Localization Methods for a Mobile Robot in Urban Environments

Atanas Georgiev and Peter K. Allen
Computer Science Department, Columbia University, New York, NY
{atanas,allen}@cs.columbia.edu

Abstract—
This paper addresses the problems of designing, building and using mobile robots for urban site modeling. It presents work on both system and algorithmic aspects. On the system level, we have designed and built a functioning autonomous mobile robot. The design extends an existing robotic vehicle with a sensor suite consisting of a digital compass with an integrated inclinometer, a global positioning unit, and a camera mounted on a pan-tilt head. The system is controlled by a distributed software architecture for mobile robot navigation and site modeling. On the algorithmic level, we have developed a localization system that employs two methods. The first method uses odometry, the compass module and the global positioning sensor. An extended Kalman filter integrates the sensor data and keeps track of the uncertainty associated with it. The second method is based on camera pose estimation. It is used when the uncertainty from the first method becomes very large. The pose estimation is done by matching linear features in the image with a simple and compact environmental model. We have demonstrated the functionality of the robot and the localization methods with real-world experiments.

Index Terms—Mobile robots, localization, computer vision

I. INTRODUCTION

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 alleviate, and possibly completely take over, tedious, monotonous and painstaking tasks. Examples include surveying, mapping, modeling, transportation and delivery, security, fire safety, and recovery from disasters. Some level of autonomy is envisioned in our cars and the wheelchairs of disabled people. Yet, after more than three decades of research, we still do not see many robots scurrying about and helping us in our daily chores.

The problem of building a functional autonomous mobile robot that can successfully and reliably interact with the real-world is very difficult. It involves a number of issues such as proper design, choice of sensors, methods for localization, navigation, planning, and others,—each of which is a challenge on its own. The environment of operation also plays an important role. Much of the existing research has been focused on solving these issues indoors because of its slightly more predictable nature (e.g., flat, horizontal, well-structured, smaller scale). On the other hand, many of the interesting applications are outdoors where fewer assumptions can be taken for granted.

In this paper, we address the problem in outdoor urban environments. These kinds of environments pose their own unique set of challenges that differentiate them from both the indoor and the open-space outdoor landscapes. On the one hand, they are usually too large to consider applying certain techniques that achieved success indoors. On the other hand, typical outdoor sensors, such as GPS, have problems with reception around buildings.

While we have tried to keep the methods presented here general, we have focused on the development of our mobile robot system with a specific application in mind. The AVENUE project at the Columbia University Robotics Laboratory targets the automation of the urban site modeling process [1]. The main goal is to build geometrically accurate 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.

High-quality site models are needed in a variety of applications, such as city planning, urban design, fire and police planning, historical preservation and archaeology, virtual and augmented reality, geographic information systems and many others. However, they are typically created by hand which is extremely slow and error prone. The models built are often incomplete and updating them can be a serious problem. 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 and Allen [2]. They developed a compre-
hensive system for the automated 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. They demonstrated the feasibility of their approach by creating models of various buildings. Additionally, a safe path planner based on Voronoi diagrams has been developed in our group by Paul Blaer [3].

Here, we are addressing the remaining aspects of this list. We are introducing an autonomous mobile platform operating according to the following scenario: Its task is to go to desired locations and acquire requested 3-D scans and images of selected buildings. The locations are determined by the sensor planning system and are used by the path planning system to generate reliable trajectories which the robot follows. When the rover arrives at the target location, it uses the sensors to acquire the scans and images and forwards them to the modeling system. The modeling system registers and incorporates the new data into the existing partial model of the site (which in the beginning could be empty). After that, the view planning system decides upon the next best data acquisition location and the above steps repeat. The process starts from a certain location and gradually expands the area it has covered until a complete model of the site is obtained.

The design and implementation of our mobile platform involved efforts that are related and draw from a large amount of existing work. For localization, dead reckoning has always been attractive because of its pervasiveness [4]–[6]. With the rapid development of technology, GPS receivers are quickly becoming the sensor of choice for outdoor localization [7]–[9]. Imaging sensors, such as CCD cameras and laser range finders, have also become very popular mobile robot components [10]–[13]. Various methods for sensor integration and uncertainty handling have been proposed [14]–[18]. A very popular and successful idea is to exploit the duality between localization and modeling and address both issues in the same process, known as SLAM – Simultaneous Localization and Map Building [16], [17], [19], [20]. Sensors and methods for indoor localization have been comprehensively reviewed in two books [21], [22]. Another excellent book presents case studies of successful mobile robot systems [23].

Researchers from the Australian Centre for Field Robotics have made significant progress towards using SLAM in outdoor settings. Dissanayake et al have proved that a solution to the SLAM problem is possible and presented one such implementation [24]. Guivant et al have further looked into optimizing the computational aspects of their algorithm and have applied it to an unstructured natural environment [25].

Chen and Shibasaki have noticed that GPS accuracy becomes poor for navigation in many areas [26]. 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.

Nayak has specifically addressed localization in urban environments by a sensor suite consisting of four GPS antennae and a low-cost inertial measurement unit [27]. 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.

Our approach delivers a functional mobile robot system capable of operating accurately under the challenges of urban environments. Whenever needed, we are making use of unique urban characteristics 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 rest of this paper is organized as follows: Next is a section that describes the work and considerations that went into the design, the architecture and the various components of our mobile platform. Section III describes the first of our localization methods, based on odometry, a digital compass module and global positioning. Section IV presents our vision-based localization methods. Experimental results are shown in section V and in section VI, we conclude with a summary and a discussion on future extensions of this work.

II. SYSTEM DESIGN AND IMPLEMENTATION

The design of a mobile robot for urban site modeling involves a myriad of issues related to both its safe naviga-
tion and the necessary data acquisition. Some of the most important issues are: the choice of a robotic vehicle, the choice of sensors, the sensor placement, the software system architecture, and the means of remote communication with the robot. Ideally, all of these 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 with our mobile platform and the scanning equipment.

A. Hardware

The mobile robot we used as a test bed for this work is an ATRV-2 model manufactured by iRobot (Figure 1). It has built-in odometry and twelve sonars. It has a regular PC on-board and provides a power supply which amounts to about three hours of standalone operation. Its payload of 118 kg is enough to accommodate the scanner, its electronics controller box, and the additional periphery we have installed. The robot can move as fast as $2m/s$ 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 sensor suite on the robot has to provide sufficient data to allow for safe navigation and accurate modeling. The modeling method already developed dictates the use of a 3-D range finder and a color camera. Both are provided by a Cynax 2500 laser scanner that has a nominal accuracy of $6mm$ and range of up to $100m$.

For successful navigation, the robot needs to be able to detect its position in the environment. An essential requirement is that the robot has an idea of its location at any given moment in time. This is best accomplished by proprioceptive sensors — ones that allow for pose computation based only on internal measurements of the robot motion. Odometry is typically used as it is provided in almost every mobile robot, requires little computation, and works in real-time.

To help reduce some of the error accumulation problems with odometry and obtain a better estimate of the robot orientation, we have added a Honeywell HMR3000 digital compass module, which includes an integrated roll-pitch sensor and provides fast update rates of up to $20Hz$. Its roll-pitch accuracy is $0.6^\circ$ and its heading accuracy is better than $1.5^\circ$ (root-mean-square).

An inherent limitation of proprioceptive and dead-reckoning sensors on mobile robots is their unbounded error accumulation. It can only be alleviated by the addition of appropriate exteroceptive sensors — ones that make observations of the robot environment. For outdoor operation, global positioning sensors are particularly attractive, since they are explicitly designed for this purpose and have the necessary infrastructure already in place. We have equipped our robot with an Ashtech GG24C GPS+GLONASS receiver which is accurate down to $1cm$ in real-time kinematic (RTK) mode.

The combination of dead-reckoning and GPS has been proven very beneficial. GPS is known to exhibit an unstable high-frequency behavior manifested by sudden “jumps” of the position estimates, especially 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 sufficient data is collected and errors are “averaged out”. On the other hand, dead-reckoning 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 space, however, it often fails in urban areas. Tall buildings in the vicinity tend to obstruct the clear view to the satellites and the signals of fewer satellites reach the receiver (Figure 2, 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 2, 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 this is the reason we have added a CCD camera to make up for the limitations of the above sensors. The camera is mounted on a pan-tilt unit (PTU) which provides two degrees of freedom (independent of the robot) for its orientation.

We should mention 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 trade-off between speed, accuracy, and range. 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.

Finally, communication with the robot is done through a wireless network. Two $11Mbit/s$ wireless IEEE 802.11b interfaces have been added. The first one was designed to take advantage of a wireless network infrastructure, such as the one installed throughout Columbia University 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 in-

1Throughout this paper we will use GPS to designate any or both of the U.S. NAVSTAR GPS and the Russian GLONASS infrastructures.
face 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.

B. System Architecture

Our system architecture (Figure 3) addresses a number of important issues, such as flexibility and extensibility, distribution of computation, communication link independence, remote monitoring and control, and data storage and utilization. It is based on the manufacturer’s development platform [29]. 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 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. 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 taken.

Our hardware setup is accessed and controlled by seven servers (Figure 3, upper row of dotted triangles) that perform some or all of the tasks above. 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 sections. 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 sections.
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 and feeds them 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 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 coming from the low-level components and provides a compact view of the current state of the system.

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 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 or for a client (such as the user interface) to connect and monitor the status of each component. The standardized way of communication between components makes the architecture very flexible in that various functionalities can be achieved by simply adding or replacing an existing component with a new one. For example, when we want to test a particular behavior of the control system indoors (where GPS data, of course, is not available), all we need to do is run a program GPSSimulator instead of the GPSServer.

C. User Interface

The user interface (Figure 4) 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.

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. Multiple robots are also supported by the architecture.

III. Localization in Open Space

The first of our localization methods is designed for real-time usage in open-space outdoor environments. It uses the robot built-in odometry and the added digital compass/tilt sensor and GPS receiver.

A. Odometry

The ATRV-2 robot has two independently controlled wheels on each side in a differential-drive skid-steering configuration
(Figure 5). 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. Counts from encoders in the motors are continuously sampled by the firmware and used together with the kinematic parameters (wheel baseline \( b \) and the wheel radius \( r \)) to compute the robot angular displacement \( \Delta \theta_k \) and travel distance \( \Delta s_k \) during each sampling interval \( [t_{k-1}, t_k] \). Then the robot pose is computed according to:

\[
\theta_k = \theta_0 + \sum_{i=0}^{k} \Delta \theta_k
\]

\[
P_k = P_0 + \sum_{i=0}^{k} \Delta s_i \text{Rot}^{\circ}\theta_i \begin{bmatrix} 1 \\ 0 \end{bmatrix}
\]

where, \( P_k = [x_k, y_k]^T \) and \( \theta_k, v(t) \) are the robot position and orientation at time \( t_k \) and \( \text{Rot}^{\circ}\theta_i \) is the 2x2 rotation matrix by an angle \( \varphi \).

The above equations reveal the well-known problem inherent to odometry — minute errors accumulate over time beyond any reasonable bound. Typically, a robot can accurately traverse a few meters but for longer distances relying on odometry becomes impractical. The error can not be bounded without the use of an exteroceptive sensor such as our GPS unit discussed in section III-D.

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 [4]. 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. Both systematic and non-systematic errors can be addressed.

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. We have used their method called the UMBmark to do so [4]. Generally, the point of the performing such a calibration is to make sure the systematic errors will be reduced to negligible compared to the non-systematic ones.

2) Non-systematic Errors: Examples for causes of non-systematic errors include slippery spots, uneven terrain, surface irregularities and over-acceleration. Non-systematic errors can not be compensated for since they are by definition random and unpredictable. However, we can try to model their “average” behavior so that we keep track of the uncertainty in the robot pose estimates. We have modeled the robot deviation from its expected motion over a sampling period with a Gaussian probability distribution with standard deviation proportional to the distance traveled. While technically not exact, this is a good enough approximation over a short distance until the robot obtains external observations of its location. This model also has the benefit that it is simple and easy to fuse with the estimates of other sensors in the Kalman filter framework discussed below.

\[
\dot{x} = \begin{bmatrix} f_p^p(\phi, v) \\ f_p^\omega(\phi, \omega) \end{bmatrix} = \begin{bmatrix} v R(\phi) e_x + u_p \\ \omega M(\phi) e_x + u_\phi \end{bmatrix}
\]

Fig. 6. A diagram of the extended Kalman filter configuration.
with respect to the robot by the orientation of the GPS antenna. Denote the coordinates of the antenna augmenting the measurement vector with the expected position of a new fix becomes available, it is incorporated in the EKF by updating and the variance of the pose computation. Every time a limit. Our GPS unit provides both absolute position information and the position uncertainty from accumulating beyond acceptable limits. The GPS receiver, for example, is added to keep the heading information is unreliable and is ignored.

D. Global Positioning Systems

The EKF framework allows us to easily integrate additional sensors. The GPS receiver, for example, is added to keep the position uncertainty from accumulating beyond acceptable limits. Our GPS unit provides both absolute position information and the variance of the pose computation. Every time a new fix 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:

\[
\mathbf{h}(\mathbf{x}) = \mathbf{p} + R(\phi) \mathbf{p}_{g}.
\]

This is linearized to obtain the GPS measurement matrix. The variance is also incorporated into the filter in a straight-forward manner. The details can be found in [31].

Since the GPS is the only sensor in this method that makes absolute 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. This is done by monitoring the variance-covariance matrix representing the uncertainty in the Kalman filter. Each of the eigenvalues of this matrix 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 if the variance associated with the robot position is greater than a threshold. If so, we consider this as an indication that additional data is needed and attempt to use the visual localization method described next. Only the uncertainty in position is considered because if the orientation is wrong it will quickly cause the position error to increase also.

IV. VISUAL LOCALIZATION

To expand the working range of our localization system, it is sufficient to provide occasional “on-demand” updates only when the open-space 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 image processing operations which can be used to increase the robustness of the overall system.

This is the underlying idea in the use of our visual localization system. As the robot moves, it uses the open-space localization method described in the previous section to keep track of its pose along with the uncertainty. 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 compute a more accurate estimate using the vision-based pose estimation method described in this section. Since this happens relatively infrequently and because of the large-scale environments, the time the robot spends in place doing image processing is small compared to the overall travel time for a given mission.

A. Environmental Model

The visual pose estimation is based on matching an image of a building taken by the camera with a model. The environmental model we use a database of smaller-scale facade models. Each facade model depicts the features of a near-planar region around a building facade (Figure 7). The features modeled are dominant straight lines — typical and abundant in a human-made environments. All lines are finite segments represented by the 3-D coordinates of their endpoints in a local coordinate system, which is registered with the “world” coordinate system for the entire site.

In order to be useful, each facade model needs to capture enough features to provide sufficient information for the robot to find its pose. The number of features varies across buildings but beyond a certain limit, adding more detail quickly reaches the point where it does not help the pose estimation. There is no need to model every facade or every building either — what is needed is that enough building facades are modeled to allow continuous localization throughout the area of interest. Hence the model we use is simple and compact. For this paper, we have created the models by hand, however, our approach to how to create them automatically is discussed in section VI.

B. Choosing a Model to Use

When visual pose estimation is attempted, a rough estimate of the robot pose is 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 8).
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 visible features on the image; if it is too far, the accuracy of the result may be low because of the fixed camera resolution.

The second step is to eliminate facade models from the first step based on the viewing angle (ranging from $90^\circ$ for an anterior view to $-90^\circ$ for a posterior view). Only models that are viewable under a large enough angle are considered. This eliminates both the facades that are not visible (negative angles) and the ones that are visible at too low an angle to produce a stable match with the image.

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, we choose to use 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 pan-tilt head holding the camera is turned toward the chosen facade and an image is taken. The pan and tilt angles are computed from the known rough pose of the robot so that the camera faces the center of mass of the model. In practice, the final orientation of the camera is different from the ideal one because of the uncertainty in the current pose. However, for the small distances involved and the typical accuracy of the pose estimates, the resulting orientation error of the camera is usually within the tolerance of the processing steps that follow. Further, since the camera is aimed at the center of the model, any small deviation will have minimal effect.

C. 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 camera is tracked by the pan-tilt unit rigidly affixed to the robot, if the camera pose is known, then the pose of the robot can be easily derived. Thus, the focus from now on is on 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. We have adopted a probabilistic approach following the well-known RANSAC paradigm first introduced by Fischler and Bolles [32]. The method consists of the following five 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.

1) Preparation: The purpose of the preparation step is to obtain the line segments and do some pre-processing necessary for the steps that follow. 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 an incremental line fitting technique is used to connect them in straight line segments.

To reduce the computational burden in the following steps, collinear lines are merged and ones shorter than a given threshold are discarded. Details about this process are presented in the appendix.

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 this 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, we 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 [33]. 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 9). 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 $p_{j,1}$ and $p_{j,2}$. If $R$ and $T$ are the rotation and translation that align the world coordinate system with the one of the camera, then

$$d_{i,j} = (N_i \cdot (R \cdot p_{j,1} + T))^2 + (N_i \cdot (R \cdot p_{j,2} + T))^2$$  \hspace{1cm} (8)$$

is the sum of squared distances of the endpoints of $s_j$ to the plane formed by $l_i$ (Figure 9). The error function that is minimized is the sum of $d_{i,j}$ for the three matching pairs:

$$E(R, T) = \sum_{i,j \in \text{matches}} d_{i,j},$$  \hspace{1cm} (9)$$

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 [34].

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 (8). 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 consensus set is used together with equation (9) to compute a better pose estimate. This is done iteratively a number of times (between 1 and 4) 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.

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 $l_i$. Let the perspective projection of $s_j$ onto the image be $s'_j$ and the orthogonal projection of $l_i$ onto $s'_j$ be $l'_i$. We set contribution $c(s_j)$ of the match between $s_j$ and $l_i$ to the length of the overlap between $s'_j$ and $l'_i$. Thus, total portion of the model covered by matching line edges in the image is:

$$C(R, T) = \sum_{s_j \in M \text{ odd}} c(s_j)$$  \hspace{1cm} (10)$$

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. We have chosen to use the latter method and compute the pose evaluation metric as

$$q(R, T) = C(R, T) / V(R, T)$$  \hspace{1cm} (11)$$

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, we use three criteria by which eliminate a given pose candidate 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.

2) If the visible portion of the model on the image is less than a threshold, the pose is also rejected as there is not 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.

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. 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 its pose to reduce the uncertainty.

The decision on how many iterations to perform depends on the number of matching lines which is impossible to know in advance. We terminate the loop after a constant number of iterations. Our justification for the number of iterations is given in the appendix.
V. EXPERIMENTS

To demonstrate the functionality of the mobile robot, the software architecture and the software components we wrote as well as to study the performance of the localization algorithms, we performed a series of tests with the 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, compass and GPS; another that focused only on the vision component; and a final test that used all sensors.

A. Localization in Open Space

The purpose of these tests was to investigate the accuracy of the open space localization method described in section III. 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 reached the current one. The maximum translational and rotational velocities were 0.5 m/s and 0.4 rad/s respectively.

To test the accuracy of the system, two 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 10 shows the planned and actual trajectories, overlaid on the map of this area of the campus. The average deviation of the robot from the planned trajectory over all sampled points 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 10. The trajectory was 132 m long and asked the robot to return to the same place where it started (Figure 11). The average error for this run was 0.251 m.

The next experiment also involved the trajectory in Figure 11, 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 results 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 within the acceptable range for our urban site-modeling task.

B. Localization with Vision

To examine the accuracy of the visual localization method, we 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, we wanted to measure the quality of the location estimation alone and minimize the interference from inaccuracies in the model. Thus, we 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 modeled were
windows, ledges and decorations — all commonly found and abundant in urban structures and easy to find using 2-D image operators (Fig. 12).

In the first test, the robot was driven along a long trajectory around a large building. 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 13.

Figure 14 shows 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 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 were 0.081m (min), 0.442 (max) and 0.268 (average). 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 can be seen from Figure 14, we wanted to obtain a quantitative confirmation. We did this by running Tsai’s method for external camera parameters estimation [35] and comparing its orientation estimates with the ones from our localization algorithm. The resulting errors were within a fraction of a degree: 0.131° (min), 0.977° (max) and 0.570° (average).

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. We took a pair of images from the same spot at locations 4 and 5 by simply panning and tilting the camera. Both pairs of images were processed with their corresponding models (Figures 15–16) and were intentionally given initial pose estimates with large errors. The resulting positions for each pair were compared with each other and revealed only small discrepancies: 0.064 m and 0.200 m. Further details can be found in [31].

C. 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 (Figure 17) and the robot was directed to follow
Fig. 14. Visual localization tests. Each image shows the matching model overlaid as it would be seen from the estimated camera pose. The left image in each pair shows the rough estimate of the pose that was used to initiate the visual localization. The right image shows the resulting pose of the algorithm.
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 17) 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 Section IV. While at rest, we scanned its camera with an electronic theodolite to obtain ground truth.

Table I compares the estimates of the robot position at each location. The top line of each table row shows the estimate of the open-space localization method prior to triggering the visual procedure and its error. The bottom line of the same row, shows the estimate and the error of the visual localization. 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 Figure 18.

<table>
<thead>
<tr>
<th>No</th>
<th>Type</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Open-space Vision</td>
<td>17.547</td>
<td>-58.079</td>
<td>-0.683</td>
<td>1.297</td>
</tr>
<tr>
<td>2</td>
<td>Open-space Vision</td>
<td>108.521</td>
<td>-61.634</td>
<td>0.279</td>
<td>1.031</td>
</tr>
<tr>
<td>3</td>
<td>Open-space Vision</td>
<td>90.660</td>
<td>-30.729</td>
<td>1.418</td>
<td>0.937</td>
</tr>
<tr>
<td>4</td>
<td>Open-space Vision</td>
<td>95.095</td>
<td>2.577</td>
<td>0.867</td>
<td>0.512</td>
</tr>
</tbody>
</table>

VI. CONCLUSION AND FUTURE WORK

This paper presented a systematic and practical approach to mobile robot localization in urban environments. It reflects work on both system and algorithmic components. The work was done as part of our 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.

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. We have also designed a distributed modular software architecture for mobile robot localization and navigation.

On the algorithmic level, we have presented two methods for localization of mobile robots. The open-space localization method uses odometry, a digital compass and GPS. The sensor integration is done in an extended Kalman filter framework. The method can be performed in real-time. The visual localization method is heavier computationally but is only used upon demand. The pose estimation is done by matching an image of a nearby building with a simple and compact model. A database of the models is stored on the on-board computer. No environmental modifications are required. We have demonstrated the functionality of the robot and the localization methods with numerous experiments.

One last thing that we need to discuss is the way we obtain...
the environmental model used for visual localization method. It is tightly coupled to the intended use of the method. Recall that the work presented here is part of a project whose goal is the creation of a detailed geometric and photometric 3-D model of an urban site. We refer to this detailed model as the **detailed model**, as opposed to the **localization model** used for localization.

The detailed 3-D models obtained from the range scans and images of the buildings are too large and complex since they capture a lot of detail (Figure 19, center). The modeling process is incremental. At each stage there is a partial model of the site available. During the scan/image registration and 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 coordinate systems of the range scanner and the camera. This simplified model (Figure 19, right) consists of 3-D line segments obtained by segmenting the range scans into planar regions and intersecting planes to obtain lines (for details, see [36]). The result of this operation is a set of line segments — the kind that we need for visual localization.

Thus, to create a localization model, only some post-processing is needed of the available 3-D line features. The set of lines need to be broken into near-planar regions and a representative coordinate system needs to be established for each such region. This is the focus of our current efforts to complete the integration between the modeling and the localization aspects of our project.

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**

**NUMBER OF ITERATIONS AND SPEEDUPS IN THE VISUAL METHOD**

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

$$ p = \frac{n (n - 1) (n - 2)}{n_3 (n_3 - 1) (n_3 - 2)} \frac{1}{n_2} \frac{1}{n_2 - 1}. \quad (12) $$

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

$$ E(k) = \sum_{i=1}^{\infty} i \cdot p \cdot (1 - p)^{i-1} = \frac{1}{p}. \quad (13) $$

We see that $E(k)$ depends on the number of matching line segments which is impossible to know in advance. Our approach is to use a fixed number of iterations which is determined on the basis of the number of lines in the model and the average number of edge lines used in the pose computation step. This number can be controlled to a large degree by choosing an appropriate threshold in the reduction steps described below.

Typically, $E(k)$ is a computationally prohibitive number and we take a number of steps to make it tractable. The first step is to merge all collinear lines in both the 3-D line and the 2-D line sets. This ensures a one-to-one match between the two sets and eliminates a great number of practically equivalent combinations.

Next, we notice that short lines are not as informative as long ones, as a slight perturbation of the endpoints of a short line (for example, due to misdetected edgels) could lead to large change in its orientation. Therefore, we discard line segments that are shorter than a threshold thus further reduce the value of $E(k)$.

Additional decrease 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 digital compass module and the building facade is assumed to be a 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.

For typical values, such as $n_2 = 30$, $n_3 = 30$ and $n = 20$, with an approximately equal number of horizontal lines and vertical lines, these reduction steps can bring the expected number of iterations down to less than 11,000. This already is a practical number. For comparison, all visual localization tests in this paper 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.

**ACKNOWLEDGMENT**

This work was supported in part by ONR/DARPA MURI award ONR N00014-95-1-0601 and NSF grants EIA-97-29844 and ANI-0099184.

**REFERENCES**


Fig. 19. Model acquisition and simplification: an image of a building (left); the 3D model created from the image and a range scan (center); a reduced version of the same model consisting only of line features (right).


