respect to the established coordinate system for the site. A **calibrate** command is used to establish or change that reference coordinate system.

A.3.2 Controller

The *Controller* provides a command interface to both the *Localizer* and the *Navigator*. Its commands set consists of variations of **goto** and **turnto** as well as the *PTUServer* commands. The variations of the **goto** and **turnto** commands are based on two parameters. The first one, that determines whether the target is spec-

ified in absolute or relative coordinates. The second parameter, determines whether the robot should be moving forward or backward. The second case is irrelevant to the navigation process but may be useful to a remote user looking at the video stream from the camera.

The *Controller* obtains feedback by registering as an observer with the *Localizer*. New pose updates are used to update the commands sent to the actuators according to the method described in section 3.4.

The *Controller* provides feedback by a state component that has a number of fields: a flag that specifies whether the *Controller* is busy or idle; another flag specifying whether the *Controller* has reached its target, is moving, has been stopped or has been paused; the current command being executed; the target parameters for the command (e.g. the target robot pose it is pursuing); and the amount of work remaining (e.g. distance to the target).

A.3.3 Navigator

The *Navigator* handles most of the communication of the robot with remote components. Its functions are different depending on the direction of the data/command traffic.

Commands sent to the robot are represented by a sequence of numbers, the first of which is the code for the command and the rest of them are the arguments. A mission is a list of such commands. The *Navigator* provides a state component which is used to store the current mission. New commands sent to the *Navigator* are appended to the end of the list.

Another state component is used to allow remote components to monitor the

progress of the mission. This state component consists of three fields: A flag that determines if the robot is busy or not; a flag that determines if the mission has been completed, paused, aborted, or is in progress; and the index in the mission list of the command currently being executed. Thus, when a new component connects to the *Navigator*, by looking at its state and the mission command list, it can determine what the robot was doing recently and what it is about to do.

The *Navigator* also registers as an observer and mirrors the data coming from the the *Localizer*, the state component of the *Controller* and, optionally, any other on-board data source. It forwards this data to remote components at a reduced data rate using a separate independent thread. It insulates this way the important core components from interactions with remote peers that may be time consuming and problematic because of networking issues.

Bibliography

[Adam *et al.*, 1999] Amit Adam, Ehud Rivlin, and Hector Rotstein. Fusion of fixation and odometry for vehicle navigation. In *Proceedings of IEEE International Conference on Robotics and Automation in Detroit, Michigan*, pages 1638–1643, 1999.

[Allen *et al.*, 2001] Peter Allen, Ioannis Stamos, Atanas Gueorguiev, Ethan Gold, and Paul Blaer. AVENUE: Automated site modeling in urban environments. In *Proceedings of 3rd Conference on Digital Imaging and Modeling in Quebec City, Canada*, pages 357–364, May 2001.

[Antone and Teller, 2000a] Matthew Antone and Seth Teller. Automatic recovery of camera positions in urban scenes. Technical Report MIT LCS TR-814, Laboratory for Computer Science, MIT, December 2000.

[Antone and Teller, 2000b] Matthew Antone and Seth Teller. Automatic recovery of relative camera rotations for urban scenes. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages I:282–289, 2000.

[Aono *et al.*, 1998] Toshihiro Aono, Kenjiro Fujii, Shintaro Hatsumoto, and Takayuki Kamiya. Positioning of vehicle on undulating ground using GPS and

dead reckoning. In *Proceedings of IEEE International Conference on Robotics and Automation in Leuven, Belgium*, pages 3443–3448, 1998.

[Arkin, 1987] Ronald Arkin. Motor schema based navigation for a mobile robot: An approach to programming by behavior. In *Proceedings of IEEE International Conference on Robotics and Automation in Raleigh, North Carolina*, volume 2, pages 264–271, 1987.

[Bar-Shalom, 1987] Yaakov Bar-Shalom. *Tracking and Data Association.* Academic Press, 1987.

[Basri *et al.*, 1999] Ronen Basri, Ehud Rivlin, and Ilan Shimshoni. Image-based robot navigation under the perspective model. In *Proceedings of IEEE International Conference on Robotics and Automation in Detroit, Michigan*, pages 2578–2583, 1999.

[Borenstein and Feng, 1995] Johann Borenstein and Liqiang Feng. Correction of systematic odometry errors in mobile robots. In *IROS'95, Pittsburgh, Pennsylvania*, pages 569–574, August 1995.

[Borenstein and Feng, 1996a] Johann Borenstein and Liqiang Feng. Gyrodometry: A new method for combining data from gyros and odometry in mobile robots. In *Proceedings of IEEE International Conference on Robotics and Automation in Minneapolis, Minnesota*, pages 423–428, 1996.

[Borenstein and Feng, 1996b] Johann Borenstein and Liqiang Feng. *Navigating Mobile Robots: Systems and Techniques.* A. K. Peters, 1996.

[Bozorg *et al.*, 1998] Mohammad Bozorg, Eduardo Nebot, and Hugh Durrant-Whyte. A decentralised navigation architecture. In *Proceedings of IEEE International Conference on Robotics and Automation in Leuven, Belgium*, pages 3413–3418, 1998.

[Brooks, 1986] Rodney Brooks. A robust layered control system for a mobile robot. *Journal of Robotics and Automation*, 2(1):14–23, March 1986.

[Brown and Hwang, 1997] Robert Brown and Patrick Hwang. *Introduction to random signals and applied Kalman filtering.* John Wiley & Sons, 3rd edition, 1997.

[Castellanos and Tardos, 1999] Jose Castellanos and Juan Tardos. *Mobile Robot Localization and Map Building: A Multisensor Fusion Approach.* Kluwer Academic Publishers, 1999.

[Castellanos *et al.*, 1998] J. A. Castellanos, J. M. Martinez, J. Neira, and J. D. Tardos. Simultaneous map building and localization for mobile robots: A multisensor fusion approach. In *Proceedings of IEEE International Conference on Robotics and Automation in Leuven, Belgium*, pages 1244–1249, 1998.

[Chen and Shibasaki, 1999] Tienen Chen and R. Shibasaki. High precision navigation for 3-D mobile GIS in urban area by integrating GPS, gyro and image sequence analysis. In *Proceedings of the International Workshop on Urban 3D/Multi-media Mapping, Tokyo, Japan*, pages 147–156, 1999.

[Cooper and Durrant-Whyte, 1994] S. Cooper and H. F. Durrant-Whyte. A Kalman filter model for GPS navigation of land vehicles. In *Proceedings of IEEE*

International Conference on Robotics and Automation in San Diego, California, pages 157–163, 1994.

[Cox, 1988] I. J. Cox. Blanche: An autonomous robot vehicle for structured environments. In *Proceedings of IEEE International Conference on Robotics and Automation in Philadelphia, Pennsylvania*, pages 978–982, 1988.

[Crowley, 1985] James L. Crowley. Navigation for an intelligent mobile robot. In *Journal of Robotics and Automation*, pages 31–41, 1985.

[Dellaert *et al.*, 1999] Frank Dellaert, Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Monte Carlo localization for mobile robots. In *Proceedings of IEEE International Conference on Robotics and Automation in Detroit, Michigan*, pages 1322–1328, 1999.

[Dissanayake *et al.*, 1999] Gamini Dissanayake, Salah Sukkarieh, Eduardo Nebot, and Hugh Durrant-Whyte. A new algorithm for the alignment of inertial measurement unit without external observation for land vehicle applications. In *Proceedings of IEEE International Conference on Robotics and Automation in Detroit, Michigan*, pages 2274–2279, 1999.

[D'Souza, 1988] A. Frank D'Souza. *Design of Control Systems*. Prentice-Hall, 1988.

[Durrant-Whyte *et al.*, 1999] Hugh Durrant-Whyte, Gamini Dissanayake, and Peter Gibbens. Toward deployment of large scale simultaneous localization and map building (SLAM) systems. In *Proceedings of International Simposium on Robotics Research in Salt Lake City, Utah*, pages 121–127, 1999.

[Dyson, 2002] Dyson. http://www.dyson.co.uk, 2002.

[Eureka, 2001] The Eureka Company.
http://www.eureka.com/whatsnew/robotvacupdate.htm, 2001.

[Everett, 1995] H. R. Everett. *Sensors for Mobile Robots: Theory and Application.* A. K. Peters, Wellesley, Massachusetts, 1995.

[Fischler and Bolles, 1980] M.A. Fischler and R.C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. In *DARPA*, pages 71–88, 1980.

[Fuke and Krotkov, 1996] Yasutaka Fuke and Eric Krotkov. Dead reckoning for a lunar rover on uneven terrain. In *Proceedings of IEEE International Conference on Robotics and Automation in Minneapolis, Minnesota*, pages 411–416, 1996.

[Gold, 2001] Ethan Gold. AvenueUI: A comprehensive visualization/teleoperation application and development framework for multiple mobile robots. Master's thesis, Columbia University, Department of Computer Science, 2001.

[Gueorguiev *et al.*, 2000] Atanas Gueorguiev, Peter K. Allen, Ethan Gold, and Paul Blaer. Design, control and architecture of a mobile site-modeling robot. In *Proceedings of IEEE International Conference on Robotics and Automation in San Francisco, California*, pages 3266–3271, 2000.

[Hofman-Wellenhof *et al.*, 2001] Bernhard Hofman-Wellenhof, Herbert Lichtenegger, and James Collins. *Global Positioning System, Theory and Practice.* Springer-Verlag, Wien, 5th edition, 2001.

[Horn, 1986] Berthold Klaus Paul Horn. *Robot Vision.* The MIT Press, Cambridge, Massachusetts, 1986.

[Horn, 1990] Berthold Klaus Paul Horn. Relative orientation. *International Journal of Computer Vision*, 4(1):59–78, 1990.

[Kortenkamp *et al.*, 1998] David Kortenkamp, R. Peter Bonasso, and Robin Murphy, editors. *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems.* AAAI Press / The MIT Press, 1998.

[Kosaka and Kak, 1992] A. Kosaka and A. C. Kak. Fast vision-guided mobile robot navigation using model-based reasoning and prediction of uncertainties. *Computer Vision, Graphics, and Image Processing — Image Understanding*, 56(3):271–329, 1992.

[Kotani *et al.*, 1998] S. Kotani, K. Kaneko, T. Shinoda, and H. Mori. Mobile robot navigation based on vision and DGPS information. In *Proceedings of IEEE International Conference on Robotics and Automation in Leuven, Belgium*, pages 2524–2529, 1998.

[Krose and Bunschoten, 1999] B. J. A. Krose and R. Bunschoten. Probabilistic localization by appearance models and active vision. In *Proceedings of IEEE International Conference on Robotics and Automation in Detroit, Michigan*, pages 2255–2260, 1999.

[Kumar and Hanson, 1994] Rakesh Kumar and Allen R. Hanson. Robust methods for estimating pose and a sensitivity analysis. *Computer Vision Graphics and Image Processing*, 60(3):313–342, November 1994.

[Leonard and Feder, 1999] John Leonard and Hans Jacob S. Feder. A computationally efficient method for large-scale concurrent mapping and localization. In

Proceedings of International Simposium on Robotics Research in Salt Lake City, Utah, pages 128–135, 1999.

[Levitt *et al.*, 1987] Tod Levitt, Daryl Lawton, David Chelberg, and Philip Nelson. Qualitative navigation. In *Proceedings of DARPA Image Understanding Workshop*, pages 447–465, 1987.

[Lin and Tummala, 1997] Cheng-Chih Lin and R. Lal Tummala. Mobile robot navigation using artificial landmarks. *Journal of Robotics Systems*, 14(2):93–106, 1997.

[Maeyama *et al.*, 1996] Shoichi Maeyama, Nobuyuki Ishikawa, and Shinichi Yuta. Rule-based filtering and fusion of odometry and gyroscope for a fail-safe dead reckoning system of a mobile robot. In *IEEE/SICE/RSJ International Conference on Multisensor Fusion and Integration for Intelligent Systems*, pages 541–548, 1996.

[Maybeck, 1982a] Peter Maybeck. *Stochastic Models, Estimation and Control*, volume 141-2 of *Mathematics in Science and Engineering*. Academic Press, New York, 1982.

[Maybeck, 1982b] Peter Maybeck. *Stochastic Models, Estimation and Control*, volume 141-3 of *Mathematics in Science and Engineering*. Academic Press, New York, 1982.

[Mobility, 1999] Real World Interface. *Mobility 1.1: Robot Integration Software, User's Guide*, 1999.

[Moravec, 1983] Hans Moravec. The Stanford cart and the CMU rover. *Proceedings of IEEE*, 71:872–884, July 1983.

[Nayak, 2000] Rakesh Nayak. Reliable and continuous urban navigation using multiple GPS antennas and a low cost IMU. Master's thesis, Department of Geomatics Engineering, University of Calgary, Canada, 2000.

[Nayar *et al.*, 1994] Shree K. Nayar, Hiroshi Murase, and Sameer Nene. Learning, positioning, and tracking visual appearance. In *Proceedings of IEEE International Conference on Robotics and Automation in San Diego, California*, pages 3237–3246, 1994.

[Neira *et al.*, 1999] Jose Neira, Juan Tardos, Joachim Horn, and Gunther Schmidt. Fusing range and intensity images for mobile robot localization. *IEEE Transactions on Robotics and Automation*, 15(1):76–84, February 1999.

[Nilsson, 1984] Nils Nilsson. Shakey the robot. Technical Report 323, SRI AI Center, April 1984.

[Park and Kender, 1995] Il-Pyung Park and John Kender. Topological direction-giving and visual navigation in large environments. *Artificial Intelligence*, 78:355–395, 1995.

[Press *et al.*, 1992] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 2nd edition, 1992.

[Robomower, 2000] Friendly Robotics. http://www.friendlyrobotics.com, 2000.

[Roumeliotis *et al.*, 1999] Stergios I. Roumeliotis, Gaurav S. Sukhatme, and George A. Bakey. Circumventing dynamic modeling: Evaluation of the error-state Kalman Filter applied to mobile robot localization. In *Proceedings of IEEE International Conference on Robotics and Automation in Detroit, Michigan*, pages 1656–1663, 1999.

[Roy and Thrun, 1999] Nicholas Roy and Sebastian Thrun. Online self-calibration for mobile robots. In *Proceedings of IEEE International Conference on Robotics and Automation in Detroit, Michigan*, pages 2292–2297, 1999.

[Schilling, 1990] Robert Schilling. *Fundamentals of Robotics: Analysis and Control.* Prentice-Hall, Englewood Cliffs, New Jersey, 1990.

[Schleppe, 1996] John Schleppe. Development of a real-time attitude system using a quaternion parametrization and non-dedicated GPS receivers. Master's thesis, Department of Geomatics Engineering, University of Calgary, Canada, 1996.

[Sim and Dudek, 1998] Robert Sim and Gregory Dudek. Mobile robot localization from learned landmarks. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems in Victoria, BC, Canada*, October 1998.

[Simmons, 1994] Reid Simmons. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation*, 10(1):34–43, February 1994.

[Stamos and Allen, 2000] Ioannis Stamos and Peter K. Allen. Integration of range and image sensing for photorealistic 3D modeling. In *Proceedings of IEEE International Conference on Robotics and Automation in San Francisco, California*, pages II:1435–1440, 2000.

[Stamos, 2001] Ioannis Stamos. *Geometry and Texture Recovery of Scenes of Large Scale: Integration of Range and Intensity Sensing.* PhD thesis, Columbia University, New York, 2001.

[Sukkarieh *et al.*, 1998] S. Sukkarieh, E. Nebot, and H. F. Durrant-Whyte. Achieving integrity in an INS/GPS navigation loop for autonomous land vehicle applications. In *Proceedings of IEEE International Conference on Robotics and Automation in Leuven, Belgium*, pages 3437–3442, 1998.

[Taylor and Kriegman, 1995] Camillo J. Taylor and David J. Kriegman. Structure and motion from line segments in multiple images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(11):1021–1033, November 1995.

[Taylor and Kriegman, 1998] Camillo J. Taylor and David J. Kriegman. Vision-based motion planning and exploration algorithms for mobile robots. *IEEE Transactions on Robotics and Automation*, 14(3):417–427, June 1998.

[Taylor *et al.*, 1996] Camillo J. Taylor, Paul E. Devebec, and Jitendra Malik. Reconstructing polyhedral models of architectural scenes from photographs. In *Proceedings of the European Conference on Computer Vision*, pages 659–668, May 1996.

[Thorpe, 1990] Charles E. Thorpe, editor. *Vision and Navigation: The Carnegie Mellon Navlab.* Kluwer Academic Publishers, 1990.

[Thrun *et al.*, 1998] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. A probabilistic approach to concurrent mapping and localization for mobile robots. *Autonomous Robots*, 5:253–271, 1998.

[Thrun *et al.*, 1999] Sebastian Thrun, Maren Bennewitz, Wolfram Burgard, Armin Cremers, Frank Dellaert, and Dieter Fox. MINERVA: A second-generation museum tour-guide robot. In *Proceedings of IEEE International Conference on Robotics and Automation in Detroit, Michigan*, pages 1999–2005, 1999.

[Thrun, 1998] Sebastian Thrun. Finding landmarks for mobile robot navigation. In *Proceedings of IEEE International Conference on Robotics and Automation in Leuven, Belgium*, pages 958–963, 1998.

[Tsai, 1987] Roger Y. Tsai. A versatile camera calibration technique for high-accuracy 3-D machine vision metrology using off-the-shelf TV cameras and lenses. *Journal of Robotics and Automation*, RA-3(4):323–344, 1987.

[von der Hardt *et al.*, 1996] Hans-Joachim von der Hardt, Didier Wolf, and Rene Husson. The dead reckoning localization system of the wheeled mobile robot ROMANE. In *IEEE/SICE/RSJ International Conference on Multisensor Fusion and Integration for Intelligent Systems*, pages 603–610, 1996.

[Zheng and Tsuji, 1992] Jiang Yu Zheng and Saburo Tsuji. Panoramic representation for route recognition by a mobile robot. *International Journal of Computer Vision*, 9(1):55–76, 1992.