6050

AIAA-94-1204-CP

N94- 30555

The Real-World Navigator

Marko Balabanović [*] Craig Becker [†] Sarah K. Morse [‡] Illah R. Nourbakhsh [‡]

Department of Computer Science
Stanford University
Stanford, CA 94305
lightning@cs.stanford.edu

S29-63
207652
p. 10

Abstract

The success of every mobile robot application hinges on the ability to navigate robustly in the real world. The problem of robust navigation is separable from the challenges faced by any particular robot application. We offer the *Real-World Navigator* as a solution architecture that includes a path planner, a map-based localizer, and a motion control loop that combines reactive avoidance modules with deliberate goal-based motion. Our architecture achieves a high degree of reliability by maintaining and reasoning about an explicit description of positional uncertainty. We provide two implementations of real-world robot systems that incorporate the Real-World Navigator. The Vagabond Project culminated in a robot that successfully navigated a portion of the Stanford University campus. The SCIMMER project developed successful entries for the AAAI 1993 Robotics Competition, placing first in one of the two contests entered.

1 Introduction

Current research on autonomous mobile robots has highlighted the difficulty of building robust, general-purpose navigation software. Problems with current systems include specificity for a particular environment, inability to deal with dynamic, real-world situations, and short life-spans, often due to the problems of cumulative sensory and control error.

We are studying the problem of robust navigation in the context of problems which can be decomposed as shown in Figure 1. In this decomposition, there is a task level, which provides the navigator level with a series of goals, and there is a physical robot capable of sensing and moving in the world. The navigator level directs the physical robot to achieve the goals of the task level while guaranteeing robust and reliable operation.

In this paper we describe a navigator level architecture called the *Real-World Navigator* that achieves robust robot control in a variety of environments. Given no domain-specific knowledge beyond a floor map, this Navigator should be able to move about an arbitrary office environment while preserving its sense of position.

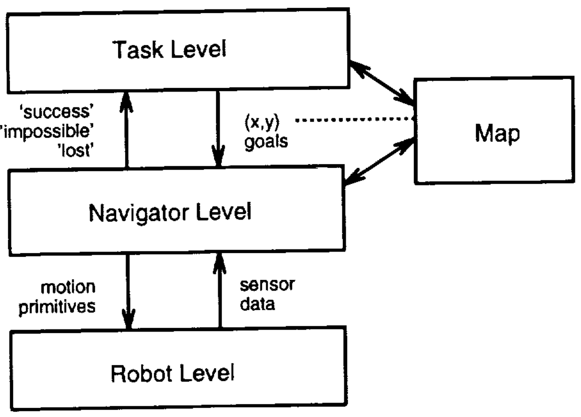

Figure 1: A three level decomposition of a mobile robot system

The sharp decomposition of Figure 1 allows us to use the Real-World Navigator with different physical robots and in different task domains. We will describe two successful implementations, involving different robots in several task domains and both indoor

[*]Funding provided by ARPA order 8607, grant NASA NAG 2-581

[†]Supported by an NSF Graduate Fellowship, research funded by ARPA grant N00014-92-J-1809

[‡]Funding provided by the Office of Naval Research under contract number N00014-90-J-1533

and outdoor environments.

1.1 Assumptions

In the descriptions of the architecture in the remainder of this paper, we make the following assumptions:

1. The system as a whole can be represented according to the interaction paradigm illustrated by Figure 1.

2. The goal coordinates that are passed down from the task level refer to locations in a shared map with bounded error.

3. The Navigator must have bounds on the error of the sensory and motion primitives through which it controls the robot level.

4. The control and sensory latencies of the robot level are appropriate to the dynamics of the environment; it is physically capable of responding to events and maintaining its safety in real time.

5. Any objects that are invisible to the robot's sensors must be present on the map. For instance, our robots have no way of detecting potentially deadly stairwells, so to ensure their (and our!) safety these areas must be marked on the map.

We make no further assumptions concerning the task or robot levels. For instance, it is possible for the task level to be a human operator.

1.2 Goals

The navigator level is an interface between the high-level goals of the robot system and the uncertainties and errors of the real world. As such, it must achieve the high-level position requests whenever they are reachable and, in the case of unreachable goals, it must signal failure. In addition, we expect the Navigator to react gracefully to a dynamic environment by avoiding both mapped obstacles and unmapped, visible obstacles in a smooth and efficient manner.

1.3 Overview

In the next section we present the general architecture of the Real-World Navigator without commitment to any specific task or physical robot. We then describe two implementations of the architecture with which we have solved various navigation tasks on different robot platforms. Next we discuss the limitations of the current architecture as well as extensions that may increase its robustness and applicability. Finally, we summarize work related to ours and present our conclusions.

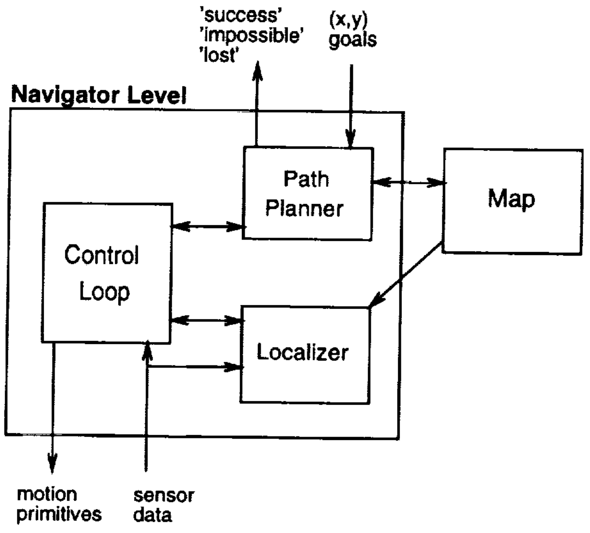

Figure 2: The Navigator consists of three subsystems: a path planner, a control loop, and a localizer. It also references an external map resource.

2 The Real-World Navigator Architecture

We consider the navigator level to be a collection of subsystems which communicate in a well-defined way. Figure 2 depicts the interaction of the subsystems that comprise the Navigator. Arrows in the figure represent data flow between the subsystems as well as between subsystems and the task and robot levels.

Briefly, the execution of a navigation task is as follows: the path planner receives goal coordinates from the task level. It then generates an appropriate plan using information from the map and invokes the control loop to execute each segment of the plan in turn. The control loop interacts with the physical robot and, if necessary, the localizer in order to reliably navigate each path segment. The localizer refers to both the raw sensor data of the robot and the geometric map.

We now discuss each of these subsystems in more detail.

2.1 Map

The map is a shared resource that is externally specified but referenced and manipulated by both the task and navigator levels. It maintains two different representations of the environment: one geometric, and the other based on the concept of highways.

The geometric representation is simply any description of the obstacles and free-space using an appropriate and agreed-upon coordinate system. For example, a reasonable geometric map for a robot that moves in a plane would be a polygonal representation of the projection of obstacles onto that plane. Note that this should be a map of physical space rather than configuration space because the localizer will compare the geometric map to sensor data.

In addition, the Navigator makes use of a highway-based representation of the map. The idea behind highways is to constrain the possible motions of the robot, both to simplify planning and to reduce the number of features that the robot must reliably sense.

Definition 1 (Highway Constraint) *Highways are possibly overlapping regions which decompose a subset of the free space of the robot's environment. The robot must always move within highways, and therefore can move between highways only through regions where they overlap.*

This constraint is related to highways in the real world. For example, planning a trip from San Francisco to Los Angeles would be much harder if we considered every possible back road instead of staying on the interstates. Using the interstates also means that we need only recognize off-ramps to move from one highway to another, rather than all the myriad types of intersection we might otherwise encounter.

Note that the highway map can either be provided by a human or automatically generated from the geometric map. Both methods have advantages. A human might want to design the highways to limit the robot's motion to certain parts of the free space (for example, to avoid a particularly busy hallway) or to hand-optimize certain motions. On the other hand, automatic generation of highways could save tedious work. There are several classical algorithms from motion planning that may be useful in automatic highway generation; examples are cell decomposition and visibility graph construction [Latombe, 1991].

2.2 Path Planner

Given the map and a goal position from the task planner, the function of the path planner is to compute a list of interim points through which the robot can move to achieve the goal. These interim points are passed in turn to the control loop, which guides the robot to each sub-goal. We assume that the path planner uses its knowledge of the geometric map to ensure that the points on this list can safely be connected by straight-line paths. Of course this assumption may be false in the face of unknown obstacles, but handling that contingency is the responsibility of the control loop which we describe below. We also assume that the path planner respects the constraints that the highway map imposes. Specifically, each of the interim straight-line sub-paths must lie completely within a highway.

Note that the choice of highway representations will influence the complexity of the path planner. For instance, suppose that we define highways as convex polygons that contain no known obstacles. Then a straight-line path connects any two points within a single highway region and planning reduces to finding a chain of overlapping highways that includes both the initial position and the goal position. On the other hand, if highways are arbitrary polygons and contain mapped obstacles, then planning a path within each highway becomes much more complex.

2.3 Control Loop

Given goal coordinates from the path planner, the control loop must direct the robot to that position. It is important that the control loop be reliable as well as complete. If it is not reliable, the robot will get "lost"; if it is not complete, the robot may fail to reach the goal point even if a path exists. Obviously, the control loop needs to interact with the physical robot, both to command changes in velocity and to receive sensor data. Furthermore, to achieve *reliable* motion, the control loop must model control uncertainty. Therefore, before we discuss the control loop itself we must define the control loop's representation of this uncertainty.

Definition 2 (Positional Uncertainty) *The positional uncertainty region U_t is defined as the region in which the robot is known to lie at time t.*

Note that there is nothing probabilistic about the uncertainty region—we know that the robot must lie within it. Also, note that the size of the region U will depend upon how well the robot can determine its current position. We assume that a robot has two general methods of position determination: by integrating its commanded velocity over time and by localizing based on sensory input and the geometric map. This means that the positional uncertainty is

```
while (¬Termination) {
        AcquireSensorData;
        if (DecideToLocalize)
            Localize;
        ComputeVelocity;
        CommandVelocity;
        UpdateUncertainty;
}
```

Figure 3: The general structure of the control loop

the result of two other types of uncertainty: control uncertainty (in the integration case) and sensory uncertainty (in the localization case).

Now that we have defined the uncertainty region, we can return to the discussion of the control loop. Figure 3 shows the high-level structure of the loop. We describe each component of the loop below.

`Termination` There are three possible ways for the loop to terminate:

1. The robot has achieved its goal. In the face of uncertainty, this means that $\mathcal{U}$ lies completely within the goal region (which encapsulates the goal point and allowable error).
2. The robot has become lost. This occurs when, in spite of efforts to localize based on sensory input, $\mathcal{U}$ remains so large that the robot cannot achieve the goal.
3. The robot has realized that there is no path to the goal. The control loop is constrained to travel only inside the current highway; therefore, this condition indicates that the robot has realized that an impassable obstacle is blocking the path to the goal.

`AcquireSensorData` In addition to acquiring sensor data from the robot level, it may be useful to fuse actual sensor data with "simulated" sensor data obtained by examining invisible, mapped obstacles in the geometric map.

Additionally, certain sensing processes such as vision may require too much processor time if done as part of a single-threaded control loop. Such sensor processes run asynchronously and `AcquireSensorData` would poll them as required.

`DecideToLocalize` This is the step in which the control loop must reason explicitly about the uncertainty region $\mathcal{U}$. This decision function tells the controller when it must re-localize and reduce the size of $\mathcal{U}$ in order to preserve goal reachability.

For example, if localizing is time-intensive, it would be appropriate to delay localization until the uncertainty region exceeds some threshold size. On the other hand, if localization is inexpensive, it would be beneficial to localize at regular intervals.

`ComputeVelocity` This step defines the system's control strategy, and could be implemented in many different ways. Its function is to combine obstacle avoidance with goal-directed behavior in order to calculate new velocities for the robot level motors. We require two guarantees: first, that the robot reach the goal when possible; and second, that it avoid contact with all sensed and mapped obstacles.

`UpdateUncertainty` As the robot moves, this routine extends $\mathcal{U}$ in accordance with the bounds placed on control uncertainty. This step is vital because it ensures the continuing validity of the uncertainty region, which must by definition always contain the robot's actual position.

2.4 Localizer

The success of the control loop depends on keeping the size of the positional uncertainty region $\mathcal{U}$ sufficiently small. Without the use of sensors, the size of $\mathcal{U}$ will, in general, only increase, since there is uncertainty in control. The role of the localizer is to use sensor data to compute a new region $\mathcal{U}_t'$ from the current region $\mathcal{U}_t$ and some set of sensor values. The hope is that $\mathcal{U}_t'$ will be smaller than $\mathcal{U}_t$, thus reducing the robot's positional uncertainty.

Note that the localizer may have internal state. In particular, this means that it may use a history of sensor values instead of a single instantaneous reading. The use of history can increase the effectiveness of the localizer by significantly decreasing the likelihood of a false localization.

3 The Vagabond Project

The Vagabond Project [Dugan and Nourbakhsh, 1993] was an effort to build a reliable outdoor navigator for the Stanford University Quadrangle. This outdoor arcade houses many of Stanford's departments and is composed of several walks that are flanked by regular pillars and sandstone walls.

Vagabond is a Nomad 100 mobile robot from Nomadic Technologies, Inc. It consists of a non-holonomic base which supports sixteen infrared sensors and sixteen sonar sensors. Its "brain" is an Apple Powerbook 170 that communicates with the sensor boards and motor controller through a serial link. The infrareds have an effective range of 0 to 15 inches while the sonars have an effective range of 15 to 150 inches.

3.1 Task Description

The Quad presents Vagabond with several great challenges. Many of the arcades are lined with six inch steps that would topple it, and, worse yet, the walks themselves have scattered potholes that are deep enough to trap it. In contrast to many forgiving office environments, the Quad allows Vagabond to actually destroy itself by mistaking its position. The dynamic character of this uncontrolled environment adds to the danger—at times bicyclists and pedestrians densely populate the walkways. Finally, direct sunlight in the Quad washes away infrared light, leaving Vagabond with sonar as its sole sensory input.

Given this very real environment, the task was to enable Vagabond to navigate successfully while avoiding the unmapped obstacles and the deadly steps. The final interface is precisely a navigator-level module. At the task level, the human provides initial position and orientation information and then supplies goal points through a graphical interface.

3.2 Implementation

Vagabond's map is a data structure with a polygonal description of every obstacle. The map differentiates visible from invisible obstacles. Overlaying this two-dimensional picture is a set of highways that are also represented as polygons. Figure 4 displays a portion of Vagabond's actual map. The filled polygons are mapped, visible obstacles while the unfilled polygons are mapped, invisible obstacles such as potholes. The shaded polygons depict the highways. Additionally, each highway has an associated speed limit that is based upon the general smoothness of its terrain.

Vagabond's path planner is an A* visibility graph search algorithm that treats both visible and invisible mapped obstacles as navigation points. The path planner finds the path with the fastest expected time of completion, based upon the top speed feature and the path length. The path planner then stores the path as a list of points to be achieved and sends the successive goal points to the control loop, waiting for success or failure and responding appropriately. In

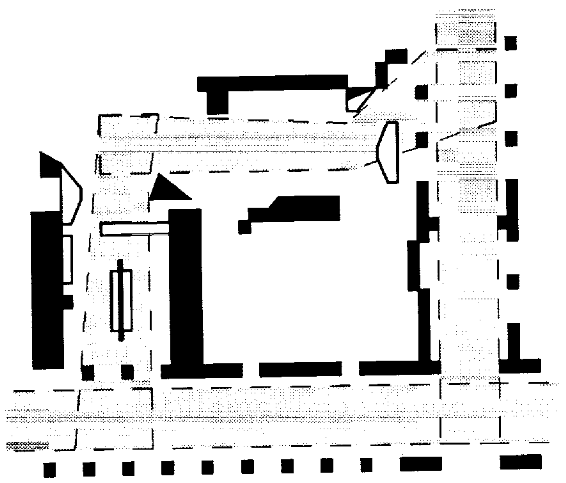
Figure 4: A section of the map of Stanford University Main Quadrangle, as used by Vagabond

the case of failure, the path planner recognizes that the goal point is not reachable from this highway, and so removes it from the map. It will then re-plan to find an alternate path to the task-specified goal point.

The control loop represents $\mathcal{U}$ as a rectangular region for the sake of computational efficiency. The `ComputeVelocity` routine employs a simple multi-level architecture with two behaviors: course maintenance and reactive obstacle avoidance. The course maintenance module resembles an aircraft course autopilot. It acts to reestablish the course and heading that define the line segment of travel between two successive subgoal points. The obstacle avoidance module modifies these ideal motion settings to avoid both sonar-detected obstacles and mapped invisible obstacles. Note that the obstacle avoidance module must ensure that the entire region $\mathcal{U}$ remains clear of any invisible obstacles on the geometric map.

The careful design of the interaction between these two modules is essential to preserving goal reachability as well as graceful behavior in the event of encountering an impassable obstacle. For instance, the desire to reestablish course should never override the refusal to allow $\mathcal{U}$ to overlap an invisible obstacle. However, intelligent obstacle avoidance demands more than a purely reactive decision system to avoid looping behavior.

The final ingredient of Vagabond's navigation system is the localization procedure. Localization is extremely time-intensive on Vagabond's hardware and is therefore minimized. The control loop only calls the localizer when the the size of $\mathcal{U}$ exceeds a threshold. The localizer has no state—it uses the current

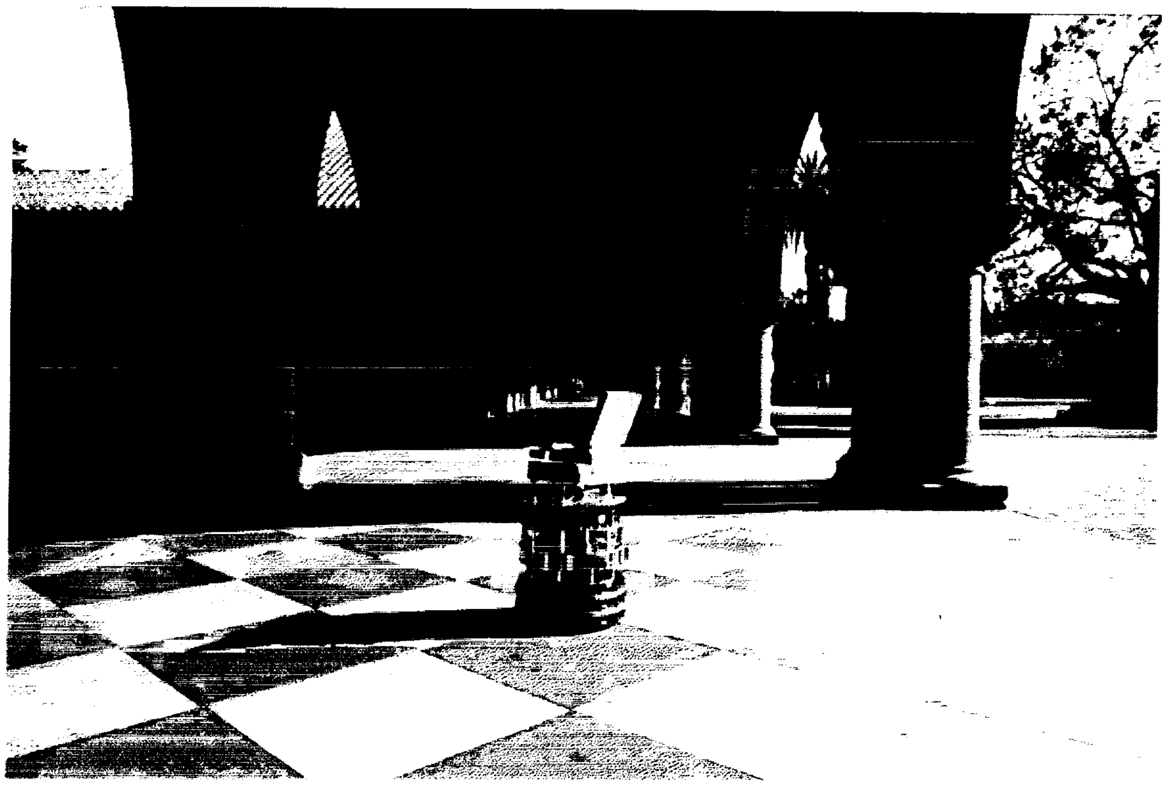

Figure 5: Vagabond navigating in the Stanford Main Quadrangle

instantaneous sensory input rather than a history of sensory data. It employs a deceptively simple scoring strategy that is surprisingly effective even in times of significant sensory occlusion (by people, bicycles, etc.). The key is the simple idea that any unexpectedly long real-world sonar value provides evidence for the elimination of a possible map position (sonars do not see through sandstone walls) while any unexpectedly short sonar value may be attributable to an occlusion by unmapped obstacles.

3.3 Results

One of the most desirable properties in a mobile robot is the ability to avoid self-destruction. For Vagabond, this meant always preserving its sense of position well enough to avoid the deadly steps. The architecture guarantees that no part of Vagabond's uncertainty region will intersect any mapped obstacle. Assuming that all steps are mapped (as they were), self-destruction could only occur after a false-positive localization. That is, Vagabond's localizer would have to localize to an incorrect location, thus violating the architectural assumption that the robot is always within $\mathcal{U}$.

Our goal was to produce a truly robust navigator. To this end, the entire development and testing process used the real world, never a simulator. We tested the final Vagabond system intensively in the Quad environment, both during quiet times (e.g. weekdays in summer) and in times of extremely dense traffic (e.g. between classes in the autumn). False localization occurred extremely infrequently during testing and never continued long enough to result in a deadly move. The only recurring cause of false localization involved onlookers who formed human walls parallel to and offset from the walls of the Quad. Sonar cannot differentiate such human walls from real walls. Happily, group dynamics seem to render human walls too transient to be a serious threat.

In contrast, Vagabond's most common failure resulted instead from an inability to localize successfully. This would eventually lead to an uncertainty region so large that it rendered any further movement impossible. In these cases, Vagabond would stop and return the "lost" termination condition to the task level. In our tests, this condition occurred in approximately 10% of all cases in which the user requested Vagabond to achieve a certain position on its map. Vagabond would reach the destination point and return success in the remaining 90% of the cases.

Vagabond moved at a slow walking pace (12 inches per second on average), typically covering distances of $\frac{1}{2}$ mile per task.

4 The Scimmer Project

The Scimmer[1] Project was organized to develop a successful entry for two contests at the AAAI Robotics Competition held in Washington, D.C. in July, 1993. The contests involved simple navigation tasks in contest arenas that simulated real-world conditions using gray office partitions, white boxes, and actual office furniture.

Scimmer is a Nomad 200 robot from Nomadic Technologies, Inc. (Figure 6). It has a three-wheel synchronous drive non-holonomic base, on top of which is an independently rotating turret housing sensors and on-board computation. The sensors include 20 pressure-sensitive bumpers, 16 sonar sensors, 16 infrared sensors, a structured light vision system consisting of a laser and CCD camera, and a second CCD camera linked to a frame-grabber for vision processing. We ran all software on-board using a 386-based PC system.

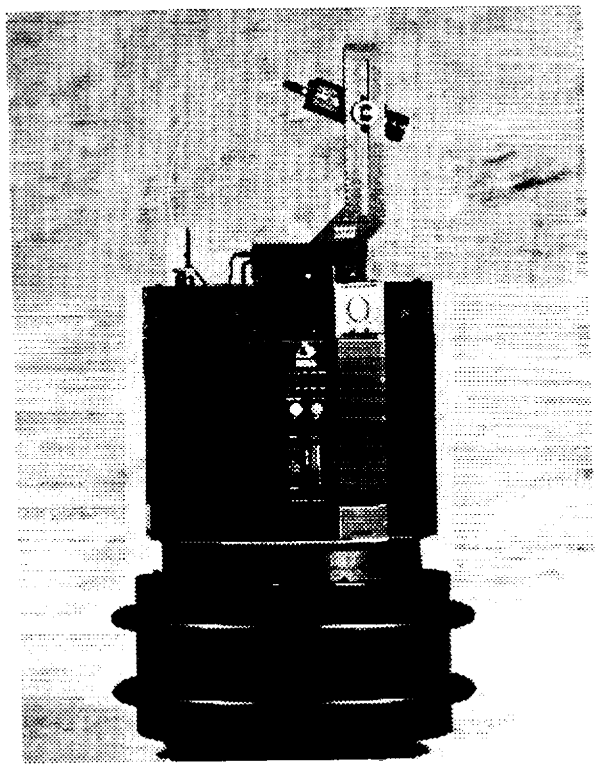
Figure 6: The Nomad 200 robot

4.1 Task Description

Contest I The environment was a large "warehouse" with an enclosed office at one end. Scimmer's task was to escape from the inner office, then race to the far wall of the warehouse. The office contained typical office furniture (e.g. file cabinets and tables) while the warehouse was cluttered with white boxes.

Contest II The environment was a simulated office building with rooms and hallways connected in a fairly typical layout. White boxes were scattered around as obstacles. The goal of the contest was to find a coffee pot and deliver it to a specified room. At the start of the contest, the robot received a map of the office building (divided into quadrants), its starting quadrant, the quadrant containing the coffee pot, and the destination room for the coffee pot. Note that the robot begins the contest with an enormous amount of uncertainty as to its initial location, so a major part of this contest was the initial localization.

4.2 Implementation

Contest I required domain-dependent code for escaping the inner office, followed by an implementation of the control loop subsystem to reach the goal region. Readers interested in the control loop implementation are referred to [Balabanovic *et al.*, 1993]. Our Contest II entry provides a more complete implementation of the navigator level; this is the implementation we now describe.

Scimmer's geometric map is a simple line drawing, with each line denoting a wall in the real world. There were no invisible but mapped obstacles (such as sharp drop-offs) in the environment. The highway map consists of both highways and nodes. Highways are polygons of free space (barring any unmapped obstacles). The nodes are simply just intersections between highways that provide task-level goal regions to facilitate movement between highways while simplifying path planning.

Scimmer's planner uses a best-first search algorithm to find the shortest path from one node to another. The planner then feeds the control loop one node at a time. Because of the nature of the task, the planner does not re-plan if the control loop fails to achieve its subgoal. Instead, it returns impossible to the task planner. Consider the problem: we're trying to find a coffee pot in one quadrant of the map. There could be multiple rooms in that quadrant; if we find a blockade along the way, we might want to

[1] Sarah, Craig, Illah and Marko's Most Excellent Robot

change the order in which we visit those rooms. Since this is a high-level task decision, control must return to the task level.

The ComputeVelocity routine that combines these desires frequently commands the robot to move at the motor controller's top speed of 20 inches per second, as the contests were timed. Once the robot is within the goal region, the control loop exits to the planner, signalling success. In the case of failure, the control loop exits signalling impossible and the planner removes that highway from the map.

SCIMMER deals with positional uncertainty in a very simplified way. Upon reaching a goal node, the control loop decides whether it should localize by referring to the map, on which all nodes are marked either "localize" or "don't localize". We entered this information manually, basing our decisions upon the degree to which different nodes would be effective places to localize. For example, nodes in the middle of a long hallway would be very unreliable whereas nodes at an intersection of three of four highways would be promising.

SCIMMER's localization, as opposed to Vagabond's uses history. As it moves, it builds a bitmap representing the objects it has detected over time with its laser range-finder. The localizer uses a general shape matching algorithm to find the best match of this sensor history against a bitmap representing the known obstacles in the world. The shape-matching metric used is the Hausdorff distance, following the general algorithm presented in [Huttenlocher *et al.*, 1991].

Once again, we avoided the use of simulation altogether during the development of the SCIMMER contest entry. Success demanded fast, robust operation in the actual contest environment—therefore, we chose this environment as our development environment.

4.3 Results

Contest I SCIMMER achieved first place. It successfully avoided all obstacles and quickly followed a smooth path to the final goal.

Contest II SCIMMER was one of only two contestants to successfully localize itself at the start of the contest without assistance. It began to follow its plan to reach the projected location of the coffee pot, but an unfortunate operating system problem caused the robot to crash a short distance from that goal.

5 Limitations and Extensions

Clearly, there are domains to which this architecture simply does not apply. For instance, the problem of visually recognizing a coffee pot requires a specific solution that does not fit in our three-level decomposition. Indeed, any problem that does not require navigation between well-specified destination points will not benefit from our architecture.

A more serious limitation involves the explicit uncertainty region that the navigator level maintains. Although the control loops we have implemented based velocity decisions on the size of $\mathcal{U}$, among other parameters, neither of our systems incorporated reasoners that would move the robot exclusively to shrink $\mathcal{U}$. One can imagine a case in which the robot needs to move from A to B, yet the direct path is so sparse that the robot must first move from A to landmark C, where the size of $\mathcal{U}$ can be bounded, and then on to B. Our current implementations would fail in this situation because neither Vagabond nor SCIMMER's path planners account for the size of $\mathcal{U}$. A possible solution is to use a path planner that predicts the localizer's reliability at any given map location.

Another significant limitation of our architecture is that it fails to provide any mechanism allowing the robot to improve its performance over time by learning more specific information about its environment. The obvious solution to this deficiency is to allow the robot to modify its geometric map during navigation, thus attaining an increasingly accurate representation of its environment over time. In reality, this is an extremely complex issue that currently has no satisfying solution. Today's robotic sensory input is too imprecise and robotic common sense too undeveloped to allow a robot to make useful decisions concerning the transience of unexpected obstacles.

Finally, the robustness of any navigation system depends largely on the richness and reliability of its sensors. Sensors such as sonar transducers are useful in many situations, but their very nature renders them unable to detect many hazards (such as downward steps and narrow chair legs) that exist in the real world. It seems useful, then, to explore other types of sensors which do not suffer from these limitations.

One could imagine designing a specific "downward step sensor" using short-range proximity sensors or touch sensors trained on the floor. In fact, ground-level tactile sensors seem to complement sonar well, detecting many of the low-lying obstacles that otherwise evade detection.

Perhaps a better solution is an increased reliance on vision. Richer, more flexible sensing would improve the performance of our Navigator by allowing more precise localization and would allow us to reduce control error by receiving constant environmental feedback while moving. Our system makes it easy to incorporate such enhanced sensing, and we believe its development is vital to eventually building truly robust systems.

6 Related Work

Researchers from both the robotics and the artificial intelligence communities have been addressing the challenges of mobile robotics for some time. However, their approaches and the focus of their research have been quite different.

The robotics community has successfully addressed the challenges of many of the components of a robot architecture. Most of the subsystems we posit as part of the Real-World Navigator have been extensively researched. Crowley [1989] develops a localizer that uses ultrasonic range data to find a robot's position on the map. His approach involves an abstraction step in which the localizer extracts potential line segments out of the sonar data. Takeda and Latombe [1992] address the problem of path planning under the specific assumption that the executor will use sensory feedback to localize during path execution. Their sensory uncertainty field computation ascribes to each possible robot position a measure of the robot's ability to localize using sensory input at that position. For example, a corner would receive a much higher score than a featureless wall.

In contrast, the AI community has witnessed a recent spate of work on architectures for robotic agents. However, these agent architectures often blur the distinction between the task level and the navigator level. As a result, most AI robot architectures do not make the strong claim that is implicit in the Real-World Navigator: that the navigation component can be fixed across application domains. Instead, a common approach is to allow higher-level components to activate, deactivate or parameterize navigation processes. Recent examples include ATLANTIS [Gat, 1992], SSS [Connell, 1992] and [Saffiotti, 1993]. A further alternative is to compile beforehand a reactive structure that will execute a plan at run-time (again, navigation is neither a fixed component nor a necessary part of these structures). Examples include [Kaelbling and Rosenschein, 1989], [Schoppers, 1987] and [Nilsson, 1994].

Another important difference between the Real-World Navigator and many other current approaches is our need for a geometric map, enabling explicit maintenance of a positional uncertainty region. A popular alternative is to navigate using robust reactive routines such as wall-following and corridor-following, and to provide a connectivity map in terms of these motion primitives as well as high-level sensory primitives (e.g. T-junctions, doorways). This technique, which evolved from the subsumption architecture [Brooks, 1986], has been successfully demonstrated by [Gat, 1992] and [Connell, 1992]. The clear advantage of these systems is that they do not require a geometric map of the environment. However, the software is usually quite domain-dependent, and any change of domain requires a great deal of rewriting. In addition, many extensions (such as avoiding mapped, invisible obstacles) do not fit neatly into this framework. Finally, it is difficult to see how such a system would be able to effectively determine that it was lost.

Three projects at Stanford are worth noting here. The Logic Group formalizes the concept of planning with incomplete information and designs a framework in which an agent may act explicitly to decrease its uncertainty [Genesereth and Nourbakhsh, 1993]. Another project focuses on landmark-based navigation where assumptions about sensing and control within specific landmark regions are used to reduce planning to a polynomial-time problem [Lazanas and Latombe, 1993]. Finally, the AIbots project [Hayes-Roth *et al.*, 1993] addresses issues involving the interface to the task level by investigating the integration of a *cognitive level*, which is currently a BB1 blackboard system dealing with task planning and deadline management, with a *physical level* which includes a path planner and a navigator.

7 Conclusion

We have introduced an architecture for mobile robot control which addresses the problem of navigation. In addition to demonstrating robust behavior in dynamic, real-world situations, the two applications we have described show that the architecture is independent of the task domain, the environment and the robot platform.

Our belief is that the success of these applications is due not only to the design of the individual components, but also to the design of the architecture itself. This allows reuse of the architecture over many different tasks, its tested framework considerably decreasing the difficulty of building robust, general-purpose navigation software.

The Real-World Navigator provides a solid foundation on which we can build highly effective real-world mobile robot applications.

Acknowledgements

Benedict Dugan played a major role as one of the two developers on the Vagabond Project. Thanks are also due to Michael Genesereth, Barbara Hayes-Roth, Jean-Claude Latombe and Nils Nilsson for their helpful comments and suggestions. Finally, we would also like to thank David Zhu, Jim Slater and John Slater at Nomadic Technologies for all their help in times of need.

References

[Balabanovic *et al.*, 1993] Marko Balabanovic, Craig Becker, Erann Gat, Steven Goodridge, David Hinkle, Ken Jung, Sarah Morse, Illah Nourbakhsh, Harsh Patlapalli, Reid Simmons, and David Van Vactor. The winning robots from the 1993 robot competition. *AI Magazine*, Winter (In Print) 1993.

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

[Connell, 1992] Jonathan H. Connell. SSS: A hybrid architecture applied to robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1992.

[Crowley, 1989] James Crowley. World modeling and position estimation for a mobile robot using ultrasonic ranging. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1989.

[Dugan and Nourbakhsh, 1993] Benedict Dugan and Illah Nourbakhsh. Vagabond: A demonstration of autonomous, robust, outdoor navigation. In *Video Proceedings of the IEEE International Conference on Robotics and Automation*, 1993.

[Gat, 1992] Erann Gat. Integrating planning and reacting in a heterogenous asynchronous architecture for controlling real-world mobile robots. In *Proceedings of the 10th National Conference on Artificial Intelligence*, 1992.

[Genesereth and Nourbakhsh, 1993] Michael Genesereth and Illah Nourbakhsh. Time-saving tips for problem solving with incomplete information. In *Proceedings of the 11th National Conference on Artificial Intelligence*, 1993.

[Hayes-Roth *et al.*, 1993] Barbara Hayes-Roth, Philippe Lalanda, Philippe Morignot, Karl Pfleger, and Marko Balabanovic. Plans and behavior in intelligent agents. Technical Report KSL-93-43, Stanford University Knowledge Systems Laboratory, 1993.

[Huttenlocher *et al.*, 1991] Daniel P. Huttenlocher, Gregory A. Klanderman, and William J. Rucklidge. Comparing images using the Hausdorff distance. Technical Report CUCS TR 91-1121, Cornell University Department of Computer Science, 1991.

[Kaelbling and Rosenschein, 1989] Leslie Pack Kaelbling and Stanley J. Rosenschein. Action and planning in embedded agents. *Robotics and Autonomous Systems*, 6(1–2), June 1989.

[Latombe, 1991] Jean-Claude Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, 1991.

[Lazanas and Latombe, 1993] Anthony Lazanas and Jean-Claude Latombe. Ladmark-based robot motion planning. In C. Laugier, editor, *Geometric Reasoning for Perception and Action*, pages 69–83. Springer-Verlag, Berlin, 1993.

[Nilsson, 1994] Nils J. Nilsson. Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, To Appear 1994.

[Saffiotti, 1993] Alessandro Saffiotti. Some notes on the integration of planning and reactivity in autonomous mobile robots. In *AAAI Spring Symposium on Automated Planning*, 1993.

[Schoppers, 1987] M. J. Schoppers. Universal plans for reactive robots in unpredictable environments. In *Proceedings of the Tenth International Conference on Artificial Intelligence*, pages 1039–1046. International Joint Committe on Artificial Intelligence, 1987.

[Takeda and Latombe, 1992] H. Takeda and J.C. Latombe. Sensory uncertainty field for mobile robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1992.