NaviGates: A Benchmark for Indoor Navigation

Ryan M. Knotts[1] **Illah R. Nourbakhsh**[2] **Robert C. Morris**[3]

Abstract

Reliable indoor navigation has been a goal of mobile robotics research in recent years. Probabilistic belief update, occupancy grid methods, and multi-tiered hybrid architectures have all been implemented in the pursuit of autonomous navigation systems that can function in the face of danger and incomplete information. The conventional approach applies AI techniques to the localization problem in order to attain reasonable navigation reliability. In this paper, we describe a simple navigation system that is neither multi-tiered nor AI-based. The system has been implemented on a Nomad 150 mobile robot that has demonstrated statistically significant navigation reliability during working hours in several buildings in the San Francisco Bay Area. In addition to providing navigation benchmarks based on long-term empirical tests, NaviGates demonstrates implementations of robot skills that are essential to robot autonomy: robust dynamic obstacle avoidance, path replanning in case of obstruction, automatic avoidance of dangerous regions such as staircases, and automatic human-guided mapping for a new building, allowing the robot to be introduced to a new office building without the need for time consuming manual mapping. Simplicity itself is an important thesis of this research, as a robot with an extremely simple control algoritm is shown to perform admirably in a variety of environments.

1 Introduction

Professor Nils Nilsson's recent Robot Challenge [Nilsson 1996] states that the time is ripe for robotics to serve as the *integrator* of the many components of intelligent and autonomous systems, from reasoning and navigation to manipulation and vision. The *Robot Challenge* will be met only when a robot functions in an unmodified office building environment, on-the-job, for a full year.

This work addresses one aspect of the *Challenge*: navigation. A useful office robot must have navigation competence; it must be able to navigate an office building with extremely high reliability.

In this paper, we present a navigation system that meets these needs. We begin by describing the hardware to which the NaviGates Project is limited. Then we describe the NaviGates architecture, followed by a description of the implemented system that achieves navigation competence.

We establish NaviGates' reliability by offering experimental results spanning several different office buildings in the San Francisco Bay Area. The variety of venues verifies the general nature of the solution we provide as well as the reliability of the mapping component.

In addition, we hope that the statistically significant nature of our experimental results can serve to establish a benchmark for indoor robot navigation, serving as a starting point in the community for improvements in navigation competence.

After presenting empirical results, we close with a discussion of NaviGates' relationship to conventional office building navigation robots.

[1] Cisco Systems, Inc., San Jose, CA 95134
[2] Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213
[3] Scitor Corp., Menlo Park, CA 94025

2 Hardware

The mobile platform used in our work is a Nomad 150 robot from Nomadic Technologies. It is controlled by a Macintosh Powerbook 170 running Macintosh Common Lisp v2.0.

2.1 Sensors

The Nomad 150 has 16 sonars in a planar ring configuration around a motorized turret. The sonars have a usable range from 12 cm to approximately 250 cm from the robot's surface, with 2.5 cm resolution. Additional navigation information is available from internal encoders, which maintain an approximately accurate picture of x-y location, turret angle and base angle.

2.2 Effectors

The wheels on the base of the robot pivot in place to allow the line of travel to vary even as the orientation of the robot stays constant. Additionally, the turret which mounts the sonars rotates with respect to the main body. Together, these features allow the robot to maintain a pair of sonars oriented perpendicular to the hallway it is traversing, a useful behavior for the feature extractor (see below).

3 Architecture

The system is designed in a layered format. (Figure 1) The layer closest to the robot's effectors controls reactive behavior, such as obstacle avoidance, and the final coordination of specified velocities. The second layer controls sub-goal directed behavior or behaviors which must be completed before the intended goal can be achieved. The third layer takes the global plan as input and selects the next subgoal for the layer below. The fourth layer is a planner, which simply creates a plan based on the map, current position of the robot and goal position. The architecture also divides the data analysis for each layer into separate data analyzers.

All levels of the architecture can access map information, and the structure of the map reflects this to a certain degree, containing information relevant to each layer.

Map
Each level of the architecture uses the Map. The Planner uses the node links to form a path from the current node to the desired node. The Node Handler uses link directions to determine whether a corner must be turned, and if so, in what direction. The Node Navigator uses the dimension of a node and expected sonar values while traveling through the node. The Avoidance and Safety level uses dimensions of the node to determine safety zones.

Planner
The planner is a generic potential field planner through a node network which provides a list of nodes to travel in order to get to the desired node. It could be replaced easily, or easily directed by a higher, strategic planner which would determine tasks.

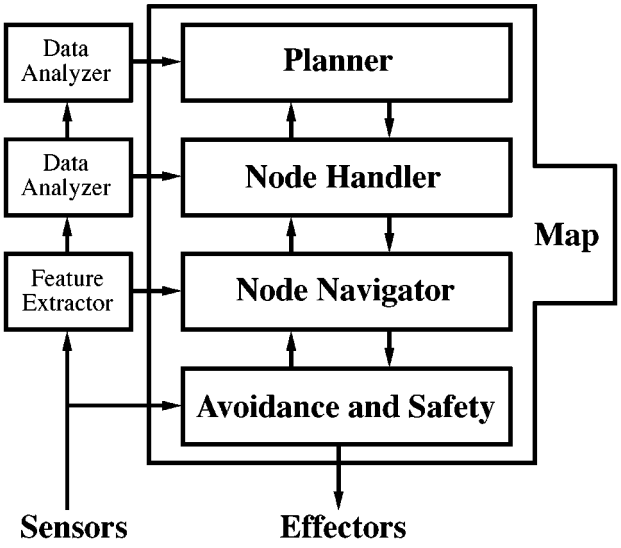

Figure 1: The Navigates Architecture

Node Handler
The node handler takes a list of nodes from the planner and hands them to the node navigator in order, indicating corners and changes of direction, and several other cases.

Node Navigator
This module takes a given node as input. The Navigator then uses all means possible to traverse the node, indicating to the Handler when the node has been successfully negotiated.

Avoidance and Safety
This level controls obstacle avoidance as the robot attempts to navigate a node and prevents the robot from drifting too far from the centerline of the node (safety zone). The obstacle avoidance is not necessarily a formal reactive system.

Feature Extractor
This box takes the raw data of the sensors and determines if a feature has been detected. If it has, the feature is passed on to the Navigator for potential matching to a mapped feature.

Data Analyzers
These might process either the output of the feature extractor or raw data to produce higher level data for the higher level layer of the architecture. The implemented system did not use these sections.

4 Implementation

The robot designed based on this architecture shows remarkable navigational reliability. This is due to the very simple method of navigation used by the Node Navigator level, and the degree to which this simple method was refined.

4.1 Map

The purpose of the map is to present a useful representation of the world the robot inhabits. Therefore the map takes advantage of our target environment to simplify the information it carries. The map consists of a linked array of nodes, which break down a given floor of a building into segments of hallway bounded by exit features. All nodes are represented as rectangular areas. In an outdoor environment, this would be a severe limitation, but for office navigation it is entirely acceptable.

The nodes are represented geometrically as a North-South and East-West dimension pair. This information is the primary data used for navigation. Additionally, the node contains four links (NSEW) to other nodes for planning purposes. The node also contains expected sonar values for travel in a given direction. This information is used to give the robot a bias towards travel in certain areas of the hallway. Given no necessary obstacle avoidance behaviors, the robot will tend to stay in the lateral position in the hallway which causes the sonar values to most closely match the expected sonar values in the map. Finally, the nodes each contain exit jump information for all four directions of travel. These features are used by the Node Navigator for purposes of localization.

4.2 Planner

The planner is a simple potential field planner that finds paths very rapidly in the map's 2D representation. Currently, the planner optimizes based on the number of nodes traversed; however, one could easily use the map's more detailed node features to optimize distance traveled instead.

4.3 Node Handler

The Node handler for this system is also very simple. Given the path information, it assigns the Node Navigator a node to traverse, and the Navigator signals the Handler when it has successfully traversed the node. In addition, the Handler is also the level at which problems of blocked nodes and non-navigable situations need to be identified and the relevant information sent back to the planner so an alternative route might be planned.

4.4 Feature Extractor

The Feature Extractor of the robot provides accurate longitudinal or "distance traveled" information at intervals. The robot detects "jumps", or significant changes in sonar values, selects jumps which probably represent landmarks or "features", and compares these features against features recorded on the map. By monitoring the 3 o'clock and 9 o'clock sonars (facing directly right and left, respectively) as the robot moves forward, the robot can determine when there is a significant value difference between the last value of a sonar and the current value of that sonar. This difference in the sonar value is referred to as a "jump". Jumps may be caused by branching hallways, office doors, structural supports, passing pedestrian, sonar error, etc. (Figure 2) For example, if the sonar reading of the left-facing sonar indicates x cm at time t and x' cm at time t+1, the robot examines this jump in the sonar value as a potential "feature". A feature is a repeatably detectable jump in the position of the hallway walls.

To minimize the probability of false feature detection, we implement a preprocessing filter that eliminates unreliable sonar values. There are three cases in which this filter suppresses feature detection. Of course, since the sonars are discretized into 2.5 cm units, a change of less than 7.5

cm is attributed to drift in lateral position and not examined as a feature. If the robot detects a jump which is too far away to be in the same hallway this jump is also ignored. This case might occur if the robot is traversing one side of a room, and detects a sonar jump on the other side of the room. And finally, jumps greater than a maximum jump threshold are ignored unless either the last value of the sonar or the current value of the sonar is greater than the width of the hallway.

The preprocessor filters can be summarized as follows. The value returned by the sonar at time *t* we will call *x*, and the value returned at time *t+1* will be *x'*. The first criteria is that *x* and *x'* differ by a value greater that the minimum jump threshold, say 5 cm, which is input when the robot begins to run. The second criteria is that both *x* and *x'* cannot be larger than a maximum hallway threshold, say 125 cm, which is a value derived from the maximum width of the hallway. Finally, if *x* and *x'* differ by more than the maximum jump threshold, say 90 cm, then either *x* or *x'* must be greater than the hallway threshold, which was 125 cm.

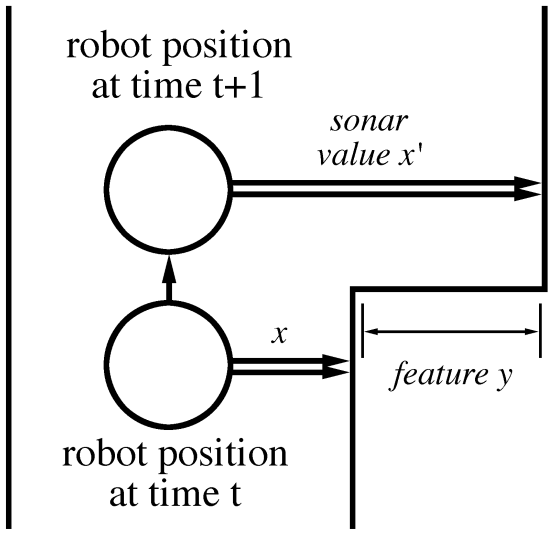

Figure 2: The 'feature' is the shift of the wall itself. The 'jump' is the difference between the values *x'* and *x*.

After a jump passes these criteria to qualify as a feature, the tolerances which allow a match of the detected feature to a mapped feature are very small. The feature has to occur within 30 cm, longitudinally, of its expected position along the wall of where the robot expects to detect the feature for it to be accepted. This 30 cm space is considered the window in which the robot is allowed to localize, and is kept small to avoid a false localization. The other strictly observed condition is the size of the feature. The detected feature must match the expected feature within 2.5 cm. That is, if we use the same terminology from our previous example, the difference between *x* and *x'* must be within 2.5 cm of the value recorded in the map for the expected feature.

The Feature Extractor was the only Data Analyzer that was needed for the project. The Node Handler and the Planner had no need of external information, so no separate Data Analyzers for those levels were constructed.

4.5 Node Navigator

The Node Navigator is a compilation of four different and mostly separate systems. These are the feature extraction system, the longitudinal positioning system, the lateral positioning system, and a world-view angle correction system. Each of these systems is to some degree dependent on the other three systems, but each is directly responsible for a different function of the navigation of the robot.

Longitudinal Localization

This brings us to the longitudinal localization system. If a feature is detected which is similar to an expected feature on the map; that is, the feature occurs within the 30 cm window and is of the appropriate size, the robot localizes its position. The robot localizes by marking its x-y position in internal encoders and calculating the x-y position of the middle of the hallway. This calculation is based on its concept of lateral position and the distance it detects from the jump which was used to localize at that feature. It then keeps track of distance traveled from that point using a projection of its position onto the center of the hallway using its concept of lateral position. All distances are measured from the most recent localization.

By using a projection point onto the hallway centerline and calculating distance along this projection, the exact longitudinal distance travel can be calculated, rather than just distance traveled from last point to current point which could introduce inaccuracies due to lateral drift caused by obstacle avoidance.

In contrast to several conventional indoor navigation systems, we limit our feature detector, using the filtering system described above, to highly reliable features. When such a feature is

detected, the robot is able to localize with confidence, obviating the need for multiple position estimates.

Lateral Localization

Calculating lateral positioning is more complex. The robot uses several different techniques to keep track of and update its lateral position, including maintenance of lateral position over multiple nodes, localization on detected features and mid-node lateral localizations.

The maintenance of lateral position over multiple nodes is non-trivial due to the representation of the map. When traversing a single node there is no problem, but when going from node to node there is an adjustment that must be made to the robot's concept of lateral position if there is a change in the width of the hallway. We decided to use a variable which represents the distance between the robot and the center of the node it is traversing. This variable is `*goffcenter*`. It is positive if the robot is to the left of the centerline, and negative if the robot is to the right of the centerline.

When the robot goes from one node into another without detecting a feature, it must update the `*goffcenter*` variable based on the current value of `*goffcenter*` and the expected jump values, which indicate if the centerline of the new node is different from the centerline of the old node. With proper calculation, a new and updated `*goffcenter*` can be obtained.

For example (Figure 3), if the robot is passing from node A to node B, it must make this calculation. From node A to node B the robot expects to see a positive jump of 15 cm on the left and 10 cm on the right. Therefore node B is 25 cm wider that node A, and offset slightly. Given that the robot is 12.5 cm to the left of the centerline in node A, it must calculate that it is 10 cm to the left of the centerline of node B in order to maintain an accurate representation of lateral position.

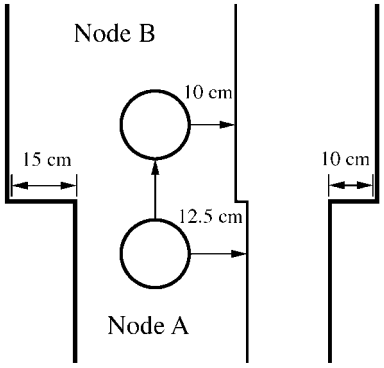

Figure 3: As the robot moves from one node to another, it must update its concept of lateral position.

The robot also localizes its lateral position when a feature is detected. This is easily done since it is assumed that if a feature is detected and matched to a feature in the map, the sonar values at the time the feature was detected are an accurate representation of the distance to the wall; from this the lateral position can be calculated.

Calculating the lateral position in the middle of the node is a more difficult proposition, since the robot must recognize the possibility of obstacles on either side which would give shorter sonar readings than the actual distance to the wall. Therefore very strict filtering is imposed. At several points in the middle of a node longitudinally, the robot examines the values of the left and right sonar. If the robot expects to see stable walls on either side of it (i.e. no doors or open spaces) and if the measured width of the hallway equals the mapped width of the hallway, then the robot may use these sonar values to localize its lateral position and reset `*goffcenter*`.

Safety Zone Maintenance

The robot is given the width of the node it is traversing as a "safe zone". That is, the robot will not allow itself to drift laterally farther than half the width of the node from the centerline of the node. Therefore if the absolute value of `*goffcenter*` exceeds half the width of the hallway, the robot will not travel farther to the side that it thinks it has drifted, and will proceed only if it believes that the direction of travel will take it back towards the center of the node it is in. This behavior allows the robot to avoid staircases, offices, lounges, elevators, and other "dangerous" areas.

Head Angle Maintenance

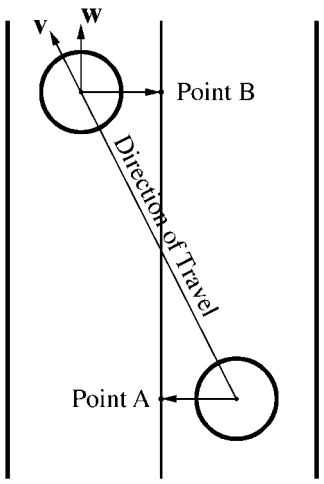

Figure 4: The robot has traveled in the direction of vector v, but correctly calculates the direction of the hallway as vector w by keeping track of its lateral position and calculating the

Finally, it is clear that all of these previously described systems will not operate with any kind of accuracy if the robot is not oriented directly down the center of the hallway. The angle difference between the robots encoders and "straight" down the hallway is reflected by the variable `*gheadangle*`, and is shown physically by the orientation of the robots turret. In order to maintain correct orientation of the robot, there are three separate systems which calculate the current value of `*gheadangle*` or make minor corrections to that value.

The first system of calculation of the `*gheadangle*` variable is contained in a function which calculates the angle when the robot makes a localization of lateral position. (Figure 4) By calculating the center point of the hallway at two different longitudinal positions, the robot may calculate the angle of the corridor which it is traveling. This method of calculating the head angle tends to be very accurate given accurate lateral localizations, but in situations where lateral localizations are sparse, this system won't function well.

The second system is a monitor of wall drift. If the sonar on one side varies over time in a monotonic fashion by a small amount when the robot expects that it is traveling straight down the corridor, a small correction is added to `*gheadangle*`. This system is the only one that will function well when the robot has open space or variable wall on one side, since this method only requires one wall to make adjustments, while lateral localization requires a wall on each side.

The third system is based on the ranges returned by pairs of sonars. Since the sonars of the turret are spaced at 22.5_ from each other, any head angle error in excess of 11.25_ will result in sonars adjacent to the 3 o'clock and 9 o'clock positions becoming shorter than the 3 o'clock and 9 o'clock sonars. Whenever these adjacent sonars are shorter, `*gheadangle*` is adjusted slightly. Filters are added to prevent adjusting the head angle while the robot is avoiding obstacles, which cause short sonar values.

5 Empirical Data

Empirical testing was conducted to demonstrate the reliable and robust navigation of the system. The robot was tested in three different buildings for at least five hours in each. None of the code was altered in any way between buildings or once testing had started. In order to perform these tests, a map of the building was constructed by the robot with a human guide. All of the maps used were generated by the robot itself, with user input limited to indicating topological features of the map, such as loops or dead ends. The empirical tests were then performed by executing a set of random navigation tasks.

Building	1	2	3
Total Test Time:	5:56	5:35	5:42
Number of runs attempted:	122	79	243
Number of runs completed:	121	79	242
Number of replans:	1	0	2
Avg. run duration in seconds:	148	257	84
Avg. run length in nodes:	28	31	11
Number of impacts:	0	0	0
Number of failures:	1	0	1

Table 1: Empirical results for NaviGates in three office environments

Terms:
Total Test Time This figure is given in hours and minutes
run One attempt to navigate to a node
replan This is the number of times the robot found its path obstructed and was forced to attempt a different route to the goal node
impacts Any contact with the environment
failure Number of times the robot was unable to achieve its goal node for any reason

The robot starts with the information of which node it's currently in and the direction it's facing. The robot then selects a random node and attempts to navigate to that node. When it achieves that goal, it selects another random node and cycles again. The process is stopped by the tester as infrequently as possible, and only for the purpose of hardware difficulty, such as battery recharging. Continuous run times are usually two hours or more. The robot is not monitored visually while running, but checked on every 20 to 30 minutes for failure.

All test data was collected during a normal working day. That is, in all the tests performed the robot had to deal with human traffic and shifting items in hallways. For example, in Building 1,

boxes were deposited on one side of a hallway after the robot had mapped that hallway. The robot was forced to negotiate a hallway which appeared to it much narrower than the corridor which was mapped.

It is apparent from the data provided that the success rate of the NaviGates robot is extremely high: out of 444 attempts, the robot correctly achieved its goal 442 times. This is a success rate of slightly higher than 99.5%. The two attempts where the robot failed involved situations where the head angle drifted significantly, and replanning did not help the robot. There were no impacts with either static or dynamic objects.

6 Related Works

There have been several recent robot navigation systems that have demonstrated success in a single environment [Gutierrez-Osuna & Luo 1996; Simmons & Koenig 1995; Nourbakhsh et al. 1995]. These robots all use a probabilistic approach (e.g. probabilistic progression and POMDP methods) to track the set of possible robot positions over time. This probabilistic approach has been motivated by the incomplete information that results from inaccurate sensors-- sonar and encoder data-- and inaccurate world models.

The idea of tracking multiple world states is indeed powerful; it has relevance to much more than simply robot localization. In contrast, our approach demonstrates that, even with real-world sonar and encoder data, there is no need for a multiple state approach in the domain of indoor robot navigation. Indeed, we have purposely tested this robot system in an array of office environments in order to demonstrate that this simple navigation technique can yield a practical solution.

An interesting approach would be a hybrid architecture that uses probabilistic progression only when necessary. For instance, if a robot navigator becomes disoriented due to external events, probabilistic approaches may allow the robot to eliminate possible states quickly, enabling a simpler navigation strategy to regain control. Not surprisingly, this hybrid approach would resemble the human cognitive discrepancy between the navigation strategy of a well-informed automobile driver and the strategy used by a lost driver.

Connell's SSS architecture uses a similar layered system for mobile robot control [Connell, 1992]. The SSS consists of three layers, very similar to the system used here. The SSS does not use separate data analyzers, however. On the other hand, the SSS system attempts to tackle a more ambitious task, that of a mobile robot which also determines goal directed behavior and complex planning. The NaviGates system is simply a point-to-point navigator: given a directive, the robot will reliably get to the desired position.

Brooks' subsumption architecture differs dramatically from the NaviGates system [Brooks, 1991]. Specifically, the NaviGates system uses a detailed map which is a representation of the environment of the robot. Brooks' creations specifically do not have an explicit representation of the external world. The NaviGates project demonstrates a simple, state-based architecture that avoids obstacles and achieves abstract goals with high reliability. Brooks' creatures react as well as NaviGates to environmental dynamics, but the subsumption architecture requires significant programming contortions in order to go to a specific place in the environment.

7 Conclusions

Simplicity itself is one of the theses of this paper. Our benchmark results demonstrate that NaviGates can reliably negotiate various new office environments with ease. Yet, the underlying system is extremely simple compared with conventional navigation systems. The AI techniques that have often been applied to indoor navigation problems are of great value in the context of high-level problem solving. However, NaviGates demonstrates that such powerful techniques need not be applied to the problem of indoor navigation. Of course, the true judge of these results will be the future, when we hope that others will publish indoor navigation results that outshine those of NaviGates.

References

[Brooks, 1991] Brooks. "Intelligence without representation", *Artificial Intelligence* 47:139-159, 1991.

[Connell, 1992] Connell, Jonathan H. SSS: A Hybrid Architecture Applied to Robot Navigation. In *Proceedings of the 1992 IEEE International Conference on Robotics and Automation,* Nice, France, May 1992.

[Gutierrez-Osuna & Luo 1996] Gutierrez-Osuna, R. and Luo, R. 1996. LOLA: Probabilistic Navigation for Topological Maps. *AI Magazine* 17(1): 55-62.

[Nilsson, 1996] Nilsson, Nils. Challenge Problems for Artificial Intelligence: Toward Flexible and Robust Robots, *Proceedings, Thirteenth National Conference on Artificial Intelligence.* pp. 1344-45, AAAI Press. 1996.

[Nourbakhsh et al. 1995] Nourbakhsh, I., Powers, R. and Birchfield, S. 1995. Dervish, An Office-Navigating Robot. *AI Magazine* 16(2).

[Simmons & Koenig 1995] Simmons, R. and Koenig, S. 1995. Probabilistic robot navigation in partially observable environments. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* , Montreal, Canada, 1995. Morgan Kaufmann.