Design, Architecture and Control of a Mobile Site-Modeling Robot

A. Gueorguiev, P. K. Allen, E. Gold and P. Blaer, CS Dept, Columbia University*

Abstract

A distributed, modular, heterogeneous architecture is presented that illustrates an approach to solving and integrating common tasks in mobile robotics, such as path planning, localization, sensor fusion, environmental modeling, and motion control. Experimental results are shown for an autonomous navigation task to confirm the applicability of our approach.

1 Introduction

This paper describes the design, architecture, and control of an autonomous mobile site-modeling robot. Site models are used in a number of applications, such as city planning, urban design, fire and police planning, military applications, virtual reality, and others. This modeling is done primarily by hand, and owing to the complexity of these environments, is extremely painstaking. The models built are often incomplete and updating them can be a serious problem.

To alleviate this, we have built a mobile robot system that has been instrumented with both range and imaging sensors and can be used to build photo-realistic, geometrically accurate 3-D models of outdoor sites. This modeling process is described in detail in [11, 15]. The focus of this paper is the design of the mobile system itself, emphasizing the task of autonomously navigating to a location to acquire the necessary sensor data for site modeling in an intelligent way. The design and architecture of this robot brings up a number of important issues in mobile robotics, including localization methods, sensor fusion, path planning and navigation, remote vs. centralized control, and wireless communications, all of which are discussed in this paper. We also present results from autonomous navigation experiments with the robot.

For a site modeling task, the robot is provided with a 2-D map of its environment. High-level planning software is used to direct the robot to a number of different sensing locations where it can acquire imagery that is fused into a photo-realistic (i.e texture mapped) 3-D model of the site [11]. The system must plan a path to each sensing location and then control the robot to reach that location. Positional accuracy is a paramount concern, since reconstructing the 3-D models requires precise registration among image and range scans from multiple acquisition sites.

2 Related Work

The problems of mobile robot localization, control and navigation, as well as autonomous map building, have been studied separately and in combination for more than a decade. Early classical approaches to navigating a robot in an indoor environment are [4, 5, 8]. More recent works include [7, 17]. The larger distances traversed by a mobile robot in an outdoor environment emphasized the need for additional sensors, due to the inherent limitations of odometric devices. Various types of inertial navigation systems were introduced [3] and methods were developed to optimize their performance [12]. Methods were also developed to fuse data from various sensors to provide higher accuracy and better reliability. The most common approach is combining an inertial navigation system with a GPS [1, 16]. Many of them utilize a Kalman Filtering technique to achieve statistically near-optimal performance [1, 7, 16]. Various other methods of determining the robot's location were proposed in [10, 14]. Two excellent approaches to map acquisition are shown in [9] and [17]. An example of a similar to our distributed approach can be found in [2]. System components integration is addressed in [13]. Our approach differs from the above in that it allows for better distribution of computational resources and easier integration of heterogeneous components.

3 Hardware Configuration

For our experiments, we use an *ATRV-2* mobile robot manufactured by RWI, Inc (Figure 1). To maintain continuous connectivity to the robot's on-

*This work was supported in part by an ONR/DARPA MURI award ONR N00014-95-1-0601 and NSF grants CDA-96-25374 and EIA-97-29844.

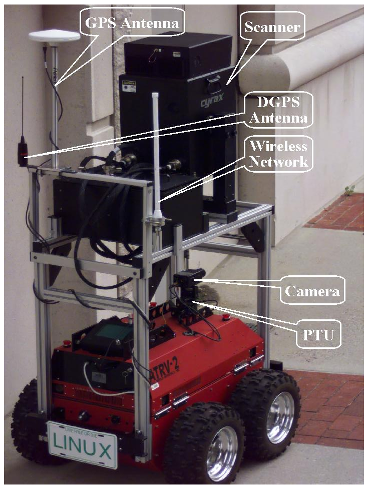


Figure 1: Our mobile robot

board computer from remote hosts, we utilize a wireless ethernet connection. Access points for wireless access are positioned to give us maximum coverage of the portions of campus on which we do our testing. Two GPS+GLONASS receivers running in RTK/CPD (Real-time Kinematic/Carrier Phase Differential) mode provide us with positioning information. One of them serves as a base station on the roof of a tall nearby building, sending differential corrections to the other, on the robot. With enough satellites visible (usually 7 or 8), this setup gives us 1Hz position updates accurate to a few centimeters. A color CCD camera is affixed to a pan-tilt unit (PTU) mounted in the front of the robot Images can be transmitted to the host computers using the software described in the next section. We have also mounted a Cyrax 2400 laser range scanner on a custom-built platform attached to the robot. The scanner uses an eye-safe class II laser and provides variable resolution scans up to 100 meters. These two sensors are the primary acquisition devices for the site modeling task.

4 Architecture

A major problem when designing a mobile robot architecture is the distribution of computation. As with battery power, payload, sensor range, and so on, computational resources are limited and usually insufficient. This is especially true when it comes to processing images or large-scale environmental models.

We believe that the correct approach is to distribute computation across multiple computers. However, placing a number of computers on the robot is not always desirable — this would be at the price of reduced payload and battery life, and is not scalable. A better solution is to use a distributed wireless system that has the advantages of providing theoretically unlimited computational and storage resources. Our efforts are directed towards investigating this approach.

As a first step, we have designed the distributed object-oriented architecture shown in Figure 2. It is based on Mobility — a robot integration software framework developed by RWI, Inc. In addition to helping us handle the low-level interface with the robot, Mobility provides us with components that are abstractions of various hardware pieces, such as sensors and actuators. It is CORBA compliant which translates into platform, operating system, and programming language independence.

The main building blocks of our system are Mobility components. Each component is a stand-alone piece of code that performs a specific task. For example, *Odo* provides odometric data from the robot and *Drive* supplies low-level control commands to the robot. Important components usually maintain a data structure (also a component) that represents their state. Other components may query this data structure about the current or a previous state (client pull approach) or register with it to receive an automatic notification when changes occur (a server push approach).

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 robot interfacing, navigation control and so on. The standardized way of communication between components makes servers easily reconfigurable and replaceable and provides extreme flexibility. 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*. Similarly, we can test the navigation system on a *Pioneer I* robot by simply running a *PioneerServer* instead of the *ATRVServer*.

Because of the underlying framework and the wireless connection, a server is not required to run on the computer physically residing on the robot. This provides the basis for the distributed nature of our system. It also raises many design questions, such as how and to what extent to utilize the resources of remote hosts. Apart from the general problems of distributed systems related to network bandwidth, connectivity, security, etc., in the context of mobile robots, the second question has a specific flavor: What part of the computation needs to be run on-board in order to guarantee robot's operability in areas uncovered by

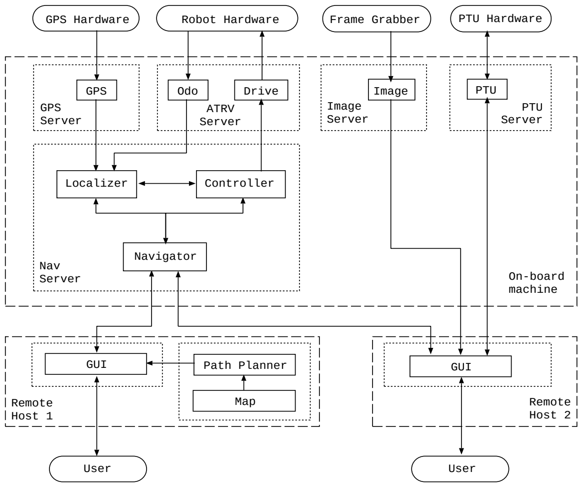


Figure 2: The architecture of our system. Solid rectangles represent modules, dotted rectangles are processes, and dashed rectangles group processes running on the same machine. The arrows show the data flow between modules.

the network? This leads to a natural classification of the software components. Those that are hardware related (e.g. sensor/actuator interfaces) or mission-critical (e.g. the low-level control system) have to be run on-board. We call them *core* components. They are, by definition, lightweight in that they do not consume a lot of resources. Conversely, *remote* components are either not critical or are extremely heavy-weight and, therefore, have to be run on remote hosts.

In our case, 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. Examples of remote components are the user interface and the *Path Planner*, which turned out to be computationally and data intensive. A description of each component is given in the following subsection.

Note that it is perfectly acceptable to have a core component and one or more remote components that are designed to perform the same task. In this case, the remote components will implement the full computationally intensive functionality of their task, while the core component will be a stripped-down emergency substitute. This is the approach we have adopted to effect a consistent control scheme over a wireless connection that can sometimes break down.

A. Description of the 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, measurement units, etc. 2) provide multiple, including user-defined, views of the data coming from the device (e.g. polar or cartesian coordinates). 3) control the volume of the data flow, for example the rate at which images will be grabbed.

Our hardware is controlled by four servers, that perform some or all of the tasks above (Figure 2). The *ATRVServer* is the interface to the robot's 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 *GPSServer* parses the data from the GPS receiver and makes it available to other modules. 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 [6]. The other two are local: east-north-up and x-y-z with respect to a user-defined coordinate system. Additional information includes the number of satellites used, current mode (RTK float or fixed), HDOP, etc.

The *PTUServer* is a simple server that we use to point the camera in a desired direction. Its two main commands are *PAN* θ and *TILT* ϕ, which are self-contained. It also allows the current pan and tilt positions to be queried.

The last hardware server, the *ImageServer* supplies a stream of images taken from the sensors. We use it together with the *PTUServer* to obtain visual feedback from the robot on remote machines.

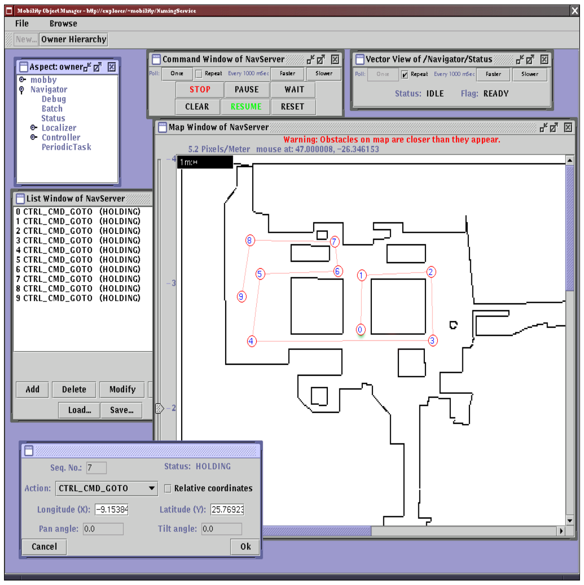
The *NavServer* (beneath the hardware servers in Figure 2) builds on top of the hardware servers and provides a higher-level interface to the robot. A set of more intuitive commands, such as "go there", "establish a local coordinate system here", and "execute this trajectory", are composed out of the low-level hardware control input. The server also provides feedback on the progress of the current tasks. It consists of three modules: 1) The *Localizer* is a part of the robot's control system that performs data fusion. It obtains new readings from the odometry and the GPS, registers them with respect to the same coordinate system, and produces an overall estimate of the robot's position and velocity. 2) The *Controller* is a control module that brings the robot to a desired pose. It executes commands of the type *GOTO* x, y and *TURNTO* ϕ. 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 *ATRVServer*. 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 modules.

A mission consists of commands that are carried out sequentially. The user specifies commands using the User Interface (details below) and sends them to the *Navigator* for execution. Alternatively, commands can be sent by any remote component. The *Navigator* itself does not execute most of the commands — it simply stores them and resends them to the appropriate components, 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 modules. This is useful in two ways: 1) it allows users who have just connected to the robot to see what it is trying to achieve and how much it has accomplished; 2) 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 network.

B. User Interface

The goal of our user interface is to provide a comprehensive real-time view of the robot's location and activities within its environment. It provides both a list view of the current set of commands and an integrated map view displaying navigation targets and paths overlaid on the map. The click-and-drag map view also facilitates manual generation of a desired trajectory or importing one from files or other components (such as the *Path Planner*). The list view is useful when we want to specify exact coordinates of a target location or a specific version of a command (e.g. move backwards). Critical navigation commands, such as *STOP*, *PAUSE*, and *RESUME* are available as a convenient stand-alone toolbar. Our layered architecture allows us to interact with the robot at several levels of detail simultaneously. While watching the progress on the map view, we can still obtain raw odometric or GPS data, or monitor the status of each component. Images from the robot's camera can



Status: IDLE Flag: READY

Warning: Obstacles on map are closer than they appear.

5.2 Pixels/Meter mouse at: 47.000008, -26.346153

```
0 CTRL_CMD_GOTO  (HOLDING)
1 CTRL_CMD_GOTO  (HOLDING)
2 CTRL_CMD_GOTO  (HOLDING)
3 CTRL_CMD_GOTO  (HOLDING)
4 CTRL_CMD_GOTO  (HOLDING)
5 CTRL_CMD_GOTO  (HOLDING)
6 CTRL_CMD_GOTO  (HOLDING)
7 CTRL_CMD_GOTO  (HOLDING)
8 CTRL_CMD_GOTO  (HOLDING)
9 CTRL_CMD_GOTO  (HOLDING)
```

Seq. No.: 7 Status: HOLDING

Action: CTRL_CMD_GOTO ☐ Relative coordinates

Longitude (X): -9.15384 Latitude (Y): 25.76923

Pan angle: 0.0 Tilt angle: 0.0

Figure 3: The Graphical User Interface

also be displayed. A PTU control interface allows us to point it in the desired direction and obtain rich visual feedback. We have been able to successfully teleoperate the robot in the corridors of our building this way. Finally, the *Path Planner* is a component that automatically generates a safe trajectory to a desired point. It uses a two-dimensional polygonal map of the area and produces a piece-wise linear trajectory that can be viewed and, possibly, corrected or enhanced using the user interface, and then sent to the robot.

5 Control

Planning a path to a desired viewpoint is only one step in the complex task of data acquisition. An equally crucial step is to accurately follow the path to prevent collision with obstacles in the environment. We must also ensure that the actual robot position and orientation at each sensor acquisition site can be accurately computed so that the scans acquired at these sites can be later correctly registered and integrated.

We make use of two sources of localization information: the GPS and the robot odometry. Both of these have problems, which require the intelligent fusion of the their position estimates. The GPS system is subject to radio link downtime, insufficient number of satellites in view, low update rates (1Hz), high-frequency jumps, and athmospheric conditions [6]. The odometry is unsuitable for long distances since it is affected by slip and calibration and modeling inaccuracies, due to which its error accumulates, growing potentially unbounded with time. Our solution is to use these two sources of information synergistically to effect real-time trajectory following, minimizing path deviations as they occur.

A. Fusing Position Estimates

The *Localizer* performs fusion of the data obtained by the sensors. Since our campus is mainly flat, we use a 2-D coordinate system to facilitate computation and, thus, our robot's state is modeled by a 5-dimensional vector $x = [\xi, \eta, \theta, \nu, \omega]^T$ where $[\xi, \eta]^T$ is the robot's position, θ is its orientation, and ν and ω are the current translational and rotational velocities.

The fusion process consists of taking a linear combination of the measurements of the two sensors:

$$\bar{x} = k\bar{x^g} + (1-k)\bar{x^o} \tag{1}$$

where x^g and x^o are the GPS and odometric estimates of the current state, and k is a coefficient that determines the relative weight of the two estimates. The above equation is applied only to the first three state variables, since we do not use the GPS to obtain velocity information. The odometric measurement of velocities is accurate enough and we accept it directly into the resulting vector. Direct GPS observations of the current vehicle orientation are not used either. We have found that deriving an estimate of the orientation from the GPS position fixes works better. As our trajectory consists of straight line segments and the sampling rate of the GPS is relatively low (1Hz), when the robot is moving towards a target, our control law produces rotational velocities that are much smaller than the accuracy of the GPS fixes. Thus, we can neglect them and assume that the robot has moved along a straight line. Hence, at a given time t_n, we fit a straight line l^g to a window of m past GPS readings and obtain the estimate $\theta_i^g = \theta^g = const$. The GPS estimates ξ_i^g and η_i^g of the robot position at time t_i are given by the GPS readings $[u_i, v_i]^T$ at that time plus a correction that accounts for the displacement of the GPS antenna from the center of odometry:

$$\begin{aligned} \xi_i^g &= u_i - a_x cos(\theta^g) + a_y sin(\theta^g) \\ \eta_i^g &= v_i - a_x sin(\theta^g) - a_y cos(\theta^g) \end{aligned}$$

where a_x and a_y are the GPS antenna coordinates with respect to the robot's local coordinate system.

The coefficient k in (1) represents our confidence in the accuracy of the GPS data during the time frame considered. It is composed of two parts: $k = k_{shape}k_{dir}$. Here, k_{shape} represents our confidence in the data as derived from the shape of the trajectory traveled and is computed as $k_{shape} = max\{0, 1 - k_l \sum_{i=n-m+1}^{n} dist([u_i, v_i]^T, l^g)^2\}$, where k_l is an experimentally derived coefficient. The coefficient k_{dir} is a factor that depends on the discrepancy between the orientation estimates from the GPS data and the model. It prevents the introduction of occasional drifts in the GPS data into the status. Thus, we evaluate k_{dir} as $max\{0, 1 - k_\theta(\theta^g - \theta)^2\}$, where θ is the angle to the target and k_θ is experimentally derived.

B. Motion Control

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. Given the current state and the target position, the target is first expressed in polar coordinates $[\Delta\phi, \Delta\rho]$ with respect to the robot's current local coordinate system. The resulting translational and rotational displacements are then multiplied by a pair of experimentally determined gains to obtain the new velocities $\nu_{new} = -k_\nu \Delta\rho$ and $\omega_{new} = -k_\omega \Delta\phi$. Before applying these velocities certain limits are imposed. First, we cancel high accelerations and then restrict the velocities within an acceptable interval. Finally, the new values are sent to the *Drive* component for execution.

6 Experiments

To test our robot's ability to correctly execute its tasks, a series of tests were performed on our campus. 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 polylines, 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. In all cases, the robot performed as expected with no visible deviation.

To further confirm these results, a more comprehensive experiment was 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. A complex desired trajectory of 14 targets and total length of 210 m was generated and sent to the robot. After its execution, sample points from the actual trajectory were marked at intervals of approximately 1 m and measurements of each sample point were obtained. Figure 4 shows the planned and actual trajectories, overlaid on the map of this area of

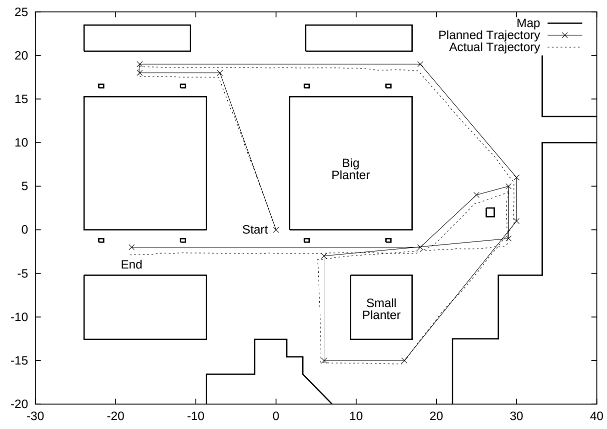


Figure 4: A complex test run

our campus. The average error in this run was 0.46 m.

It should be noted that the performance of our system strongly depends on the accuracy of the GPS data that we get. During the experiment above, the number of satellites used were 6 or 7 most of the time, occasionally dropping to 5 or increasing to 8. The GPS receiver was working in RTK float mode in which its accuracy is seriously deteriorated compared to when it works in RTK fixed mode. The latter mode provides accuracy to within a few centimeters, however, it is only available when 7 or more satellites provide good signal-to-noise ratio over a long period of time.

Another test that we performed was a polygonal trajectory in the shape of the digit “8” around the two planters in the center of Figure 4. The trajectory was 132 m long and asked the robot to return to the same place where it started. During 3 such runs, RTK fixed mode was intermittently available and the robot always returned within a foot of the starting point. In contrast, when using odometry only, the robot never succeeded to go around the big planter alone.

7 Summary and Future Work

This paper has described a mobile robot system that is being built to autonomously navigate in a complex environment to create 3-D site models. The system has a hardware/software architecture that allows integration of sensing, control and user interaction. Tests have shown that the robot can effectively navigate in real-time in our campus environment. We are extending the control and localization to include real-time visual feedback. Using known visual landmarks on campus, we can provide a third means of positional rectification (beyond GPS and odometry) which can easily be included in our sensor fusion module. We are also using the site models we create to perform an update to the site map, including a full 3-D map, which can be used for later navigation tasks.

References

[1] T. Aono, K. Fujii, S. Hatsumoto, and T. Kamiya. Positioning of vehicle on undulating ground using GPS and dead reckoning. In *ICRA'98, Leuven, Belgium*, pages 3443–3448, 1998.

[2] P. Backes, G. Rabideau, K. Tso, and S. Chien. Automated planning and scheduling for planetary rover distributed operations. In *ICRA'99, Detroit, Michigan*, pages 984–991.

[3] J. Borenstein and L. Feng. *Navigating mobile robots: systems and techniques*. A. K. Peters, 1996.

[4] J. Crowley. Navigation for an intelligent mobile robot. In *IEEE Conf. on Applications of Artificial Intelligence*, pages 79–84, 1984.

[5] A. Elfes. Sonar-based real-world mapping and navigation. *IEEE Trans. on Robotics and Automation*, RA-3(3):249–265, June 1987.

[6] B. Hofman-Wellenhof, H. Lichtenegger, and J. Collins. *Global Positioning System Theory and Practice*. Springer-Verlag, Wien, 4th edition, 1997.

[7] L. Jetto, S. Longhi, and G. Venturini. Development and experimental validation of an adaptive extended kalman filter for the localization of mobile robots. *IEEE Trans. on Robotics and Automation*, 15(2):219–228, April 1999.

[8] J. J. Leonard and H. F. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Trans. on Robotics and Automation*, 7(3):376–382, 1991.

[9] F. Lu and E. Milios. Globally consistent range scan alignment for environmental mapping. *Autonomous Robots*, 4:333–349, 1997.

[10] A. Mallet and S. Lacroix. Toward real-time 2D localization in outdoor environments. In *ICRA'98, Leuven, Belgium*, pages 2827–2832, 1998.

[11] M. Reed and P. K. Allen. Constraint-based sensor planning for scene modeling. In *CIRA'99, Monterey, CA*, 1999.

[12] N. Roy and S. Thrun. Online self-calbration for mobile robots. In *ICRA'99, Detroit, Michigan*, pages 2292–2297.

[13] A. Schultz, W. Adams, B. Yamauchi, and M. Jones. Unifying exploration, localization, navigation and planning through a common representation. In *ICRA'99, Detroit, Michigan*, pages 2651–2658, 1999.

[14] R. Sim and G. Dudek. Mobile robot localization from learned landmarks. In *Proceedings of the IEEE/RSJ Conf. IROS'98 in Victoria, BC*, October 1998.

[15] I. Stamos and P. K. Allen. Integration of range and image sensing for photorealistic 3D modeling. *In ICRA'2000.*

[16] S. Sukkarieh, E. Nebot, and H. F. Durrant-Whyte. Achieving integrity in an ins/gps navigation loop for autonomous land vehicle applications. In *ICRA'98, Leuven, Belgium*, pages 3437–3442, 1998.

[17] S. Thrun, W. Burgard, and D. Fox. A probabilistic approach to concurrent mapping and localization for mobile robots. *Autonomous Robots*, 5:253–271, 1998.