Situationally Driven Local Navigation for Mobile Robots

by

Marc Glenn Slack

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science and Applications

Approved:

David P. Miller, Chairman

Morton Nadler

John W. Roach

Sallie Henry

Clifford A. Shaffer

April, 1990
Blacksburg, Virginia

Situationally Driven Local Navigation for Mobile Robots[1]

by
Marc Glenn Slack

Committee Chairman: David P. Miller
Computer Science and Applications

Abstract

For mobile robots to autonomously accommodate dynamically changing navigation tasks in a goal-directed fashion, they must employ navigation plans. Any such plan must provide for the robot's immediate and continuous need for guidance while remaining highly flexible in order to avoid costly computation each time the robot's perception of the world changes. Due to the world's uncertainties, creation and maintenance of navigation plans cannot involve arbitrarily complex processes, as the robot's perception of the world will be in constant flux, requiring modifications to be made quickly if they are to be of any use. This work introduces *Navigation Templates* (or NaTs) which are building blocks for the construction and maintenance of rough navigation plans which capture the relationship that objects in the world have to the current navigation task. By encoding only the critical relationship between the objects in the world and the navigation task, a NaT-based navigation plan is highly flexible; allowing new constraints to be quickly incorporated into the plan and existing constraints to be updated or deleted from the plan. To satisfy the robot's need for immediate local guidance, the NaTs forming the current navigation plan are passed to a transformation function. The transformation function analyzes the plan with respect to the robot's current location to quickly determine (a few times a second) the locally preferred direction of travel. This dissertation presents NaTs and the transformation function as well as the needed support systems to demonstrate the usefulness of the technique for controlling the actions of a mobile robot operating in an uncertain world.

---

[1]This work was supported in part by a grant from the Jet Propulsion Laboratory under a contract from the National Aeronautics and Space Administration, and by a grant from the Naval Surface Weapons Center.

Acknowledgements

There are many people that have had a great impact on my life throughout my college years and as this dissertation represents the termination of those years I would like to thank many of them. First I thank David Miller who for some unknown reason saw something in me five years ago which led him to believe that I could make it as a graduate student and has been my advisor and mentor ever since. His guidance and encouragement in the formation of my work has been invaluable. David has also provided me with many unique opportunities: jobs in California, conference trips, and of course that special summer in Dahlgren which I will never forget (though I have tried). He and his wife Cathryne Stein have also been greatly valued friends and it is that friendship that I value the most.

Jim Firby (Dr. Graphics) provided many critical reviews of my work helping me to focus the amorphous ideas of the early stages of this work into the resulting tangible concepts. Throughout, his casual demeanor and seemingly omniscient grasp of Artificial Intelligence has been an inspiration as well as a valuable resource. In addition to his professional help, Jim and his wife Joanne have proven to be caring friends and extraordinary wine-drinking companions.

I owe a special thanks to Sallie Henry for finding my FIFO-SJF bug in that operating systems program many years ago, for playing a critical role in my admittance into graduate school, and for her friendship.

Where would any person writing a dissertation be without their readers? I have had the great benefit of having become acquainted with Joanne Firby "English Teacher Extrodinaire". Her multi-colored multi-media comments will be appreciated by all who venture to read further. I also thank Allison Moser, and Eran Gat for both their friendship and the time they spent reading drafts of my dissertation.

I would also like to thank a number of my friends who throughout the college years supplied plenty of entertainment and friendship: Kent and Sharon Kunkel, Skipper Payne, Bob and Angie Mooney, Joan Kalnitsky-Aliff and Rick Aliff, and John Lewis. A special

thanks to Steve Wake for his companionship throughout the days of classes and qualifiers and especially for marrying Debora, a close friend and the best thing that ever happened to chocolate, rich dairy products, and other fattening and tasty culinary delights.

Thanks to my family: Gary and Connie Slack for their moral and financial support; my sister Barbara for being herself and for the free ski condo in Colorado; and, of course, my grandparents, George and Millie and Glenn and Lizetta, for their continuing stability.

Finally, I thank Pam (my wife of almost six years) for helping me through the stress and for living with me even during those dark ~~days~~ months of writing when I had no real life, only a preoccupation. To a large degree, it was her love and support that made this accomplishment possible.

Contents

Chapter 1

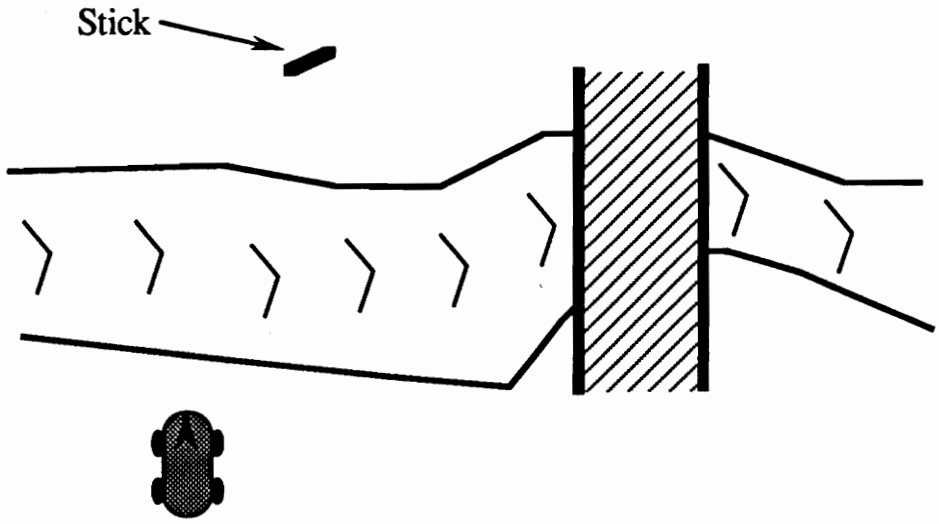
1. Local Navigation

Mobile robots operating in the "real-world" (hereafter simply referred to as the *world*) will be expected to perform a variety of tasks such as, delivering mail, getting a cup of coffee, or walking the dog. To accomplish such tasks a robot must be able to move through the world from place to place (e.g., "go to 555 Stone street", "go to the coffee room", or "go to the top of the hill"). Moving from place to place requires that the robot be sensitive to both the known environmental constraints (e.g., "avoid Gray street", "use the sidewalk", or "do not hit that rock"), as well as the inevitable uncertainties that occur when moving about the world. There may be objects in the hall; other agents may be in pursuit of unknown goals; sensors will not provide precise readings; even the robot's actuators will not function with complete certainty. While in service of its current task, a robot must be able to accommodate environmental constraints as well as respond to the world's uncertainties in a reasonable and timely fashion.

1.1. Introduction

One goal of mobile robotics research is the creation of a robot navigation system which manifests both smooth and goal-directed behavior in the robot being controlled. A robot is said to display smooth behavior when it moves in a continuous fashion, with one motion flowing into the next. A robot is said to display goal-directed behavior when its actions appear to be in direct service of its current task. Consider a situation in which a bridge connects two banks of a creek and a robot has the task of fetching a stick from the opposite bank (see Figure 1.1). As part of this task, the sub-task of moving over to the stick must first be accomplished. A robot displaying goal-directed behavior in this situation would

move directly to the bridge, cross the bridge and then move over to the stick (assuming that the robot should not get wet, the creek was not dry, the robot cannot jump, etc).



Crossing a creek to retrieve a stick

Figure 1.1

This dissertation focuses on the problem of navigation for mobile robots. For the purposes of this discussion, several pertinent terms are defined. A particular instance of needing to move between two places characterizes a *navigation task*. The process of determining the way to move between two places, accounting for internal and external constraints, is referred to as *navigation*. The mechanism which directs activity (charting courses, creating plans, etc.) during navigation between two places is the *navigator*. In summary, the navigator directs the process of navigation in the service of a particular navigation task.

Traditionally, navigation has been approached by creating a detailed internal model of the navigation task, using that model to create a plan which accomplishes the task, and finally executing that plan. In theory, this three step navigation process (sensing, planning, and acting) allows a robot to display the type of smooth goal-directed behavior desired. In practice, however, such approaches lack robustness in the face of the world's uncertainty, requiring plan execution to be interrupted frequently and the sense-plan-act cycle to be repeated. Repeated cycling through the sense-plan-act loop to accomplish tasks could

potentially be a robust solution to the problem, as it allows the system to handle the world's uncertainties implicitly; however, the computational complexity of such systems is typically too great to attain the cycle rates needed to employ such an approach effectively. Therefore, while such approaches are goal-directed in that the resulting action does move the robot directly to the stick via the bridge (in the above "fetching the stick" example), the action does not display the desired smooth behavior. This shortcoming becomes especially apparent when the state of the world is in flux.

New approaches to controlling the actions of robots take a more reactive view, never committing to a particular action for any significant period of time. Consider the sequence of actions that might occur if a navigation system built with this view is presented with the above task of fetching a stick. In such a situation the robot might initially proceed directly toward the stick. However, upon sensing that water is blocking the way, and "knowing" that getting wet is "bad", the robot would turn to the right or the left and proceed along the bank of the creek. In the case in which the correct direction is chosen, the robot would cross the bridge and move to the stick. In the other case, the robot might proceed for some distance along the bank, turn around, and then proceed back along the bank to the bridge. While this may seem a bit undirected, such reactive approaches generally require relatively little computation and commit the robot to a particular action for only a brief period of time. Thus, a reactive system implicitly handles the world's uncertainty by continually updating its actions based on the robot's current perception of the world. This results in the type of smooth behavior desired in a robot. However, due to limited representational and reasoning powers, purely reactive systems are not goal-directed, relying on physical search through the environment, as well as the environment's cooperation, to achieve high level goals (such as fetching a stick).

Hence, while traditional approaches allow reasoning about the implications of environmental constraints with respect to some navigation task, the plans generated are unable to account satisfactorily for the world's uncertainties. On the other hand, while reactive approaches are capable of operating at the time scale of the world and keeping the robot moving and out of trouble (most of the time), they lack directedness and suffer from an insufficient ability to reason about the impact of environmental constraints on schemes for accomplishing navigation tasks. Humans are, in general, very good at handling navigation tasks, due largely to our ability to represent and to reason about the situation at an abstract level as well as to our ability to use known routines for acting based on

solutions found at this abstract level. A person walking down a hallway employs learned routines when approaching a blind intersection, passing another person, or entering a door. We hypothesize that almost every situation[1] encountered is only a slight variation on some previously encountered situation for which a solution is known. The ability to exploit situational knowledge is the motivating force behind the emergence of situationally-driven robot control systems. This dissertation describes the application of situationally-driven control to the robot navigation problem. By giving a reactive system the guidance provided by situational control, a robot is able to exhibit both smooth and goal-directed behavior.

1.1.1. Defining the Problem

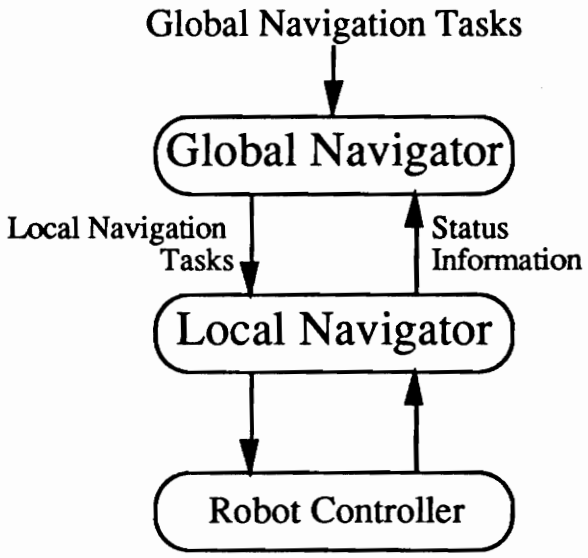
The general problem for a robot navigator is to move the robot in service of the current navigation task, while accounting for both internal and external constraints. This requires knowledge of the robot's local surroundings as well as knowledge about places and features that are beyond the robot's local space. For example, consider walking to the store. At one level you may be actively negotiating a sidewalk. However, at another level you may know that the sidewalk leads to an intersection and that the store is across the intersection. These two levels of reasoning and representation associated with accomplishing navigation tasks lead to a natural division of the navigation problem into two smaller interacting problems and the formation of two interacting navigation systems: a local navigation system operating on local navigation tasks and a global navigation system operating on global navigation tasks (see Figure 1.2). The global navigation system accepts global tasks, such as "go to the coffee room", and produces a sequence of local navigation tasks which it uses to direct the actions of the local navigation system (e.g., "go out the door", "go up the hall", etc.).

For the purposes of global navigation, space can be reasoned about as the connection of highly idealized symbolic representations of physical space {McDermott84}. For example, a hallway can be represented as a long rectangular object wide enough for a robot to pass through. Navigation plans at this level can be constructed from connections of symbolic objects (e.g., "just past Joe's office, turn left, the coffee room is the second door on the right"). At the global level, there is no mention of the way the robot is oriented with

---

[1] A situation is a combination of environmental constraints and the current task.

respect to the hall or, by definition, what unforeseen obstacles will be encountered along the way; such issues are deferred to the local navigation system.



Interaction between local and global navigators

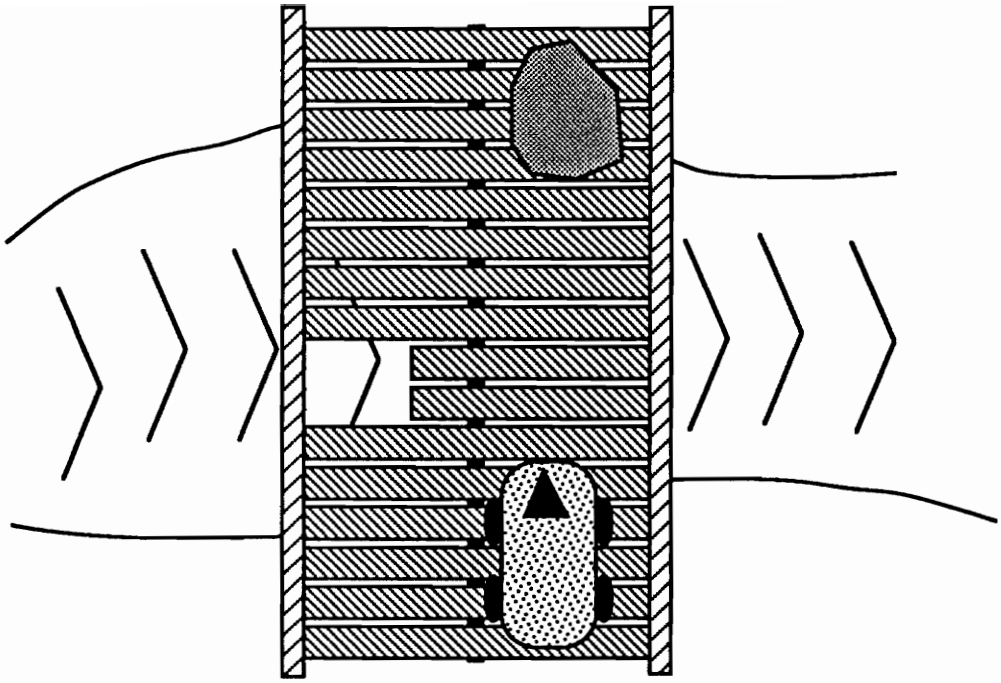
Figure 1.2

Locally, global issues may be ignored, since they play only a secondary role (e.g., knowing that the global navigation task is to "go to the coffee room" is, for all practical purposes, irrelevant to the problem of negotiating a hallway). The local navigator is more concerned with sensing the environment to acquire the information needed to achieve the current goal, reasoning about the physical constraints of moving a robot in physical space, and dealing with the immediacy of the world's uncertainties. While both local and global navigation are complicated problems involving many subtle issues, this dissertation focuses specifically on the issues of local navigation. It assumes the existence of a "higher level" system (global navigator) which generates a stream of local navigation tasks.

Although the separation between global and local navigation is somewhat arbitrary, the following definition of a local navigation task is suggested: A local navigation task is a globally primitive navigation task requiring that the physical interaction of the robot with the world be reasoned about (e.g. "go up this hall", "head in that direction", or "move over to that place"). To a certain degree, local refers to the space around the robot that the robot can either sense directly of has sensed recently to derive an accurate internal model. The

job of the local navigator is roughly defined as follows: Given a local navigation task, a method of acquiring local sensor data, and a method of directing the activities of the robot, move the robot in such a way as to accomplish the navigation task in a smooth and goal-directed fashion.

Recall the navigation task of fetching the stick from the other bank of the creek. Accomplishing the sub-task of simply moving over to the stick has aspects of both the global and local navigation problems. Globally, the task must be subdivided into an appropriate sequence of local navigation tasks (e.g., "move over to the bridge", "cross the bridge", and "move over to the stick"). The global navigation system hands each of these globally primitive tasks in turn to the local navigation system. One local task directs the robot to move across a bridge; thus, requiring the local navigation system to reason about the robot's physical interaction with the bridge, as well as the implied environmental constraints associated with the situation. For example, consider, the situation depicted in Figure 1.3, where in addition to the known boundary of the robot's actions by the rails of the bridge, there are also unanticipated constraints to be handled by the local navigation system: the bridge has a hole as well as a large obstacle at the far end of the bridge. Such unanticipated constraints are typical of the types of issues that must be handled by the local navigation system, as such information is simply not available to the global navigation system at the time the global plan was constructed. Furthermore, even if they were known to the global navigation system, unless the constraints made an alternate route preferable to crossing the bridge, there is nothing the global navigation system has to say about the way to interact with these constraints, since such constraints require reasoning about the physical interaction between the robot and the environment. The result is that the global navigation system is not concerned with the way in which the robot should proceed across the bridge and the local navigation system is not concerned with whether or not to cross the bridge. Rather the global navigation system determines that it is necessary to cross the bridge and the local navigation system determines the way in which the bridge should be crossed.



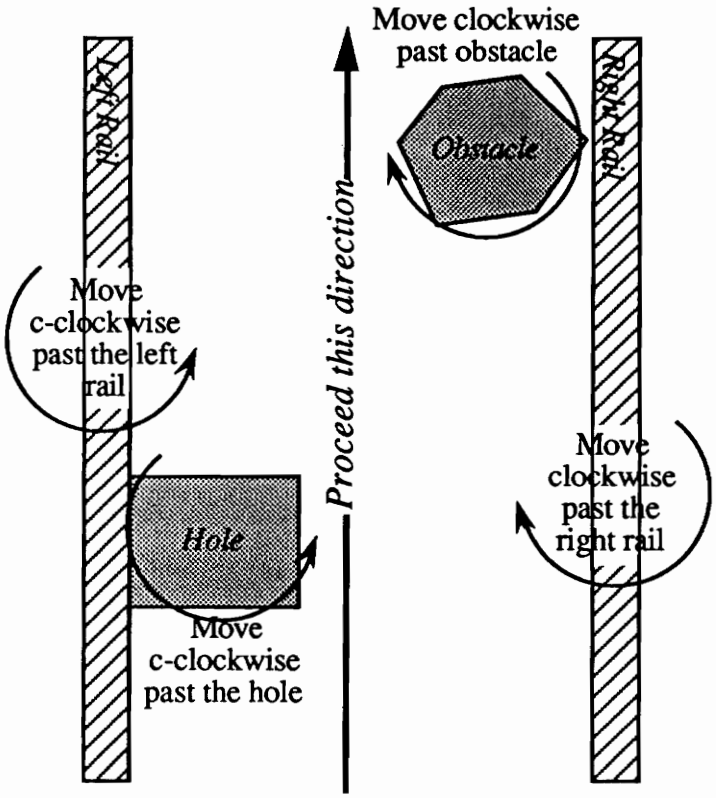
Local navigation task: "Cross the bridge"

Figure 1.3

1.1.2. The Approach

This dissertation presents a navigation system (LNav) capable of handling navigation tasks such as "go in *that* direction", "follow *this* path", or "go over to *that* place". LNav maintains a model of the robot's local environment, built incrementally and updated as the robot moves and/or sensors provide new or refined information about the environment. Sensor information is gathered to support the navigation task through a mechanism allowing LNav to direct the activities of the sensing system. The resulting representation is used to create and maintain a rough navigation plan for accomplishing the current navigation task. An example of an LNav navigation plan for crossing the bridge in the above example is shown in Figure 1.4. Notice that the plan captures only the fundamental constraints that must be considered for the robot to accomplish the task. This allows the navigation plan to be flexible and to be changed easily as the robot's perception of the world changes. The rough navigation plan is then passed to a mechanism that transforms the navigation plan into guidance for the robot's low level control loop in real-time.

A more detailed presentation of the traditional approach to solving navigation tasks, including a survey of techniques developed under the traditional paradigm, as well as some inherent limitations to the approach, follows. A similar discussion of the merits and shortcomings of the reactive approach to robot control is also presented. This is followed by a discussion of an alternative, situationally-driven approach, and a more detailed description of the LNav system.



Navigation Task: "Cross the bridge"

An LNav Navigation Plan

Figure 1.4

1.2. Traditional Approaches

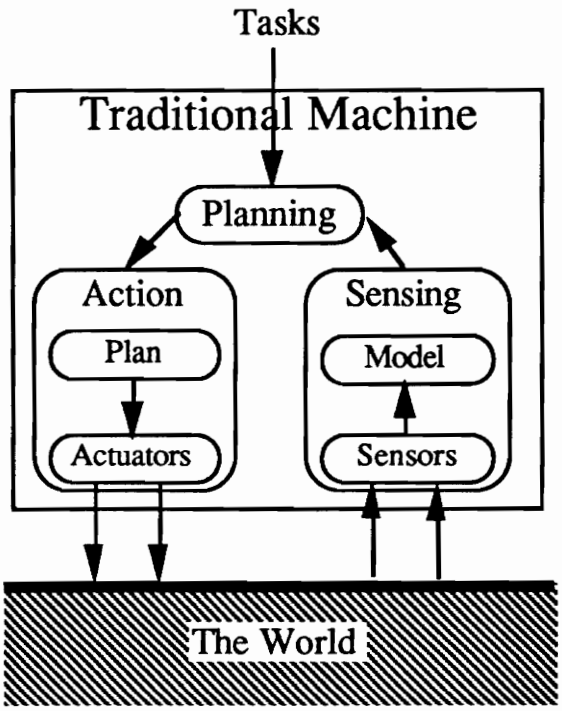
1.2.1. Definition

Broadly speaking, the "traditional approach" to robot control refers to the paradigm developed in the 1970s for automatically accomplishing goals {Fikes71, Sussman73, Sacerdoti77}. The traditional approach refers to a particular way in which information is assumed to flow through the system. The overall problem of mapping sensors to actuators is broken into three distinct and largely independent parts: sensing, planning, and acting (see Figure 1.5). The sensing portion transforms the sensor data into a meaningful internal representation of the world. The planning system accepts the tasks to be achieved and the model of the world generated by the sensing system, and creates a plan for accomplishing the current task. The action system takes the plan and uses it to generate actuator commands.

For local navigation systems employing the traditional view, the same flow of information and division of the problem into sensing, planning, and acting is used; but, the functionality of each component and the information passed between the components is specialized to facilitate navigation. The sensing system still transforms sensor data into a meaningful representation of the world's state. However, much more emphasis is placed on creating an internal representation of the world that encodes metric information about the world (e.g., the table is 2 x 3 meters and 3.2 meters away). The class of tasks on which the planning system operates involves moving between two places in space. The plans generated by the planning system also cater to the issues of local navigation, centered on the physical interaction between the robot and its local environment. The operation of the action system remains more or less the same, transforming the generated plan into actuator commands.

The heart of a local navigation system employing the traditional approach resides in the planning system, as the transformation from internal model of the world to a plan for achieving a navigation task occurs there. Local navigation planning is generally referred to as the path-finding (or path planning) problem. Path-finding is roughly defined as follows:

given a description of the world (provided by the sensing system) along with starting and ending positions, plan a path that will move the robot between those two positions. A path typically refers to a detailed sequence of actions which will move the robot between its starting and ending positions (e.g., "turn +12 degrees, move +1.2 meters, turn -33.3 degrees ..."). The path should account for all the environmental constraints, as modeled by the sensing system, as well as a number of standing objectives (e.g., minimize distance traveled, maximize visibility along the path, etc.).



Traditional robot control architecture as applied to local navigation

Figure 1.5

In addition to the architecture depicted in Figure 1.5, most traditional approaches to local navigation make two characteristic but flawed assumptions:

- Omniscient sensing: Reliance on the sensing system to be all seeing, able to sense without error as well as predict and model future states of the world.
- Omnipotent action: Reliance on the action systems to execute plans without error.

1.2.2. Approaches

A system expected to autonomously direct a robot through the world must produce either an explicit or implicit model of the world upon which to base its actions. In path-finding systems, such models are generally explicit and take many different forms, all of which are functions of the type of data returned by the sensors and/or the needs of the planner. One way to classify the different types of local models is to partition them based on the environment modeled. While there are a number of operating environments that have been assumed by path-finding systems, they can typically be divided into two general classes: structured and unstructured. Structured environments are generally described by a collection of geometric objects (e.g., a desk, a wall, or a car) located in Cartesian space. Unstructured environments are described more abstractly as a collection of sensor readings relating metric and/or qualitative information about the environment (e.g., the distance to the nearest obstacle ahead, or an elevation map of the local terrain). The primary difference between these two environment classes is one of perception or a lack thereof. Structured environments are structured because a meaningful symbolic representation of the local environment is created from the sensor data. Unstructured environments are so labeled because a symbolic language is either unavailable or not employed. For example, natural terrain is typically considered to be an unstructured environment simply because it is difficult to parse raw sensor data into symbolic representations of rocks, hills, and trees images of those objects. Thus, the distinction between structured and unstructured spaces is one of internal modeling and not an inherent aspect of the environment being modeled.

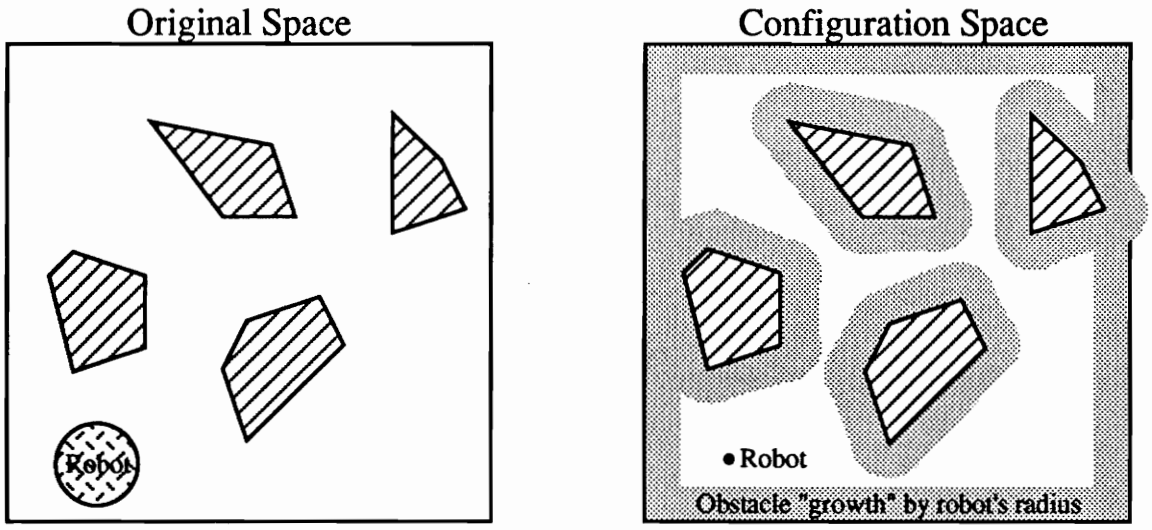
A characterization, with examples, of path-finding systems operating on each of these two types of world models (structured and unstructured), follows. To facilitate the discussion it is important to note that almost all path-finding systems, independent of the type of model on which they operate, transform the navigation task into one of searching through a weighted graph for the minimum cost path from a node representing the robot's current state to a node representing a goal state. Each link of the graph represents a particular action available to the robot. Thus, the resulting path plan corresponds to the links of the identified path through the graph. Three basic algorithms have been employed to search through the graph and identify a path (sequence of links) to accomplish the navigation task: Dijkstra's algorithm {Dijkstra59}, Nilsson's A* algorithm {Nilsson71}, and spreading activation {Charniak86}.

1.2.2.1.Path-Finding Through Structured models

Numerous path-finding systems assume that the sensing system parses sensor data directly into a structured representation of the world, usually in the form of geometric descriptors, such as polygons, polyhedra, and/or circles. The general approach is to transform the model of the world and the current navigation task into a graph search problem. Free space techniques create a graph by first identifying chunks of free space (i.e., the space not occupied by an obstacle) and then connecting them together to form a graph characterizing the connectivity of the identified free spaces {Brooks82, Nguyen84, Rueb87, Tournassoud88, Avnaim88}. Voronoi approaches, similar to free-space approaches, combine the identification of free space and the creation of the graph into one process {Shamos75, Kirkpatrick79, O'Rourke84}. Some approaches avoid creating an explicit graph but rather search through an implicit graph in the model itself. Iterative refinement is one such approach, posing a path and then recursivly modifying it to account for environmental constraints {Lamadrid86, Lamadrid87}.

Configuration Space (CSpace), possibly the most extensively researched of the structured representations, has been used as a spatial model by a number of path planning systems. The approach was introduced in 1970 {Rosen70}, expanded upon in 1977 {Udupa77}, extensively researched by Lozano-Perez {Lozano-Perez79, Lozano-Perez83}, and later extended in a number of ways to handle more complicated situations {Brooks83, Erdmann86, Laumond87}. The basic approach assumes a geometric model of the robot as well as a polygon description of all the objects present in the local operating space assuming operation in a two dimensional world. Each object (rigid body) is described by its configuration, characterizing the pertinent information about the object. For example, in two space, the configuration of a rigid solid is characterized by its x, y location and orientation in Cartesian space. Similarly, there is a configuration characterizing the robot. The way in which the configuration of the robot interacts with the configurations of the objects yields the CSpace. The result is a model encoding all the safe configurations (i.e., the robot does not intersect an obstacle) for the robot in the given operating space. The classic CSpace example involves a circular robot with a zero turn radius moving among static polygon obstacles, allowing the robot's orientation to be ignored because of symmetry. The CSpace, in this case, is obtained by "growing" the obstacles by the radius of the robot (see Figure 1.6). This simplifies the path-finding problem from that of finding

a path for a finite sized robot moving through Cartesian space to that of path-finding for a point moving through configuration space.



Constructing a configuration space model from a geometric model

Figure 1.6

Path planning systems operating on a CSpace model of the local environment find paths that move a point representing the initial configuration of the robot to a point representing the goal configuration. A primary feature of the CSpace technique is that for the purposes of creating a navigation plan the robot can be considered a point. The CSpace model transforms the general problem of finding a path for a finite robot with multiple degrees of freedom moving amongst obstacles into the problem of finding a path for a point through the CSpace. However, the simplification comes at a cost; one dimension is added to the CSpace for each degree of freedom of the robot. Finding a path through the CSpace amounts to connecting the initial and goal configurations by a series of valid configuration transitions. This is typically accomplished by transforming the general problem of finding a path through the CSpace to a graph search problem. The constructed graph is a Visibility Graph (or VGraph) {Lozano-Perez79, Moravec83}, and is created from a set of convex polygons by connecting each vertex in the space with those vertices that are "visible" from that given vertex. If both the start and goal configurations of the robot are also considered vertices during the construction of the VGraph, then the graph represents a set of possible paths between the start and goal configurations. A number of path-finding systems utilize

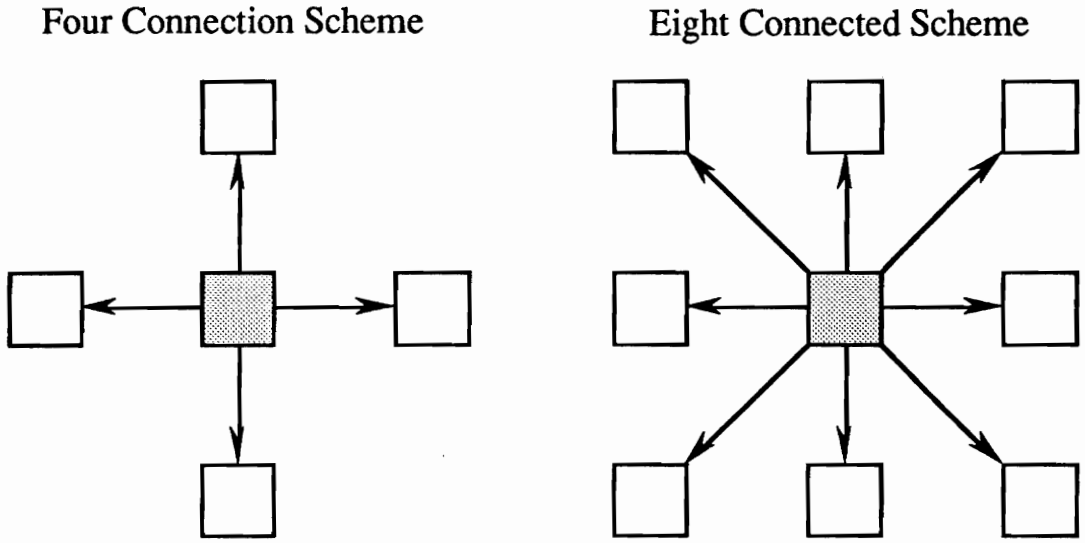
CSpace and VGraphs {Laumond83, Chatila85}. A major criticism of the CSpace-based approaches is the high dimensionality of the resulting CSpace when planning for robots with high degrees of freedom. As a result, researchers have tried various techniques to reduce the dimensionality of the CSpace {Lozano-Perez86, Faverjon87, Mazer87}. However, reductions in the dimensionality come at the cost of reduced functionality.

1.2.2.2. Path-Finding Through Unstructured models

Most models of unstructured environments are created from a regular tessellation of space into cells, with each cell mapping onto physical space. The most common of these is a rectilinear tessellation represented as a grid of square cells. Each cell in the grid has a physical mapping onto the world and contains characteristic information about that particular portion of the world. The type of information contained in the cells depends on the domain of operation. Systems operating in natural terrain generally fill such representations with elevation, slope, and texture information {Wilcox87, Marra88}, while systems operating in an indoor environment fill the cells with information mapping the presence of obstacles {Moravec85, Elfes87, Elfes89}. The basic grid has also been generalized to form hierarchical representations of space. One such generalization is the quadtree or its three-dimensional cousin the octree {Ayala85, Samet89, Samet90}. A quadtree is a recursive subdivision of space into its four constituent quadrants. Hierarchical representations allow the construction of models characterizing different portions of the space at different levels of resolution. For example, the portion of the environment immediately surrounding the robot could be modeled in great detail, while locations further removed might be modeled more coarsely {Moravec87}.

All of the models of unstructured space considered here are tessellations of space into squares or cubes, either uniform or hierarchical. Path-finding through such spaces can be seen as a search through a weighted graph, in which the graph is constructed by connecting each cell in the grid to neighboring cells {Thorpe84, Miller87, Slack87b}. This results in connecting each cell to either its four or eight adjacent neighbors (see Figure 1.7). The weight on the links connecting the cells is a function of the characteristics of information about the space modeled by the cell, the relationship between the two cells connected, and the abilities of the robot. By constructing an appropriate cost evaluation function, the paths found by searching through the graph of connected cells will implicitly account for environmental constraints (e.g., keeping the robot from running into things, accounting for

resource utilization, etc.). The resulting paths are generally subject to stair-stepping effects {Smith}, in which the path takes unnecessary jogs due to the discrete nature of the graph. This effect can be controlled by sophisticated search techniques {Slack88}, or by post-processing the path returned {Thorpe84}. Path-finding with hierarchical representations is a bit more complicated, yet similar in nature to the techniques used for simple grids {Fujimura86, Kambhampati86, Puri87}.



Construction of a planning graph from a grid-based spatial model.

Figure 1.7

1.2.3. Inadequacies

While there are many elegant planning techniques among the numerous approaches adopting the traditional approach to local navigation, there are three fundamental issues not fully addressed by this body of work. The first involves the ability of the sensing system to create models of the world. The second concerns the ability of the action system to carry out generated plans. The third is the ability to recover from interruptions in execution resulting from overly strong assumptions about the abilities of the sensing and/or action system(s). Each of these issues will be presented in turn and will include some discussion of alternative views that might provide ways to address these issues.

1.2.3.1. Sensible Sensing

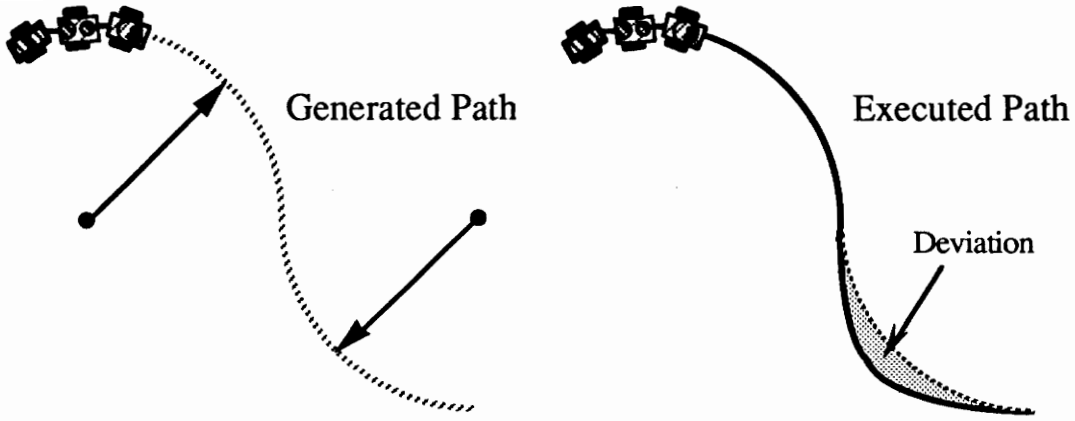
A fundamental flaw in most path-finding systems is the bold assumptions they make about the kinds and fidelity of representations that can be constructed using state of the art sensing systems. Systems which assume that the sensing system can "see" through walls, or construct arbitrarily accurate models of objects are in clear violation of possibility. Many systems make a similar but weaker assumption by relying on the sensing system to accurately map everything that is possibly visible {Oommen86, Rao86, Feng89}. The impact of such assumptions about the abilities of the sensing system is largely a function of how and when the information is used. If the information is only assumed to be accurate in the robot's immediate surroundings, and the control system cycles through the sense-plan-act cycle before the robot is moved beyond the range of its reliable sensor information, then these assumptions might be viable. However, planning systems are all too often constructed without acknowledging the fact that real sensors can provide only a partial and uncertain view of the world.

Even when a path-finding system does not make unrealistic assumptions about the abilities of the sensing system, the traditional view places a significant burden on the sensing system to provide the information required to satisfy the current goal. As defined by the traditional architecture, the sensing system is required to provide the planner with information in the absence of any notion of the current task. Thus, because there is no way for it to determine what is relevant and what is superfluous, the sensing system must provide as much information about the world as it is currently able to sense. A typical solution assumes that the necessary information will be in front of the robot, resulting in systems that can actively sense only the space in front of the robot . While such schemes supply the robot with the necessary information the majority of the time, they seriously limit the robot's functionality in many environments, as they render the robot incapable of moving in one direction while sensing in another (e.g., checking traffic while crossing a street). Another approach assumes that all of the local space is fully relevant and proceeds by actively sensing in all directions. This approach has the obvious disadvantage that much of the robot's computational resources are wasted processing information that is irrelevant to the current task. These problems become especially troublesome when high bandwidth sensors, such as cameras capable of producing reams of data, are employed.

One method of reducing the burden on the sensing system involves modifying the basic architecture to support the notion of task-directed sensing {Puri87, Firby89b}. Task-directed sensing involves the focusing of sensor activity with respect to the problem being solved. This can be accomplished by modifying the sensing system to accept guidance and by modifying the planning system to provide the sensing system with a focus of attention. This approach to sensing has proven to reduce substantially the workload of the sensing system without significant loss in the fidelity of the information {Anderson85, Burt88}. If, in addition to providing a focus of attention, the sensing system is also provided with a search image, then powerful template matching techniques can be employed by the sensing system to interpret the sensory information and further reduce the complexity of acquiring sensor information {Adelson84, Andress88, Kriegman88, Kriegman89}.

1.2.3.2. The Uncertainty of Acting

Typically, path-finding algorithms generate solutions in the form of a detailed path to be executed by the action system. Such paths are often described as a sequence of line segments or circular arcs that the robot's action system is supposed to follow {Gat90b}. However, due to limits on the accuracy with which the world can be modeled and limits on the robot's ability to execute a given path exactly, the manifested behavior will never match the robot's idealized picture of what will occur when the plans are transformed into actuator commands {MingWang88}. For example, consider the path depicted in the left of Figure 1.8 in which a path has been described as the concatenation of two circular arcs, one turning to the right and the other to the left. If the robot executing such a path requires a finite amount of time to make a transition from one articulation to the other and the robot is in continuous forward motion, then it is impossible for the robot to follow the path exactly (see right portion of Figure 1.8). One approach to dealing with this type of execution error is to require the generated paths to be kinematically feasible. For example, clothoid curves are kinematically feasible paths for robots with Ackerman steering mechanisms {Kanayama85}. However, even when generated paths are feasible to execute in principal, some disparity will remain between what is anticipated and what actually happens. Models are never exact; actuators are never perfect. In many cases this disparity can be ignored, but too often the disparity is ignored when it should not be.



Path execution is *never* exact.

Figure 1.8

One approach to handling this dilemma is to process the sense-plan-act cycle fast enough for accumulated uncertainties to remain at an acceptable level. However, this approach has not met with much success, since such approaches are typically limited by the computational complexity of the path-finding technique. Another approach is to increase the fidelity of both the model of the local environment and the model of the robot used during the planning process. However, for many planning systems, increasing the fidelity of the model is impossible or impractical. For example, in most systems employing grid-based search techniques, increasing the fidelity of the local model and/or the robot violates the assumption that the robot always fits within the space of a grid cell made by the planning system. While there are solutions to this problem {Slack87a}, they solve only part of the problem, and typically result in significant increases in the representational cost. Systems having no explicit bound on the resolution of the representation find that as the fidelity of the local model increases, the computational complexity of the planning process also increases. Due to the uncertain nature of the world, there is only a limited amount of time to respond to changes in the world; therefore, increasing the complexity involved in formulating a response should be avoided if at all possible.

All of these problems are a result of the fact that the plans generated are highly detailed, are expected to control the robot's actions for significant periods of time, and are disconnected from the robot's changing perception of the world. Another approach would be to create only a rough plan whose details became explicit only at execution time. This

allows the abilities of the robot to be reasoned about at a more abstract level, relying on the robustness of the plan to handle local uncertainties. This idea is typified by the numerous systems which use gradient fields as plans to be transformed into action. Such approaches avoid the problems of explicit paths because action is generated by controlling the robot's position based on the local value in the gradient field. A number of such planners employing this technique have been implemented and show encouraging results: potential fields {Andrews83, Krogh84, Khatib85}, path gradients {Arkin87, Kheradpir88, Payton88, Arkin89, Feng89}, and configuration gradients {Dorst88, Trovato89}.

1.2.3.3. Recovery

Path-finding systems that make strong assumptions about sensing and/or acting systems eventually find that their generated plans fail to accomplish the intended navigation task. This failure is due to the uncertainties in the world that cannot be accounted for adequately in the plan. This failure can occur with or without being detected. In the case that the failure cannot be detected, there is no way to recover in a graceful fashion. Detection of failure has led to a number of efforts involving the verification of execution. Execution verification can be accomplished by monitoring appropriate sensor activity in order to ensure that key features and states occur at the appropriate time and/or in the appropriate order. To some degree, detection has been addressed through execution monitoring techniques which place acceptability limits on sensor readings during the robot's traversal of a path {Gat89}. More sophisticated techniques schedule sensor activity to verify that the state of the world matches the assumptions made by the planner at the time that the plan was generated {Miller88}.

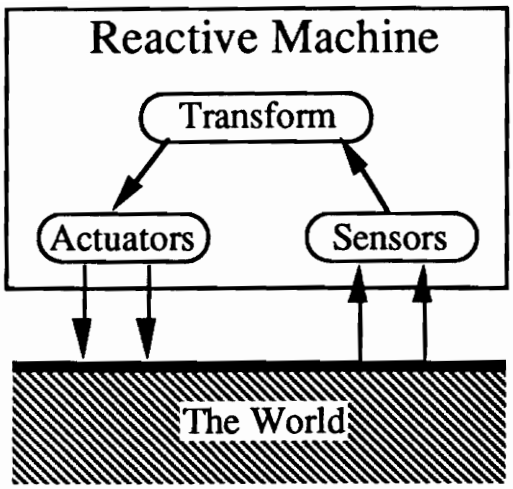
The preferred method for dealing with the world's uncertainties and the possibility of failure is to attempt to avoid these issues altogether. In planning literature this is referred to as "iterative planning", in which the plan is intended to drive the robot's actuators for only a short period of time, while an updated plan is being constructed. In this way the robot is always acting with the most recent information possible. In systems employing the traditional view, this approach has not met with much success as generation of a new plan incurs a significant computational expense. However, the idea of controlling a robot by quickly and repeatedly updating a robot's actions based on the world's current state has led to the creation of an alternative approach: the reactive approach.

1.3. Reactive Approaches

The primary difficulty with the traditional view of robot control for real-world operation lies in overcoming the intrinsic uncertainties associated with the physical environment. In general, it cannot be known if there is a trash can blocking the path, or if the car in front will shortly blow a tire, until such events have been confronted. However, when such events are recognized, there may hardly be time to do anything but "react" to the situation. This has led many researchers to examine the problem of controlling a robot's actions from a different point of view.

1.3.1. Definition

The traditional view takes a top-down approach and is rooted in the assumption that building an internal model of the problem is necessary in order to generate actions that are sensitive to the world state and the current goal. An alternative view, the reactive view, approaches the problem from the bottom-up, in which action is continuous; defined at any moment by both the current state of the robot and the currently available information about the world. In effect, the reactive view eliminates the sense-plan-act cycle, replacing it with a continuously computed function that transforms sensor readings directly into actuator commands (see Figure 1.9). The approach replaces explicit goals, such as "go home", by a transformation function encoding a number of implicit goals, such as "do not run into anything", "avoid dead ends", and/or "move north when possible". The actions of the machine, and thus the goal at any moment, are determined by the transformation function and the current sensor readings. The strength of the approach lies in the assumption that the transformation function can be quickly calculated. By continually recomputing the transformation, reactive machines avoid the need to reason explicitly about the world's uncertainties. Uncertainties are handled implicitly, because no action runs open-loop for more than the time it takes to compute the transformation function (typically much less than a second). Because the actions of a reaction machine are continuously being updated, such systems manifest the type of smooth real-time response to detectable changes in the state of the world that is desired in an autonomous robot.



The reactive approach to acting in the world

Figure 1.9

1.3.2. Approaches

A number of architectures exist which adopt the reactive view and generate actions directly from sensor readings. One common characteristic of these systems is that they occupy a fixed amount of memory (or silicon). Reactive systems fall into two basic categories: those that make little or no provision for the use and manipulation of internal state, and those that do.

Brooks and his students at MIT have made a significant impact on reactive systems through the use of his "Subsumption Architecture" {Brooks86a}. The approach creates reactive machines by building up layers of primitive goal-achieving behaviors formed from precompiled sensor to actuator transformations. The architecture is termed "Subsumption" because the higher level behaviors influence the input and output of the lower level behaviors in order to suppress the activity of more primitive behaviors and to manifest more complex reactions. Thus, for example, a reactive machine that moves forward and avoids running into things can be created from two behaviors: a low level behavior simply driving the robot's motors forward, and a subsuming behavior that continually reads a proximity sensor to determine if a collision is imminent. If the higher level behavior detects that a collision is imminent, then it inhibits the activity of the lower behavior causing the

robot to stop. If at some point the higher level behavior ceases to detect a possible collision, then its inhibition of the lower drive behavior stops and the robot continues to move forward. The robots created using this architecture typically maintain little or no state information, relying instead on information currently attainable from their sensors to determine the most appropriate action to take. This is facilitated by the assumption that the necessary state information is available from the world and that there are sensors capable of extracting that information upon demand. For example, such sensors might provide information detecting "the direction of the closest obstacle to the robot", or "how fast a wheel is currently turning". One key to the success of these systems lies in the fact that the type of goal-achieving behaviors of which these systems are capable matches well with the type of sensory information extractable from the world. For example, such goals might include "don't run into anything", or "maintain a velocity of one meter per second". The success of this approach can be measured by a number of real robots that have been created employing the Subsumption Architecture and have operated in realistic environments {Brooks86b, Connell87, Brooks88}.

More powerful machines have been created by augmenting the basic reactive system with a mechanism for using and manipulating internal state {Agre87, Kaelbling87, Gat90}. Such a mechanism can be integrated into the basic reactive approach if it is made transparent to the operation of the transformation function. In order to accomplish this, reading state information is handled by the transformation function in the same way as reading sensor data, and writing state information is handled in the same manner as controlling an actuator. By making state information appear as both an actuator and a sensor, these systems can form internal feedback loops. A critical part of remaining transparent to the transformation function is that the augmentation must not cause the transformation function undue delay. If the transformation function is slowed sufficiently, then the assumption that the world does not change significantly during the time it takes to compute the transformation becomes invalid and the entire paradigm falls apart. With the ability to use and manipulate state information, these machines are finite Turing machines; as such, they are able to encompass a larger class of implicit goals and operate in a wider range of environments than those unable to manipulate state information. For example, by keeping track of the number of times that a particular state has been detected without the intervention of some other particular state or set of states, a reactive system can be given the ability to escape local minima {Gat90}. Such an ability would be beyond those of a

reactive system which is unable to use or manipulate state information in this way. However, due to bounds on the amount of time and memory available to these machines, having the ability to use and manipulate state does not inherently allow them internally to construct arbitrarily complex models or to perform arbitrarily complex computations.

1.3.3. Inadequacies

The purely reactive view has no provision for explicit goals and, as a result, no mechanism for explicit plans for achieving goals. Goals in reactive systems are implicit; precompiled into the transformation function. This is not to say that the goals are not dynamically defined, but rather that they cannot be explicitly manipulated the way that traditional systems manipulate goals. Because the goals in reactive systems are specified prior to execution, there is serious doubt as to how complex this type of system may get while remaining within the limits of practicality. As the number of sensor feeds, the number of actuator outputs, and the number of internal goals increase, the complexity of building a system that appropriately conducts all of these interactions becomes confusing at best.

A further concern is that reactive systems provide no mechanism for explicitly reasoning about the way a current action will affect future opportunities. This lack of foresight in *purely* reactive mechanisms limits the complexity of the goals that can be attained as well as the types of failures that can be avoided. Hence, while this level of reactiveness is necessary, it is not sufficient for attaining higher level goals such as "go to Joe's office". Achieving high level goals requires the use of more abstract information about the environment, and a means of handling explicit goals.

1.4. Situational Approaches

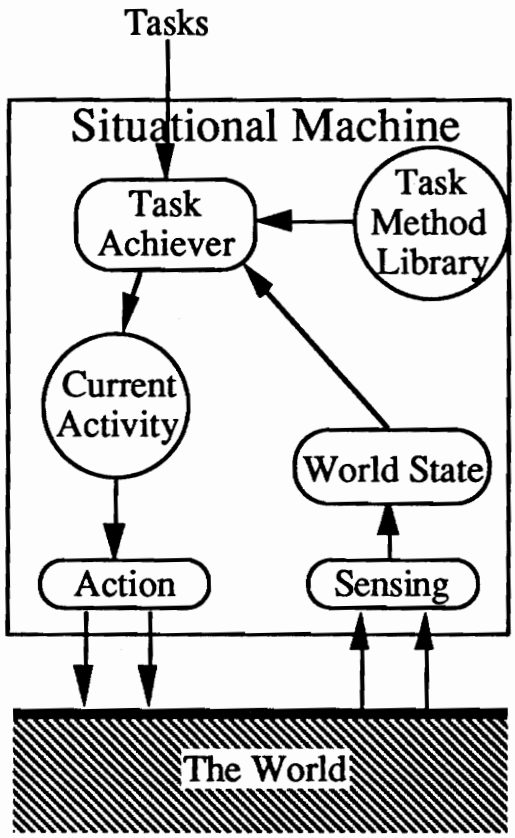
The insensitivity of traditional approaches to the world's uncertainties and the inability of reactive approaches to handle explicit goals has led to the creation of a third approach to controlling robots: the situational approach. There are many problems involved in operating in the world: the unexpected box, the unpredictable person, unreliable actuators, inaccurate sensors, etc. Each of these problems represents a different type of uncertainty

that a robot operating in the world must be able to handle. While uncertainty in the world comes in many forms, the world is not an entirely unpredictable place. On the contrary, the world is more predictable than not. Due to the laws of physics and the influence of social laws and conventions, the world changes in a more or less continuous fashion, causing situations to develop and flow into each other rather than changing discretely. Although it is not possible to know future states of the world, they can be bounded; typically unfolding within a known set of possibilities. Cars usually travel on the street, not on the sidewalk. People generally walk on the floor, not on the ceiling. Another observation in support of the world being a semi-predictable place is that in the routine of everyday tasks, a truly unique problem rarely arises {Agre88b}. Most of our day-to-day activity involves following well-known and well-established routines. This is not to say that we always do the exact same thing, but simply that once past childhood we rarely do anything entirely new; shopping at Vons in California is nearly indistinguishable from shopping at Krogers in Virginia. It is this semi-predictable nature of the world that allows systems employing a situational approach to make assumptions about the best way to interact with the world in order to accomplish tasks. Situational systems exploit the world's semi-predictability by characterizing the state of the world as it relates to accomplishing the current task and then accessing established routines for determining the best course of action.

1.4.1. Characterizing Situations

A *situation* is the combination of the current activity, the state of the world, and the current task to be accomplished. For example, walking on a bridge in and of itself does not refer to a situation but simply characterizes the current activity. For something to be labeled a situation, the activity must be related to a task. Walking across a bridge refers to both the current activity as well the task to be accomplished. Sometimes the current activity and the current task are incompatible. Leaning on the rail of a bridge and the task of crossing the bridge are incompatible, as rail leaning makes no perceptible progress toward the objective of crossing a bridge. This is where situational approaches fit in. Roughly speaking, situational approaches operate by continually trying to ensure that the current activity being performed by the robot is the best known activity for accomplishing the current task (i.e., situational systems are always trying to optimize the robot's current situation). Figure 1.10 shows the basic architecture of a situationally based machine. Sensing gathers information about the world and passes it to a world modeler which integrates it into its current model

of the world. The heart of the system is a module called the task achiever. The task achiever uses the world state information, the current task, and a library of methods for achieving tasks to continually maintain as the current activity the best "known" activity for achieving the current task. The current activity is used by an underlying action mechanism to generate actuator commands. It is important that, unlike the detailed plans of the traditional approach, the current activity is typically on the level of "proceed down *this* hall", or "move *that* way across the field". Because the current action makes only a rough commitment to a particular line of action the approach avoids the problems associated with the use of detailed plans.



Basic architecture of a situational machine

Figure 1.10

Although the situational approach to generating actions appears somewhat similar to the reactive approach, there are a number of significant differences between the two

methods of controlling robots. The clearest difference is the way in which goals are handled. In reactive systems the goals are implicit, having been compiled into the transformation function. Thus, reactive systems do not and cannot know the objective. In situational systems the goals are explicit, given to the system at execution time. This explicitness allows them to be more directed in their activities since knowledge of the objective is available to these systems. As an example of what is meant by a situational approach to solving a problem, consider the act of making purchases at a store. This is a generalized routine that most people can reliably perform in almost any store in which they choose to shop. However, planning the entire process before the actual event is pointless, since information about which cash registers are open, or which of the open registers has the shortest line is unavailable, and also irrelevant until you are ready to check out. Once ready, however, there are a number of routines that can be employed to facilitate the activity: choose the shortest line, place the items to be purchased on the counter, reply to any idle conversation, pay the cashier. Note that most of the information about what to do next has nothing to do with the type of store, the day of the week, or the objects being purchased. Situational approaches derive their strength from the fact that the aspect of the world that is critical at any moment is usually an isolated and separable facet of the larger picture. This allows them to identify and employ only the information and techniques that are relevant to the current situation. By limiting the information considered to only that which is deemed relevant to the situation, determination of an appropriate action is greatly simplified. Reactive Action Packages (RAPs) {Firby87, Firby89a}, Procedural Reasoning System (PRS) {Georgeff87}, and Universal Plans {Schoppers87} are a few of the more important contributions in this area.

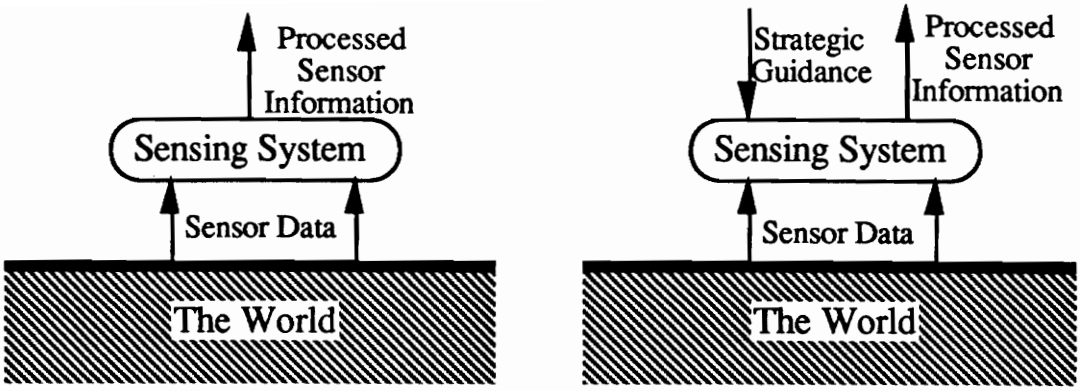
1.4.2. Task-Directed Activity

Any system which acts as a local navigator for a robot must comply with the inherent limitations on the amount of time available for recognition of and response to changes in the state of the world. There are two good ways of reducing the amount of computation involved in the local navigation process. One method is to reduce the amount of information flowing into the system by focusing the attention of the sensing system on the task being accomplished. For example, if the current navigation task is to move down a hallway, then there is no reason to actively sense an entire panorama when only the portion of space directly ahead is known to be relevant. The other method is to exploit established

routines so that the robot is able to respond quickly to changes in its perception of the world. For example, avoiding an obstacle on the floor of a hall can be accomplished with a precomputed routine once the situation has been recognized.

The key to an effective situationally-based system is its ability to characterize the current state of the world. One source of information used in characterizing the world's state is provided by a sensing system. Sensing systems for robots expected to operate in realistic worlds must handle, in a timely fashion, the vast amounts of sensory data available at any moment. The common approach assumes that the sensing system "simply" interprets all of the data available in the current sensory frame. Interpreting the contents of an arbitrary sensor reading, especially a camera image, is a complex problem since sensory noise, ambiguities, and the sheer volume of data serve to confound the problem. One benefit of the situational view is that it forms a basis for task-directed sensing, in which sensor activity and computational resources are directed in support of the current task {Firby89b}.

The notion of task-directed sensing is a relatively new approach to sensing the world, thus requiring a change from the traditional unidirectional flow of information through a sensing system (see Figure 1.11). In addition to the raw sensor data, the modified sensing system must accept information providing strategic guidance. Such strategic guidance would provide the sensing system with expectations about where to sense, what things might be present, and/or where certain objects might appear. Humans most likely use some sort of sensory expectations to drive the interpretation of the sensory information that they receive. Psychologists refer to this process as "perceptual filtering". Consider waking up at a friend's house when you expected to be in your own bed. When you open your eyes expecting to see the ceiling in your own bedroom and instead see something else, a dichotomy arises. In this situation, the raw sensor data is not matching your visual expectations. Having realized that an anomalous situation has occurred, the difficulty is quickly resolved as you remember the way the evening before had ended. However, for that brief period of time while the dichotomy existed, interpreting sensory data was more difficult. A number of researchers working in the area of sensing are now embracing the notion of task-directed sensing to help guide the activities of the sensing system {Anderson85, Andress88, Burt88}. Giving a sensory system a focus of attention greatly reduces the general perception problem to one that can be more realistically handled in the time scales required for response to the world's uncertainties.



Flow of information through a sensing system; traditional and task-directed

Figure 1.11

Once the sensor information necessary to accomplish the task has been acquired there is still the problem of directing the robot's actions in service of the current navigation task and in light of the currently perceived situation. Central to the concept of situated action is the fact that the pathways through spaces encountered in everyday activity can for the most part be known, once the proper contextual frame has been established. Consider walking down a suitably wide hallway in which another person happens to be walking in the opposite direction toward you. In this situation, if you move closer to the right wall, the other person will usually move to their right and you will pass each other with plenty of room to spare. By employing a previously known social convention, the problem of knowing the exact space-time trajectory of the other person is avoided. Rather than carefully planning out the details of the robot's interactions with the environment, it suffices to categorize the situation and select a plan for behavior that is appropriate for the perceived situation. This is the principle behind the design of the navigation system LNav.

1.5. LNav: A Local Navigator

LNav (for Local Navigator) directs the actions of a mobile robot in service of local navigation tasks. LNav is designed with the following in mind:

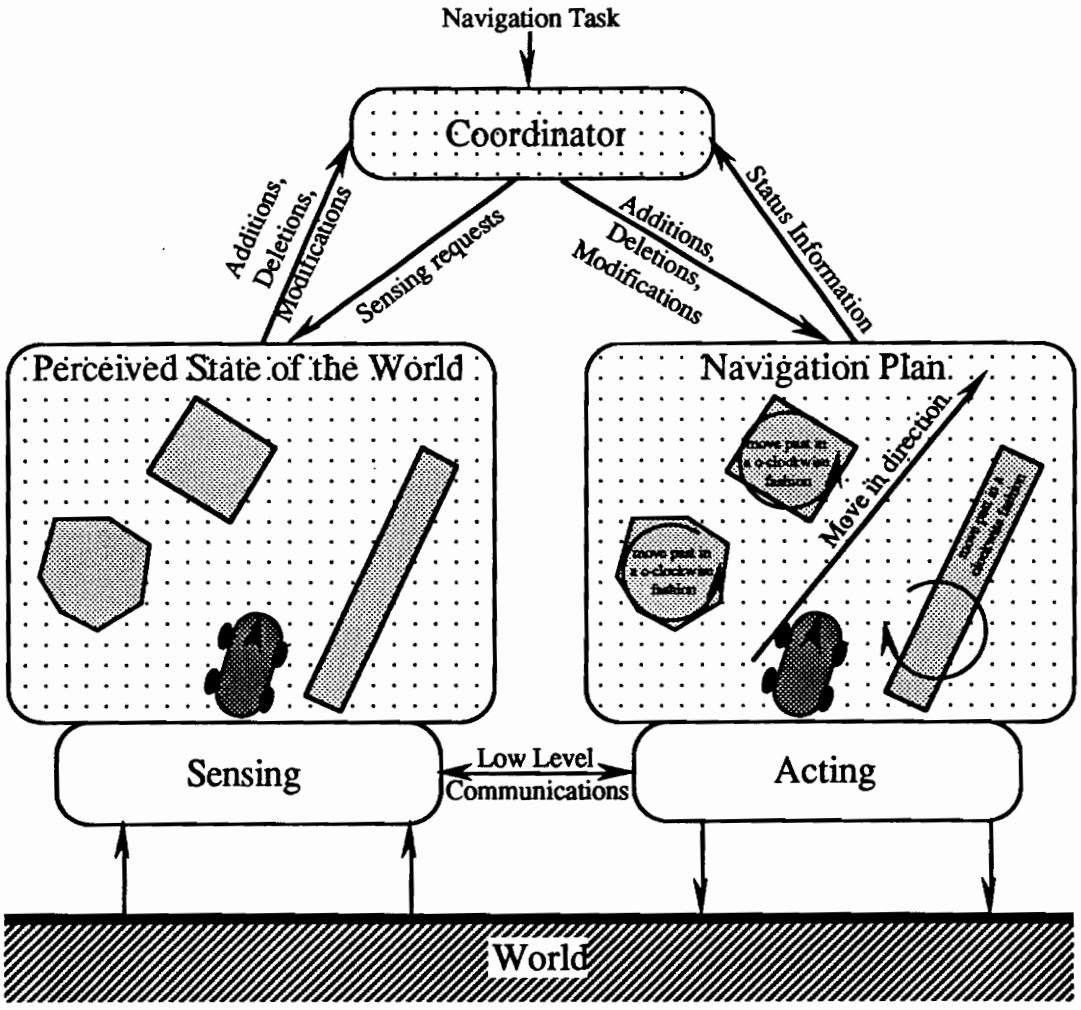
- The belief that in order for a mobile robot to carry out "high level" navigation tasks in a goal-directed fashion it must have a plan for accomplishing such tasks.

- The fact that, due to uncertainty in the world, any navigation plan which directs the robot's actions must be well connected to the robot's perception of the world.

LNav is composed of three modules. The first module, the NaT Manager, is responsible for maintaining the robot's perception of the world. The second module, the Feature Manager, is responsible for maintaining the robot's current navigation plan. The third module, the Coordinator, coordinates the activities of the other two modules, thus providing the thread which binds the navigation plan to the robot's perception of the world. The Coordinator accepts navigation tasks (e.g., "go *that* way" or "get to *that* spot") from the "high level" system and directs the activities of the other two modules in support of the current task. LNav employs Navigation Templates (or NaTs) as the building blocks used to construct navigation plans. NaT-based navigation plans are rough plans that are formed by simply characterizing the way the robot should move past a particular object in the world while trying to achieve the current navigation task. This allows NaT-based navigation plans to remain highly flexible. As objects in the world come into view, leave from view, change their perceived location in the world, or otherwise change their relationship with respect to the navigation task, the NaT-based navigation plan can be revised without interrupting the actions of the robot. To ensure that the sensing system also acts in service of the navigation task, the Coordinator directs the sensing system's actions by making sensing requests. The information gathered by the sensing system is used to update the robot's perception of the world, this in turn results in the Coordinator being notified of the change; thus, providing it with an opportunity to modify the navigation plan to accommodate the change. In this way the Coordinator keeps its current navigation plan connected to the robot's perception of the world (see Figure 1.12).

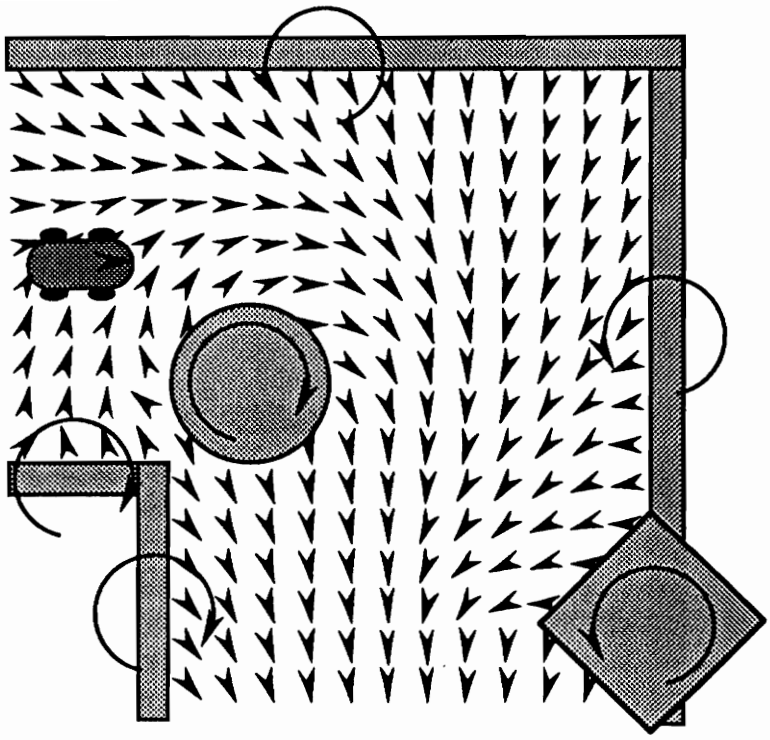
Due to the real-time nature of the world, this approach assumes the existence of a low level control loop which connects the sensing and action systems of the robot in much the same way as a reaction machine transforms sensor readings to actuator commands. Thus, from LNav's perspective, the robot's low level control loop is constantly trying to satisfy an implicit set of goals, such as "move ahead", "do not run into anything", or "stay

upright". In order to allow the robot to accomplish higher level goals, such as "go up this hall" or "do not go behind the desk", LNav provides the control loop with guidance in the form of a preferred direction of travel at any given position. This information is generated by a transformation function that operates on the current NaT-based navigation plan to provide the preferred direction of travel from any given position. Figure 1.13 shows the results of transforming a NaT-based navigation plan into directional guidance at a number of positions. The low level control loop uses this information to move the robot in service of a specific navigation task without having to know of the task's existence.



Coordinating sensing and acting

Figure 1.12

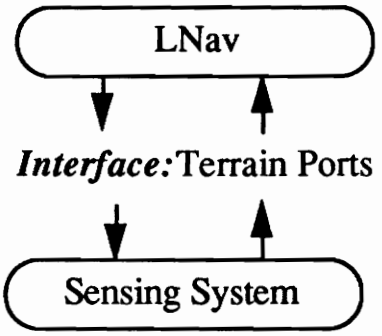


A NaT-based navigation plan and the resulting flow field

Figure 1.13

For purposes of the LNav system discussed herein, a sensing system that provides an incomplete and unstructured view of the world is assumed. The interface between LNav and the sensing system is a structure called a *terrain port* (see Figure 1.14). A terrain port is a window used by LNav to look at the view of the world provided by the sensing system. Terrain ports have a physical shape and are associated with a physical location in the local environment. Thus, when LNav requires information about some surface area in the local environment, it opens a terrain port covering that area, directs the sensing system to acquire information for that region, and then extracts information through the port. Because a structured representation of the world is required for constructing a NaT-based navigation plan and the sensing system provides only an unstructured view of the world, LNav uses the information provided by the sensing system to extract features from the information and construct a structured model of the robot's local environment. This

process is accomplished through the use of filters which transform the information available from the sensing system via a terrain port into a symbolic format.



Connecting LNav to a sensing system

Figure 1.14

Navigation Templates and the transformation function used to provide the robot's low level control routines with guidance are developed herein. The interface between LNav and the sensing system is also developed. Finally, an LNav Coordinator is presented which employs NaTs and terrain ports to demonstrate that, by connecting NaT-based navigation plans to the robot's perception of the world, the plan is able to absorb many of the world's uncertainties, thus greatly simplifying the complexity associated with the construction and maintenance of navigation plans.

1.6. Outline

LNav is a local navigator which guides a robot's actions in service of a stream of local navigation tasks generated by a "high level" system. For the purposes of discussion the existence of a directable sensing system which provides an unstructured representation of the world is assumed. LNav directs the sensing system's activity to build a feature-based representation of the local environment and to ensure that the information required for LNav to accomplish the current navigation task is available. The feature-based model of the world is used to construct and maintain a rough navigation plan built from Navigation Templates. The resulting NaT-based navigation plan is highly flexible and allows LNav to

quickly modify the plan in order to keep it up to date with the robot's current perception of the world. The navigation plan is quickly transformed into the guidance provided by LNav to the robot's low level control loop. The remainder of this dissertation presents the details of the LNav system.

Chapter 2 introduces Navigation Templates (or NaTs), describing their structure and the way they can be used to form navigation plans. The technique used to transform a NaT-based navigation plan into real-time guidance for the robot's low level control loop is also presented.

Chapter 3 extends the concepts developed in chapter one, extending the basic conception of NaTs allowing them to be used in a wider variety of situations. Also discussed is the way NaTs and the combining function address a robot's need for moment-by-moment direction.

Chapter 4 presents the basic data structures and information flow strategy used by LNav to direct the sensing system in support of the current navigation task. These structures also provide the basis for building a symbolic representation of the local environment.

Chapter 5 describes an implementation of the LNav system which employs the work presented in the prior chapters to direct the actions of a robot in service of a local navigation task.

Chapter 6 presents a summary, comments on the limitations of the approaches presented in this dissertation, outlines a number of extensions to the the presented work, and makes some concluding remarks.

Chapter Two

2. Navigation Templates I

The world is in a state of constant change and full of uncertainties. For robots to autonomously carry out navigation tasks involving their interaction with the world they must have a plan for accomplishing said tasks if they are to be carried out in a goal-directed fashion. However, due to the world's uncertainties, maintenance of a navigation plan cannot involve arbitrarily complex reasoning processes, because the robot's perceived situation will be in constant flux. Modifications to a navigation plan must be made quickly if they are to be of any use. Useful navigation plans must be able to provide the robot with immediate and continuous guidance as well as be highly flexible in order to avoid operating with invalid plans or having to replan each time the robot's perception of the world changes. With few exceptions, previous works in local navigation have attempted to control a robot's actions either by creating detailed navigation plans requiring substantial amounts of computation, or through techniques that create no plan at all, thus lacking direction. Some humans seem to approach the problem from a different direction; they do not worry about explicitly handling the world's uncertainties as they move through their daily life, but rather rely on their ability to negotiate most situations by mapping the current navigation situation onto one for which a known solution exists. For example, walking down a hallway is, for the most part, no different from walking down an aisle in a library. Both involve making a traverse along a corridor bounded by impassable obstacles on either side.

Navigation Templates are primitive building blocks for constructing highly flexible navigation plans which capture the essence of the navigation situation (i.e., the task and relevant environmental constraints). There are two types of Navigation Templates: those which are used to characterize the basic local navigation task being pursued, and those used to model known environmental constraints and characterize the relationship of the

constraints to the navigation task. Once a navigation plan has been built from a set of Navigation Templates, a powerful heuristic is employed to isolate the currently critical aspects of the plan and quickly (a few times a second) generate guidance for the robot's low level control system. As time passes and/or the robot moves through the world, causing changes in the robot's perception of the world, the navigation plan must be incrementally updated in order to remain useful. The fact that Navigation Templates do not depend upon one another allows them to be quickly translated, rotated, scaled, inserted and/or deleted from the navigation plan without affecting the other templates. Thus, a Navigation-Template- based navigation plan can be quickly modified in order to remain synchronized with the robot's perception of the world.

2.1. Motivation

The general problem of navigation involves reasoning about both global and local issues. At the level of global navigation, a navigation system can reason about the interactions between the robot and the environment at an abstract level; free from the problems associated with the physical interaction between the robot and the world. In contrast, at the local level a navigation system must be intimately connected to the physical world and reason about the robot's physical relationship to the objects in the world as those objects relate to the current navigation task. Making this separation allows the global navigation system to ignore the problems involved with the robot's immediate interactions with the environment and the way unforeseen obstacles affect the robot's actions. The separation also allows the local navigation system to operate without any knowledge of the overall objective and to focus solely on the immediate problems of sensing the world and maneuvering through space in service of the current navigation task. A navigation system adopting this paradigm accomplishes global navigation tasks such as "go to Joe Random's office" by reasoning abstractly about the way to accomplish the task, and then constructing a sequence of local navigation tasks such as: "Proceed up *this* hall", "Turn right at *that* intersection", and "Enter *that* doorway". Only the most rudimentary real local navigation information is contained in the description of the local navigation tasks, setting only the basic parameters of the problem. In addition, there are all of the implied environmental

constraints (e.g., a hall has two walls) as well as those constraints that cannot be foreseen until they are encountered (e.g., a box on the right side of the hallway).

A robot which is to act in a smooth and goal-directed fashion must maintain a navigation plan that remains consistent with the robot's perceived state of the world, continually providing the robot with an up-to-date indication of what action to take in order to make progress on the current local navigation task. Most work in robot navigation has focused on the generation of detailed local navigation plans, characterizing well defined paths through space for the robot to follow (e.g., "proceed forward 1.3 meters, turn 4.5 degrees to the right, then move forward 3.2 meters"). As was noted earlier, such plans are not well connected to the robot's perception of the physical world and therefore lack robustness in the face of the world's uncertainties and/or the robot's inability to follow the specified path exactly. As a result, if the navigation plan is to be kept current, plans must be quickly and repeatedly computed to provide for the robot's continual requirement for an immediate action which will make further progress on the navigation task. However, such approaches typically require too much computation to be of any practical use for local navigation.

In lieu of generating detailed plans, many local navigation systems have opted for schemes that maintain a representation of the navigation situation and employ a technique which quickly and repeatedly extracts tactical information from the representation to guide the robot's actions. This approach is typified by many of the gradient field approaches. The gradient field approach supports the robot's need for guidance by providing an indication, at any given position, of the direction of motion that best serves the current navigation task. While there are a number of approaches used to define gradient fields, only those that provide local determination of the gradient are capable of keeping up to date with the robot's perception of the world. The others {Dorst88, Trovato89} generally require a large amount of search to determine the gradient's direction at a given position, and consequently cannot keep pace with the local changes in the robot's perception of the world.

Potential field techniques were the first to be explored as a way of generating gradient fields for local navigation {Andrews83, Krogh84, Khatib85, Khosla88}. This technique starts with a geometric representation of the local environment (typically a set of polygons or polyhedra) and a goal location. The gradient can be thought of as directional forces

assigned to all locations in space. The force at any particular position is a function of an attractive force associated with the goal and the repulsive force associated with the obstacles in the space. The attractive force of the goal causes the robot to move towards the goal, while the repulsive force associated with obstacles causes the robot to maintain a safe distance from those obstacles. One major attraction of this approach is that it can be implemented as a direct transformation of the current sensor readings (ranging sensors) to actuator commands, thus keeping the plan very much in tune with the perceived state of the world. This approach has been taken in the building of many reactive machines. A serious drawback with this approach is that local minima are often generated, resulting in dead-ends in which the robot can become trapped. A number of techniques for alleviating this problem have been implemented. One solution employs a more conventional path-finding algorithm which first finds an explicit path avoiding local minima and sets up a sequence of way-points or subgoals. The way-points are then used to construct a sequence of potential field gradients, one for each way-point {Krogh86}. The result is a plan insensitive to moderate amounts of error in its execution and which causes the robot to avoid local minima. However, the generation of the way-points is computationally expensive and causes the system to be unresponsive both to larger changes in the environment as well as to changes in the current navigation task. Another solution to the local minima problem works in the opposite direction. Starting with a potential field, the technique employs a search through that potential field in order to find a path that avoids the local minima. This approach results in the generation of a sequence of discrete motions (e.g., "turn -12 degrees, proceed +3.22 meters, etc."), and can take a considerable amount of time to generate. This approach is also computationally expensive, making it inappropriate for local navigation.

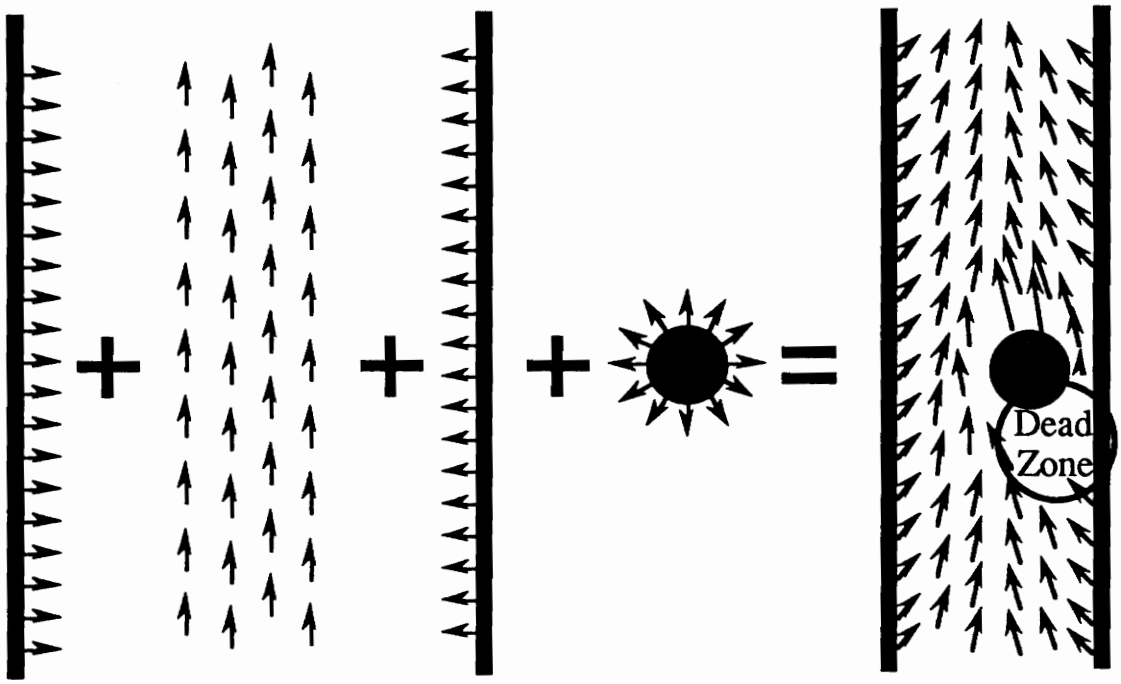
Clearly absent from these gradient-based approaches is the use of situational knowledge to direct the creation of the gradient field. Recent work in robotics has begun to point to the situational information that resides between the environmental constraints and the current task as a way to cope with the complexity of interacting with the world {Georgeff87, Agre88a, Firby89}. The benefits of situational knowledge in local navigation result from an analysis of the interaction between the navigation task and the identified environmental constraints. This analysis produces knowledge that aids in determining the way to respond to the current situation. For example, simply knowing that the navigation task is to walk along a sidewalk provides enough information to place

bounds on what is relevant (e.g., cars parked along the curb can be ignored as they play no role in walking along a sidewalk). Some research has started to apply situational knowledge to the generation of gradient fields in order to address the local navigation more directly. Arkin introduces the notion of a motor schema which assigns a gradient field to identifiable environmental constraints in a way that depends on the navigation task {Arkin87, Arkin89}. The resulting gradient at any position is obtained in the same way as that arrived at by the potential field approaches: by summing the effects of the various gradient fields at the specified position.

All of these local gradient summation approaches suffer from a common problem of generating local minima. This generally occurs when the effects of the various gradients being summed cancel out one another, resulting in dead spots where the resulting gradient field provides no guidance. In the situation depicted in Figure 2.1, for example, the resultant gradient is created by summing the effects of four component gradients: one used to define the navigation task of moving up the hallway, another used to represent an obstacle in the hall, and two used to represent the walls of the hallway. The gradient field resulting when the effects of the four gradients are summed together (see the right portion of Figure 2.1) results in a dead spot in front of the obstacle and next to the wall. While a robot using the resulting gradient field to guide its actions would usually not encounter this particular local minimum, if the obstacle had not been noticed far enough in advance or temporary effects had forced the robot to the right of the hall in front of the obstacle, then the gradient field would no longer provide useful local navigation information.

A fundamental flaw of gradient summation techniques is that while they may in part be driven by the situation, they do not take full advantage of the information that can easily be extracted by a simple characterization of the situation. As a result, the tactical guidance that they provide does not sufficiently account for the relationship between environmental constraints and the local navigation task needed for robust control of a mobile robot. However, if an analysis of the situation is carried out, more sophisticated heuristics can be employed to overcome these limitations. For example, the problem depicted in Figure 2.1 might be overcome if the true navigation task were characterized not simply as proceeding up the hall but rather as proceeding up the hall passing the obstacle to its left. Creation of navigation plans encoding more situational knowledge and the incremental transformation of the resulting navigation plan into tactical guidance is the topic of the remainder of this

chapter. Navigation Templates provide a way of describing navigation plans which allow such situational knowledge to be encoded in the resulting plan.



Problem with gradient summation techniques

Figure 2.1

2.2. NaTs: Describing Pathways Through Space

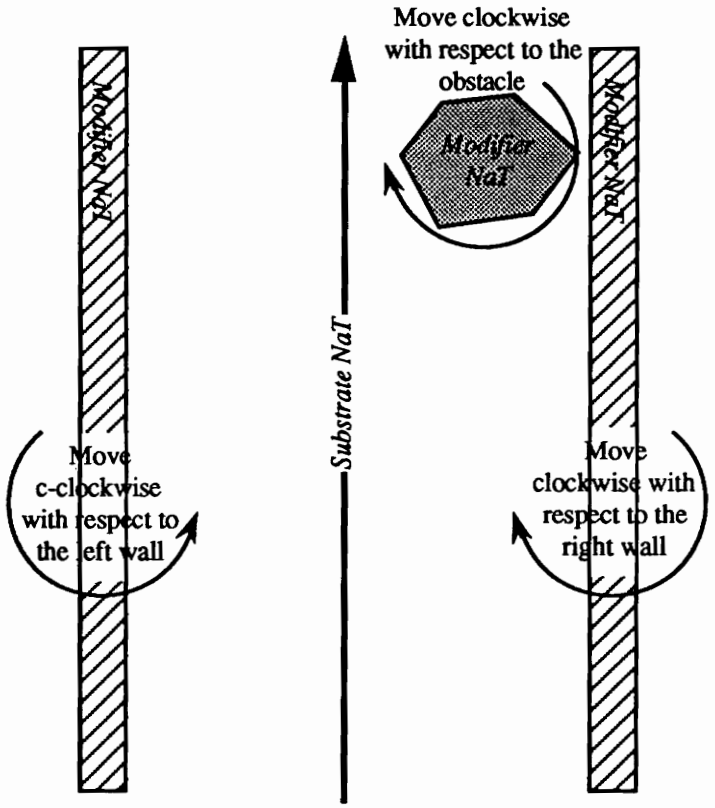
This section presents the primitives used to build navigation plans characterizing navigation situations. The technique rests on the assumption that it is possible to maintain an explicit rough navigation plan. For example, consider traveling through a building, in which the navigation task is simply to proceed up a hallway, with the right and left walls as the relevant environmental constraints. An important part of constructing a navigation plan is to characterize the relationship between the known environmental constraints and the navigation task. Hence, in the example, part of the navigation situation is the relationship between each wall and the navigation task of traversing the hall. The right wall's

relationship indicates that the robot should not get too close to the wall and that the desired direction of travel with respect to the wall is clockwise. Similarly, the left wall should not be approached too closely and the intended direction of travel with respect to the left wall is counter-clockwise. Now consider the way the current characterization of the navigation situation must change when the perceived state of the world changes. Assume that while proceeding up the hallway the robot's sensing system identifies an obstacle along the right side of the hallway. This new environmental constraint must be quickly incorporated into its navigation plan if the robot is to continue to make progress on the current navigation task. In order to incorporate the new obstacle into the current navigation plan, its relationship to the navigation task of traversing the hallway must be characterized. Given that the situation is occurring in a hallway, a useful rule of thumb states that "static obstacles to the left of the center of the hallway should be passed in a counter-clockwise fashion and those to the right in a clockwise fashion", where the direction is determined with respect to the particular obstacle. Thus, the new navigation plan is: "Proceed up the hallway, moving clockwise with respect to the right wall and the identified obstacle and counter-clockwise with respect to the left wall". A navigation plan capturing such situational information can employ powerful heuristics to quickly determine the most appropriate direction of travel.

Navigation Templates (or NaTs) are introduced as primitives used to create and maintain navigation plans. NaTs come in two forms: substrates and modifiers. Substrate NaTs are used to characterize the essence of the navigation task (e.g., "go up the hall", "go over there"). Modifier NaTs are used to model the environmental constraints and their relationship to the navigation task (e.g., "go left around the box", "avoid the car by going to the right of it"). Returning to the hallway example, a navigation plan can be built using NaTs as follows: associate a substrate NaT with the task of moving up the hall and associate a modifier NaT with each of the three obstacles (i.e., two walls and an object), annotated with the information indicating the intended direction of travel with respect to the obstacles (see Figure 2.2).

Once a navigation plan has been built from a collection of NaTs, a combining function looks at the characteristics of the individual NaTs to determine a direction of travel at any given position . If followed, the direction of travel will move the robot closer to accomplishing the navigation task, while at the same time accommodating known environmental constraints. Because the combining function can be run quickly (i.e., a few

times a second), it can be used as a resource by a low level vehicle control routine which is responsible for driving the robot's actuators. While this appears similar to the gradient summation-based approaches, there are some important differences. The internal representations of the obstacles in the environment have been annotated with navigation information relating them to the navigation task. This annotation allows the combining function to use more powerful heuristics when determining the direction of the gradient field at any given position.



Example of a NaT-based navigation plan

Figure 2.2

2.2.1. Substrate Navigation Templates

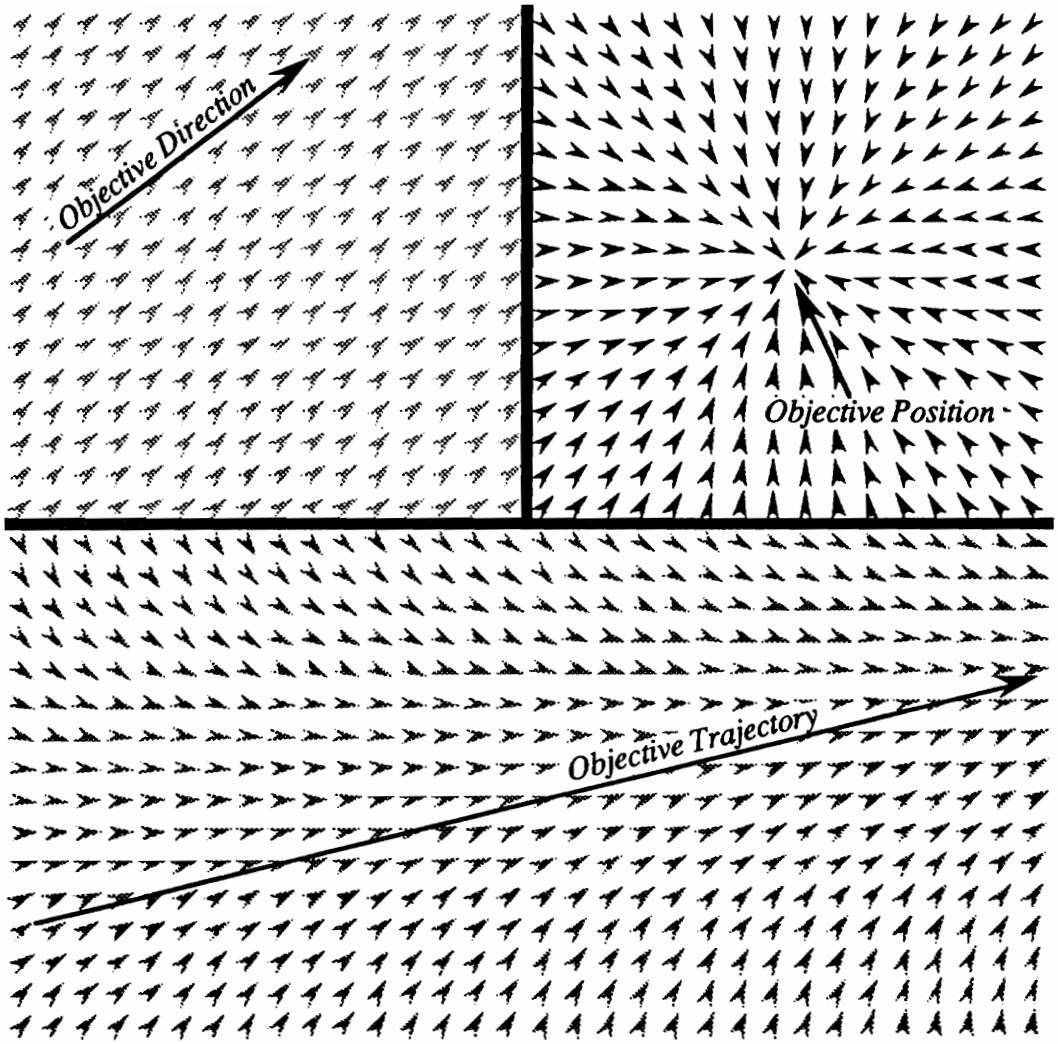
Substrate Navigation Templates (or s-NaTs) describe a particular navigation task as well as provide a basic scheme for accomplishing the task. Each s-NaT defines a gradient

field indicating at every position in space the direction of travel that best serves the navigation task being represented. For example, if the current navigation task is to move up a hill, then the gradient would, at every position, be directed to the top of the hill. It is important to note that the substrate is defined independently of the environmental constraints which limit the robot's ability to follow the substrate's gradient field. As a result, the gradient field of an s-NaT is just as likely to direct the robot through a wall as it is to direct the robot to avoid walls. While it is possible to create an s-NaT with almost any gradient field, this work has identified three types of s-NaTs which can be used to characterize a large class of navigation tasks: direction s-NaTs, position s-NaTs, and trajectory s-NaTs.

The simplest of the s-NaTs discussed herein are the direction and position s-NaTs. A direction s-NaT is used to describe navigation tasks such as "head west", "move across the room" , or "walk through the field", in which lateral deviations with respect to the direction are permitted. The gradient field for a direction s-NaT is trivially defined at every location to point in the *objective direction* (see Figure 2.3). To describe tasks such as "go over to the car", or "move to the rock", a position s-NaT is used. The gradient field for a position s-NaT is defined at every position to point toward the *objective position* (see Figure 2.3). The final type of s-NaT is a trajectory s-NaT, where the gradient converges to an *objective trajectory* through space. This s-NaT is useful for characterizing navigation tasks such as: "proceed up this hall", "follow this path", or "drive in this lane " (see Figure 2.3).

2.2.2. Modifier Navigation Templates

S-NaTs provide a basic scheme for accomplishing a navigation task in the form of a gradient field. Because the world is full of obstacles that place restrictions on the robot's ability to move through the world, the s-NaT's basic scheme for accomplishing a particular navigation task will need revision in order to accommodate environmental constraints. Modifier navigation templates (or m-NaTs) are used to model environmental constraints as they relate to the current navigation plan. For example, a trajectory s-NaT can be used to provide a basic scheme for moving up a hallway, while m-NaTs would be used to model the relationship between relevant environmental constraints, such as the walls or boxes, and the navigation task. Together the s-NaT and the m-NaTs are used to construct a navigation plan.



Objectives and gradients of direction, position, and trajectory s-NaTs

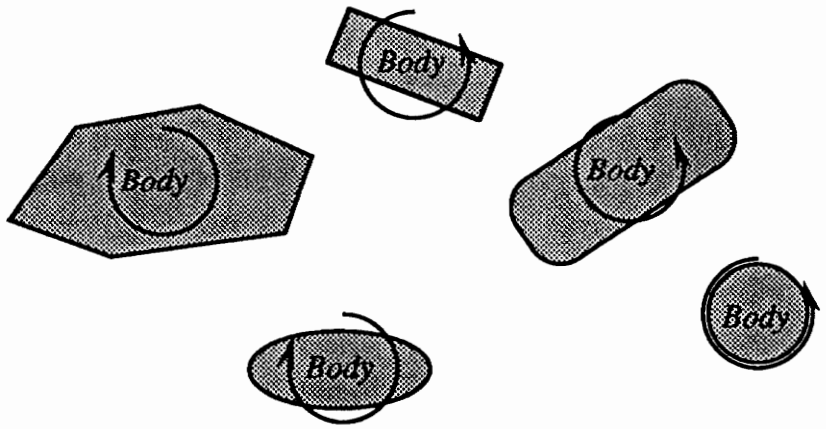
Figure 2.3

M-NaTs are used to model environmental constraints and provide information needed by the combining function to determine the preferred direction of travel at a given location. An m-NaT can be created for any convex geometric object[1]. For example, m-NaTs can be

[1]The restriction results from a number of geometric properties assumed about an m-NaT's body by the combining function.

created for circles, convex polygons, or half circles. M-NaTs can also be created for the two degenerate convex geometric objects: the point and the line. The geometric object for which the m-NaT is being created is referred to as the m-NaT's body (see Figure 2.4).

Central to building a navigation plan is the notion that there are two ways to move around an obstacle: clockwise and counter-clockwise. Thus, each of the m-NaTs has an associated spin, indicating in which direction around the obstacle the navigation plan dictates the robot proceed. Determining the spin of an m-NaT is typically accomplished through a simple analysis of the way that the obstacle relates to the robot and the current s-NaT. For example, in creating a navigation plan for moving up a hallway, the m-NaT representing the right wall is assigned a clockwise spin and the m-NaT representing the left wall is assigned a counter-clockwise spin. Roughly speaking, the way in which the combining function determines how an m-NaT affects the resulting gradient at a particular point is a function of its location in space, the m-NaT's spin, the direction of the underlying s-NaT's gradient at that position, and the location and characteristics of the other m-NaTs in the navigation plan.



Possible m-NaT bodies

Figure 2.4

2.3. Combining NaTs

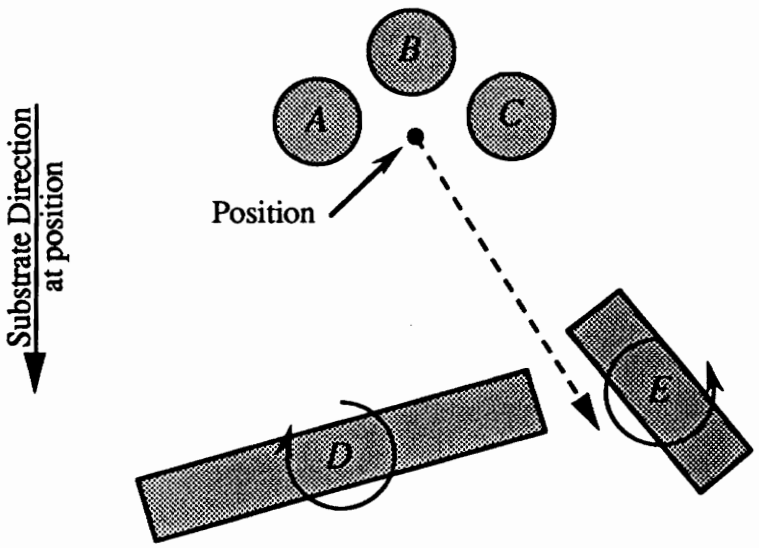
S-NaTs are used to characterize a navigation task and to provide a basic scheme for accomplishing that task (i.e., its gradient field). M-NaTs are used to model environmental constraints as well as their relationship to the navigation task (i.e., their spin). Together an s-NaT and a number of m-NaTs are used to construct a rough navigation plan for accomplishing a given navigation task. This section presents the NaT combining function which transforms a rough NaT-based navigation plan into a gradient field[2]. The resulting gradient is intended to serve as run-time guidance for the robot's low level control loop. To determine the direction of the gradient at a given location, a two-step process is employed. The first step of the combining function isolates the immediate navigation objective. For example, the current navigation task might be to traverse a hallway and avoid the box on the right side of the hall; however, the immediate navigation objective is to move between the left wall and the box. The second step of the combining function uses the identified immediate navigation objective as well as more local constraints to determine the direction of the gradient at the given position. For example, while the immediate navigation objective might be to move between the box and the left wall, the robot might be located close to the right wall, in which case the gradient should flow away from the wall before proceeding between the box and the left wall. However, it is important that progress towards the current navigation task is not lost while moving away from the wall; therefore, even the motion away from the wall should serve the navigation task. The remainder of this section is broken into three parts. The first presents the process by which the immediate navigation objective is identified. This is followed by a presentation of the process used to determine the direction of the gradient. Finally, some comments and observations on the entire combining process are given.

---

[2]The term gradient field is used loosely here, as the combining function does not produce a gradient field in the strictest sense of the word, but rather the combining function returns a preferred direction of travel at a given point. However, in many situations it is clearer to refer to the output of the combining function as a gradient field.

2.3.1. The Immediate Navigation Objective

Consider the situation depicted in Figure 2.5, in which the navigation task is to proceed "south" and in which there are five m-NaTs modeling the current navigation constraints. While moving in a southerly direction, the robot has passed m-NaTs *A*, *B*, and *C*; consequently, they are of little concern. M-NaTs *D* and *E*, on the other hand, are in the way with respect to the robot's task of moving in a southerly direction. Furthermore, because m-NaTs *D* and *E* represent environmental constraints that must be accommodated, their spins are shown in the figure. Given this information, it is reasonably clear that the immediate task is not to move south as indicated by the s-NaT. Instead, the immediate task is to pass between m-NaTs *D* and *E*, since moving between *D* and *E* satisfies the constraints of moving clockwise with respect to m-NaT *D* and c-clockwise with respect to m-NaT *E* as indicated by the navigation plan. This is not to say that m-NaTs *A*, *B*, and *C* should have no effect on determining the direction of the gradient at the robot's location, rather that it is m-NaTs *D* and *E* that should drive the determination of the gradient direction at the given position. This example illustrates the need for identification of the *immediate navigation objective* in order to focus the process of generating a gradient, for without a focus it is difficult to create a gradient that flowed between m-NaTs D and E.



Isolating the important characteristics of the immediate situation

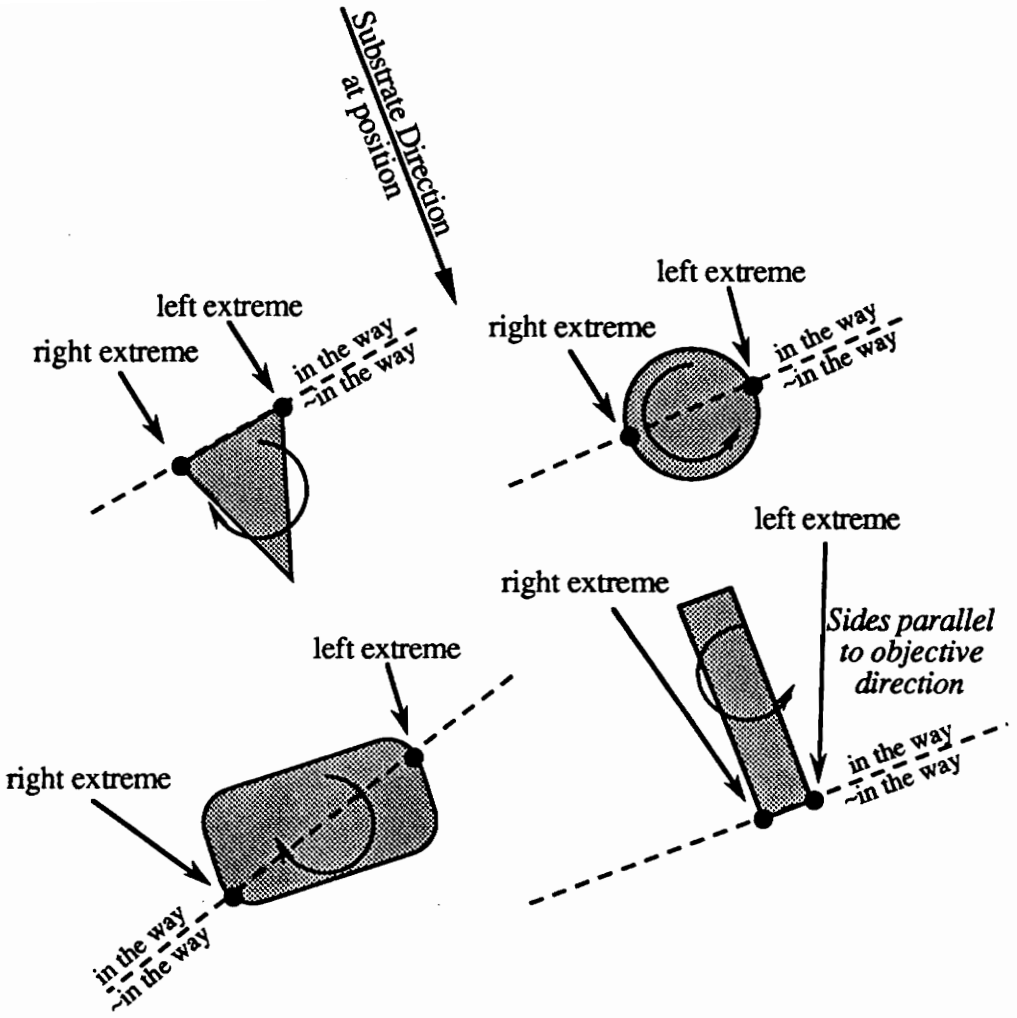
Figure 2.5

In order to identify the immediate navigation objective, a number of geometric properties and relationships are required of each m-NaT in order to characterize the spatial relationship between an m-NaT and the current s-NaT. The first operation needed to identify the immediate navigation objective is a Boolean operation, which is used to determine whether or not a given m-NaT is *in-the-way* with respect to the direction of the s-NaT's gradient at a given location. For example, when one is standing next to a wall with the intention of moving away from the wall, the wall is not in-the-way. However, if the intention is to move to a location on the other side of the wall, then the wall is in-the-way and must be negotiated in order complete the task. The in-the-way test determines whether or not the m-NaT is relevant to accomplishing the navigation task. The intent is that if the m-NaT is considered to be in-the-way, then the gradient should flow around the m-NaT in the direction indicated by the spin on the m-NaT, otherwise the m-NaT has no particular relevance[3]. The in-the-way Boolean operation accepts an m-NaT, a position, and the direction of the s-NaT's gradient at that position. In order to determine whether or not an m-NaT is in-the-way, each m-NaT must provide a geometric operation that accepts a direction and returns two points. These two points represent the right-most and left-most extremes of the m-NaT's body with respect to the direction of the s-NaT's gradient at the given position. If an extreme is a line rather than a single point, then the end point of the line furthest in the direction of the gradient is used to define the extreme. The two points are used to form a line splitting space into two half-planes; in one plane the m-NaT is in-the-way and in the other it is not in-the-way (see examples in Figure 2.6). Thus, moving in the direction of the s-NaT gradient from a point in the edge of the body that is in the half-plane considered to be in-the-way would cause a collision with the body, while a similar point in the other half-plane would move away from the body.

The next geometric relationships occur strictly between a given m-NaT and a given position. Two of the operations are referred to as the *primary clockwise bound* and the *primary c-clockwise bound.* These require that each m-NaT provide an operation that returns the two points on the m-NaT's body representing the *clockwise most visible point* and *c-clockwise most visible point* to some specific position (see Figure 2.7). The primary clockwise bound is defined as the angle of a ray starting from the given position and

---

[3]Not exactly true as will be shown later.

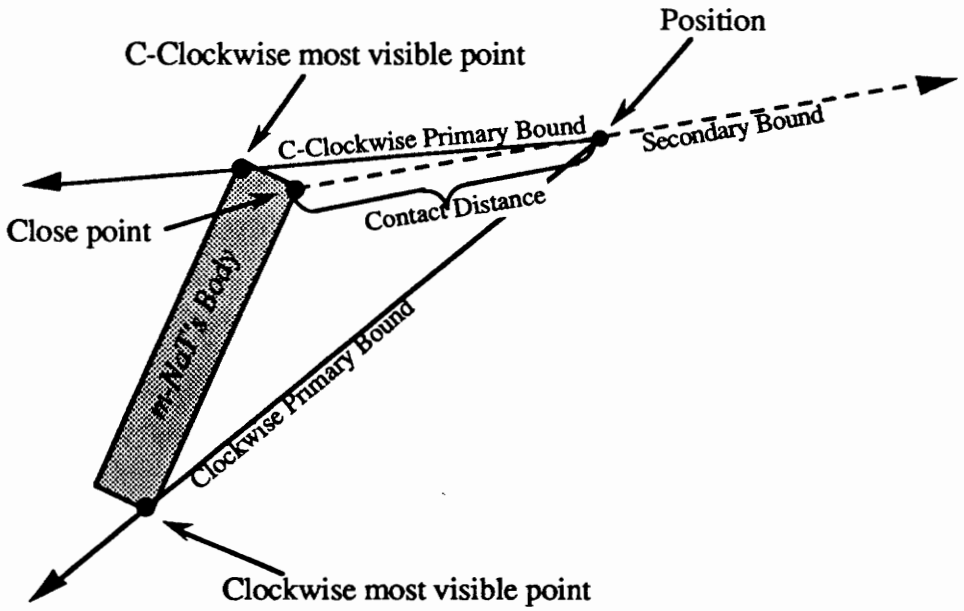


Determining whether an m-NaT is in-the-way or not in-the-way

Figure 2.6

passing through the clockwise most visible point on the body of the m-NaT. The definition of the primary c-clockwise bound is analogous. These two bounds characterize the angle of occlusion produced by the body of the m-NaT as seen from the given position. The next two relationships, referred to as the *contact distance* and the *secondary bound*, require that each m-NaT be capable of providing the point on its body that is closest to a given position. The contact distance is simply the euclidean distance between the closest point on the m-NaT's body and the given position. The contact distance is used to order the m-NaTs in

such a way that the impact of the m-NaTs closer to a given position can be considered before those farther away. The secondary bound refers to an angle defined by a ray starting at the close point on the m-NaT's body and passing through the given position (see Figure 2.7). The primary bound and the secondary bound are used to define what is termed the *viable angular range* of the particular m-NaT (or ϕ), which represents the range of angles from the given position that are compatible with the m-NaT's spin. For example, if the spin of some m-NaT is clockwise, then from a given position ϕ is the clockwise range of angles between the secondary bound and the clockwise primary bound. These geometric relationships are used in combination with the in-the-way Boolean, as well as the spin of an m-NaT, to determine the relevance of an m-NaT with respect to the current navigation task.



Determining the valid angular range for a particular m-NaT

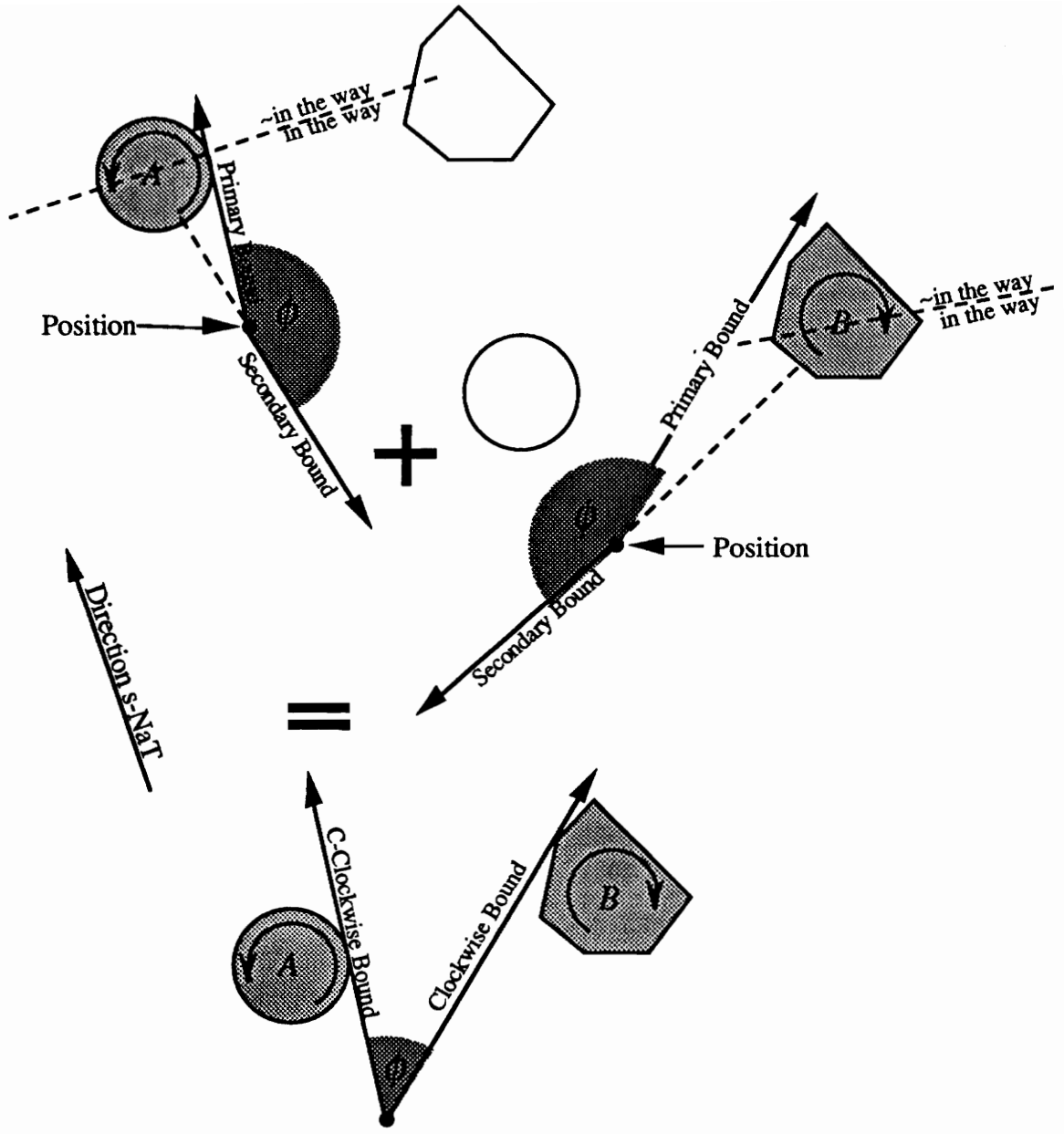
Figure 2.7

The first step in determining the direction of the gradient at a given position is to identify the immediate navigation objective. This is accomplished by considering the way in which the spins of m-NaTs considered to be in-the-way are spatially related to the given position. This reduction of the m-NaTs being considered to only those that are in-the-way can be made because the m-NaTs not in-the-way do not, in general, constrain the robot's

ability to accomplish the navigation task (i.e., the substrate gradient is locally flowing away from the m-NaT). The immediate navigation objective as seen from any given position falls into one of three cases:

- Currently there are no m-NaTs in-the-way.
- Currently one or more m-NaTs are in-the-way; however, the immediate navigation objective can be reduced to that of moving around a particular m-NaT (where the direction around the m-NaT is determined by the m-NaT's spin).
- Currently two or more m-NaTs are in-the-way; however, the immediate navigation objective can be reduced to that of moving between two particular m-NaTs by passing one in a clockwise fashion and the other in a c-clockwise fashion.

The immediate navigation objective is determined by identifying the viable range of angles (or ϕ) in which the robot can move from a given position that will satisfy as many of the constraints in the navigation plan as possible. Consider the situation depicted in Figure 2.8, in which there are two m-NaTs whose constraints are being combined in order to identify the immediate navigation objective. The top portion of the figure shows ϕ representing the constraints imposed by m-NaT *A*, which lies between secondary and primary c-clockwise bounds (moving in a c-clockwise direction). Similarly, the middle portion of Figure 2.8 shows the ϕ associated with m-NaT *B*, which lies between its secondary and primary clockwise bounds (moving in a clockwise direction). Combining the constraints of m-NaTs *A* and *B* occurs simply by identifying a new ϕ which is common to both m-NaTs. The resulting ϕ represents the set of directions that are relatively c-clockwise with respect to m-NaT *A* and clockwise with respect to m-NaT *B* (see the lower portion of Figure 2.8). Thus, this example depicts a case in which the current navigation objective is to move between m-NaTs *A* and *B*.

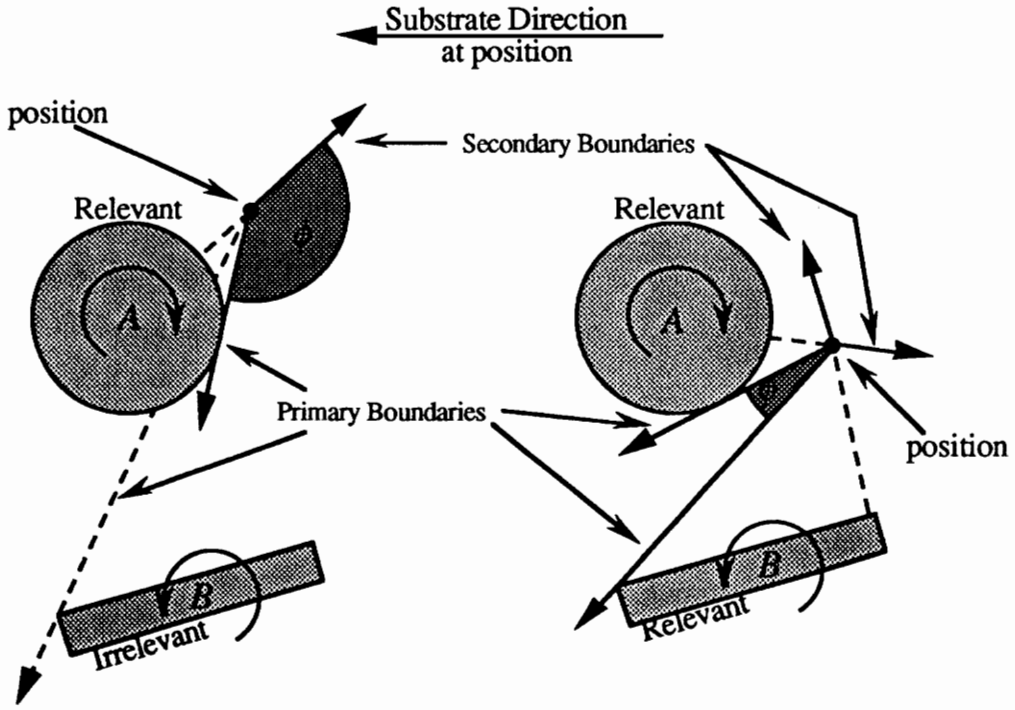


Identifying the immediate navigation objective

Figure 2.8

As a further example, consider the two situations depicted in Figure 2.9. The right and left portions of the figure are exactly the same, except for the position in question. This example illustrates the way in which determination of ϕ is influenced by the position

at which the gradient's direction is being determined . The process of determining ϕ is accomplished by sequentially considering the effects of each of the m-NaTs that are in-the-way (with respect to the position in question and the direction of the s-NaT's gradient at that position), ordered by their contact distance from the position in question. Consider the situation depicted in the left portion of Figure 2.9, in which m-NaT *A* places the first constraint on the definition of ϕ. The result of the constraint imposed by m-NaT *A* is that the primary bound of m-NaT *B* does not lie within ϕ imposed by m-NaT *A*. As a result, m-NaT *B* is ignored and ϕ is defined entirely by m-NaT *A*; thus, the immediate navigation objective is to move around m-NaT *A* in a clockwise fashion. In the right portion of Figure 2.9 the situation is quite different. In this situation, the primary bound of m-NaT *B* does fall within ϕ as defined by m-NaT *A*, the resulting ϕ is determined by the combined effect of both m-NaTs, and thus the immediate navigation objective is to move between the two m-NaTs.



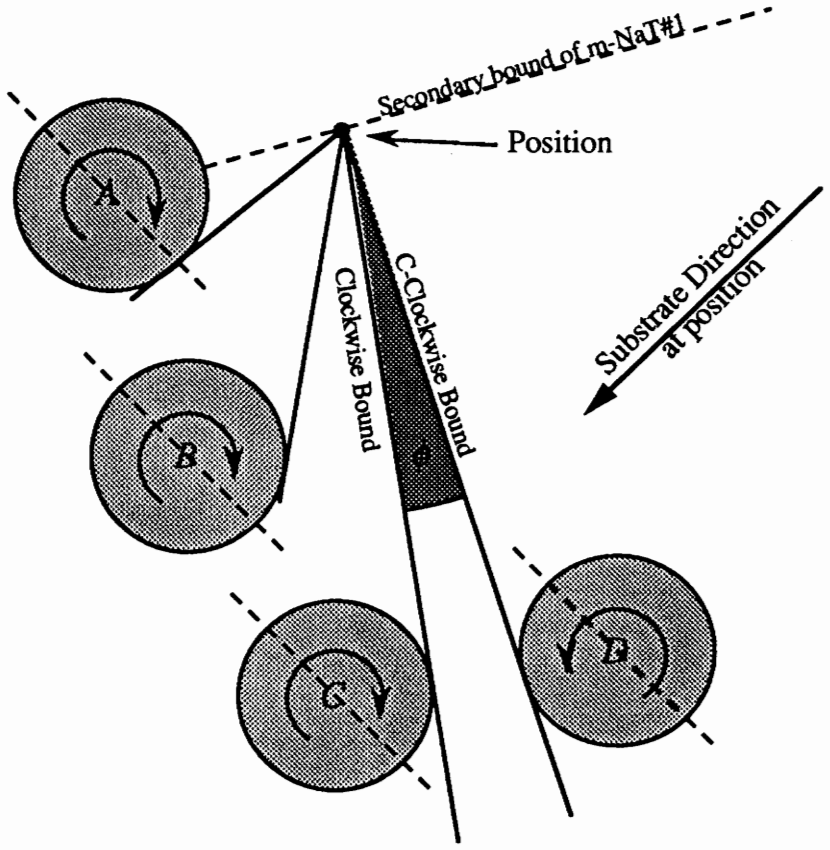
The line between relevancy and irrelevancy

Figure 2.9

The general process of determining ϕ and identifying those m-NaTs that define the final clockwise and c-clockwise bounds of ϕ starts with the closest m-NaT, as determined by the contact distance, and continues incrementally with the next next most remote m-NaT. The process is terminated and the immediate navigation objective is identified when either all of the m-NaTs have been considered, or one of the m-NaT's relevant primary bound (i.e., the one associated with the spin of the m-NaT) does not fall within the currently defined ϕ. This latter condition ensures that distant constraints do not unduly influence the situation when there are local constraints that might soon become relevant.

Consider the more elaborate situation depicted in Figure 2.10. The figure shows a situation in which there are four m-NaTs, all of which are considered to be in-the-way with respect to the indicated position. The three m-NaTs closest to the position in question all have a c-clockwise spin, while m-NaT *D* has a clockwise spin. Determining the current navigation objective at the indicated position begins by considering *A*'s effect and continues until either there are no more in-the-way m-NaTs to consider or the constraints implied by one of the m-NaT's cannot be met. From the given position, *A* places a constraint on ϕ that restricts it to the range of angles between *A*'s secondary bound and its primary clockwise bound (moving in a clockwise direction). M-NaT *B* is the next closest; therefore, its constraints are considered next. The primary bound of *B* falls inside the currently defined ϕ and, as a result, its constraints are combined with those imposed by m-NaT *A*. Consequently ϕ is constrained to fall between the secondary bound of *A* and the primary clockwise bound of *B* (moving clockwise). Similar to m-NaT *B*, m-NaT *C* further refines ϕ to be the clockwise range of angles between *A*'s secondary bound and the *C*'s primary clockwise bound. The constraints thus far would indicate that the immediate navigation objective would be to move around m-NaT *C*. Finally, the constraints imposed by m-NaT *D* are examined. *D*'s spin is c-clockwise, thus its primary c-clockwise bound is the one that is relevant. Because *D*'s primary c-clockwise bound falls within the current ϕ, *D*'s constraints must be integrated. The final definition of ϕ at the position in question is the clockwise range of angles between the primary c-clockwise bound of *D* and the primary clockwise bound of *C*. Thus, the resulting immediate navigation objective at the given position is to move between m-NaTs *C* and *D*. A gradient summation approach presented with the same situation would have implicitly identified m-NaTs *A* and *B* as being the most important at the given position, resulting in the robot bouncing around the borders of m-

NaT *A*, then *B*, and finally around *C*, as opposed to moving directly to the gap between *C* and *D*.



Determining the viable angular range in the presence of multiple m-NaTs

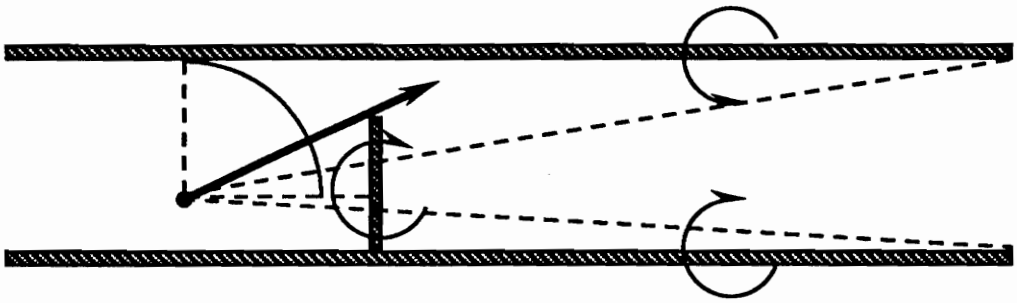
Figure 2.10

The order in which the m-NaTs are considered proceeds from nearest to farthest as determined by the contact distance from the given position. However, this approach leads to some undesirable effects. Consider the situation in Figure 2.11, in which a hallway situation is depicted. The navigation task in the figure is to move down the hallway from left to right while avoiding the barrier that blocks part of the hallway by moving clockwise around it. At the indicated position, all three of the m-NaTs are considered in-the-way. Consider the way the algorithm defined thus far for identifying the immediate navigation objective will work in this case. Because the walls of the hallway are closer to the indicated position than the barrier blocking the hallway, they are considered first. This results in

restricting ϕ to that indicated by the two long dashed lines. When an attempt is made to incorporate the effects of the barrier, the process is halted because the clockwise primary bound of the barrier does not fall in the current ϕ. Thus, at the indicated position, the immediate navigation objective is to move between the walls of the hallway, resulting in a gradient that flows directly toward the wall until the barrier is closer than the upper wall. Although the gradient will ultimately flow up the hall past the barrier, it would be better for the immediate navigation objective at the indicated position to move around the barrier rather than simply moving up the hallway. To accomplish this, the scheme used to order the m-NaTs for incorporation into the immediate navigation objective is modified. Rather than ordering them by their contact distance, they are ordered by the distance between the given position and either the clockwise or c-clockwise most visible point: whichever corresponds to the m-NaT's spin. The line segments defined by the given position and either the clockwise or c-clockwise most visible point are referred to respectively as the *primary clockwise segment* and *primary c-clockwise segment.* Having made this simple modification, the immediate navigation objective in the hallway and barrier example becomes moving around the barrier. However, simply considering the constraints based on the length of their primary segments is not sufficient. Consider a situation in which the barrier in the aforementioned hallway example is on the other side of the wall and it has a c-clockwise spin. Because its primary c-clockwise segment is still shorter than the primary segments of the m-NaTs representing the walls of the hallway, the barrier's constraint is considered first. This results in a ϕ that would direct the gradient into the wall. To alleviate this, an m-NaT affects the determination of the immediate navigation objective only if its relevant primary bound is visible from the position at which the immediate navigation objective is being defined. This is accomplished by determining whether any of the m-NaTs intersect the relevant primary segment of the m-NaT being considered. If it is not visible, then it is discarded and the process continues with the next m-NaT in the ordering. If the relevant bound is visible, then the m-NaT does affect the determination of the immediate navigation objective. To support this, each m-NaT must provide a Boolean test which determines if the body of the m-NaT is incident with a given line segment. Thus, other than the change in ordering and the test for visibility, the process of identifying the immediate navigation objective remains the same.

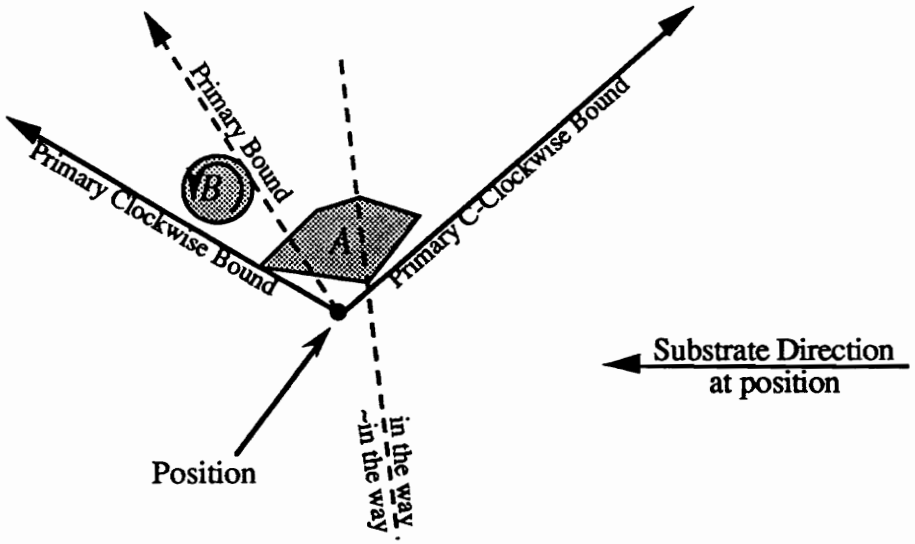
For simplicity, the approach described thus far for determining the immediate navigation objective has omitted any consideration of the m-NaTs that are not considered to

be in-the-way. Sometimes, however, m-NaTs that are not in-the-way can play a role in determining the immediate navigation objective at a particular position. For example, consider the situation depicted in Figure 2.12, in which m-NaT *A* is not in-the-way and m-NaT *B* is considered in-the-way. Given the algorithm defined thus far for identifying the immediate navigation objective, because m-NaT *B* is the only m-NaT that is in-the-way, the immediate navigation objective would be to move around its c-clockwise bound. However, such an immediate navigation objective would cause a collision with m-NaT *A*. So the combining function must account for the constraints imposed by the m-NaTs that are not in-the-way.



Negotiating a hallway with a barrier

Figure 2.11



M-NaTs which are not in-the-way are not always irrelevant

Figure 2.12

To rectify this problem, all m-NaTs must be considered at least superficially when the immediate navigation objective is defined. The resulting algorithm used to identify the zero, one or two m-NaTs which define the immediate navigation objective at a given position is as follows:

1) Order the m-NaTs from nearest to farthest, based on the length of their relevant primary bound segment. Note that m-NaTs not in-the-way must be entered twice; once for each primary bound. This is because the spin of m-NaTs that are not in-the-way does not matter and thus it is possible for either of their bounds to place constraints on ϕ.

2) Set the initial clockwise and counter clockwise bounds of ϕ to be opposite the direction of the substrate gradient at that position.

3) Combine the constraints imposed by the next closest m-NaT to further constrain ϕ.

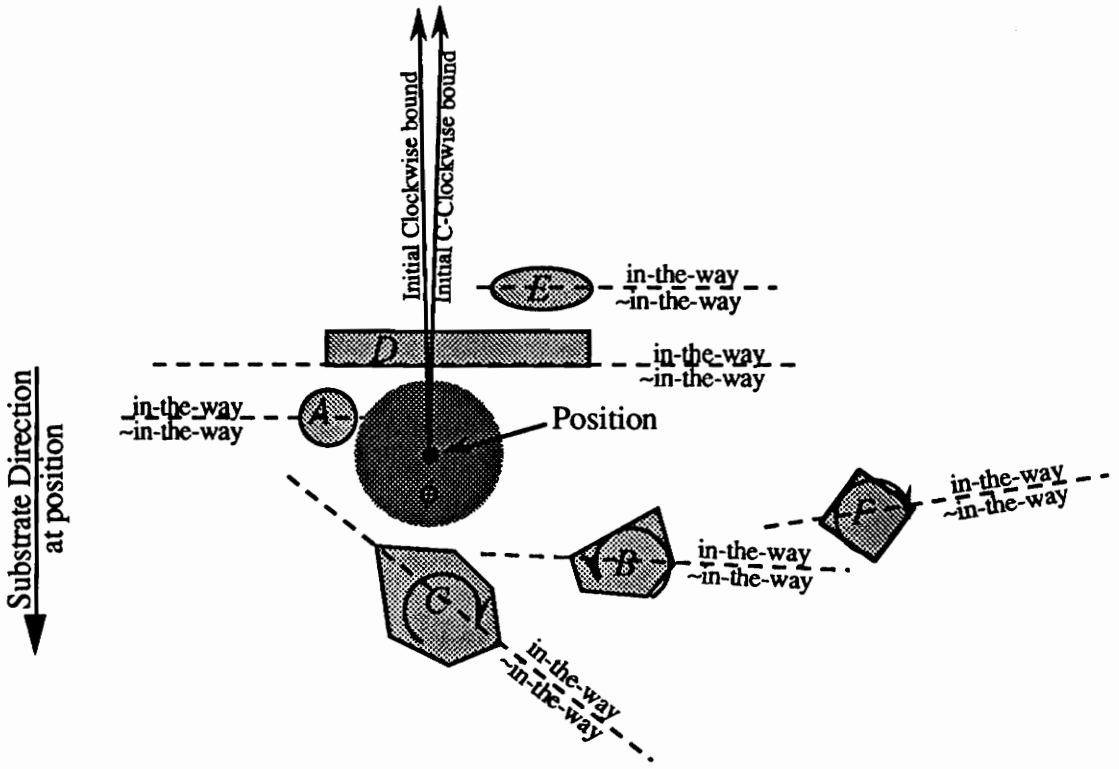
4) If the constraints of the next closest m-NaT are inconsistent with ϕ as currently defined, then stop and return the current immediate navigation objective; otherwise, if there are no more m-NaTs to process, then stop and return the current immediate navigation objective; otherwise repeat step 3 with the next entry in the list.

As an example of this process, consider the situation depicted in Figure 2.13, in which there are six m-NaTs which must be combined in order to identify the immediate navigation objective. The process starts by ordering the m-NaTs based on the distance to their constraints. This results in the following constraint ordering:

1) *A*'s clockwise
2) *A*'s c-clockwise
3) *D*'s clockwise
4) *C*'s clockwise
5) *E*'s clockwise
6) *D*'s c-clockwise
7) *B*'s c-clockwise
8) *E*'s c-clockwise

9) *F*'s clockwise

The second step in the process sets the initial clockwise and c-clockwise bounds of ϕ to be opposite the direction of the s-NaT's gradient at the given position as shown in the figure.

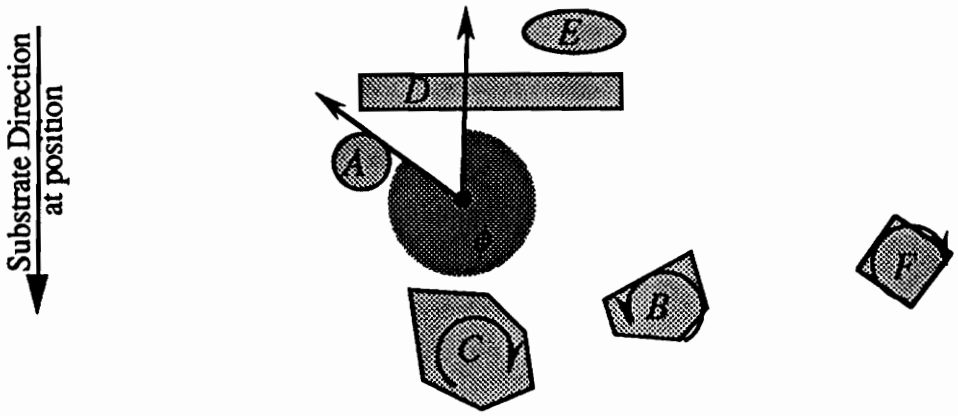


Example to illustrate the process used to identify the immediate navigation objective

Figure 2.13

The combining process begins by modifying ϕ (which at the beginning spans two pi radians) to accommodate the first constraint on the ordered list. In this situation the first constraint is the c-clockwise bound of *A*. Because *A* represents an m-NaT that is not in-the-way and there is no restriction as to the way m-NaTs not in-the-way are passed, *A*'s constraint is handled a bit differently than those of m-NaTs that are in-the-way. A bound of an m-NaT that is not in-the-way affects ϕ by modifying the bound of ϕ that was originally (i.e., at the end of step 2) most closely aligned with the direction of the

constraint[4]. Thus, in the case of A's c-clockwise bound, it is the clockwise bound of ϕ that is affected (see Figure 2.14). In a similar manner, the process is repeated for A's clockwise bound, resulting in the clockwise bound of ϕ being further constrained. The reason that both of the bounds of an m-NaT not in-the-way must be entered into the list is that it is possible for its constraints to have an impact on both of the clockwise and c-clockwise bounds of ϕ. For example, if m-NaT D were the only m-NaT, then it would affect both the clockwise and c-clockwise bounds of ϕ.



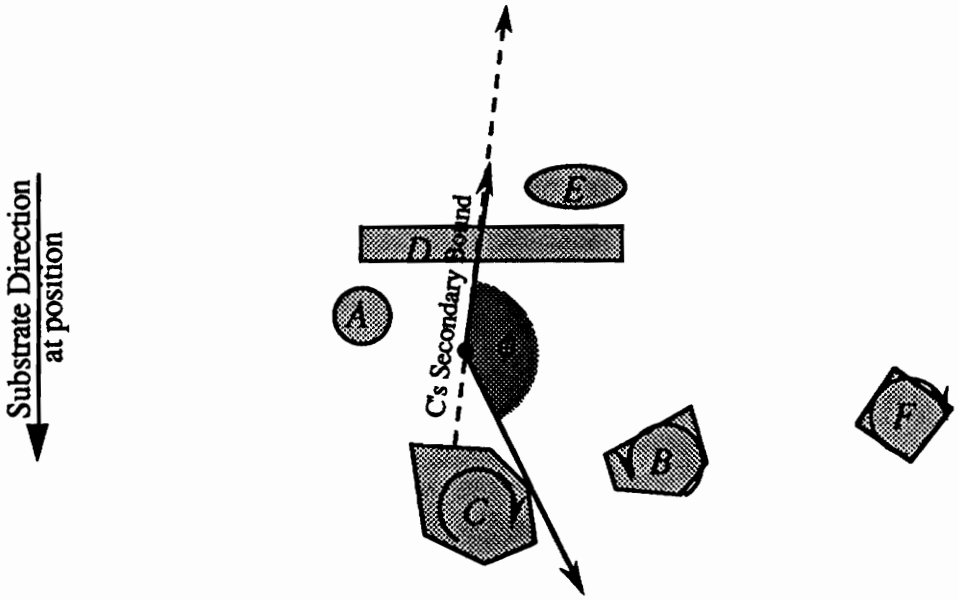
Including the c-clockwise constraint of m-NaT A

Figure 2.14

The next constraint is D's clockwise bound. This constraint does not fall within ϕ as currently defined, and because D is not in-the-way the constraint is simply ignored and the process continues with C's clockwise bound. Because C's primary bound is visible and within ϕ, C's effects can be combined. Because C is in-the-way, its constraints are more explicit, resulting in the clockwise bound of ϕ being set to C's clockwise primary bound. C's secondary bound affects the c-clockwise bound of ϕ , advancing it slightly in a clockwise direction (see Figure 2.15). The next bound to be combined is E's clockwise

[4]The angle between the clockwise bound of ϕ and some other angle is measured by the c-clockwise angle between the two, starting at the clockwise bound of ϕ (similar but reversed for c-clockwise bound of ϕ).

bound; however, because E's bound is occluded by D, its constraint is skipped and the process advances to the next closest constraint.

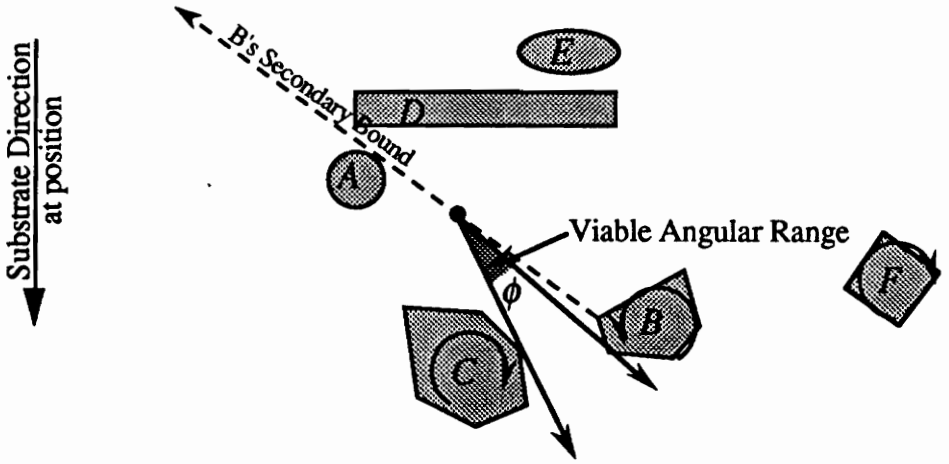


The result of having included the clockwise constraint of m-NaT C

Figure 2.15

The next constraint is D's primary c-clockwise bound. Because the angle of this constraint is closer to the original value of ϕ's c-clockwise bound, ϕ's c-clockwise bound is affected, advancing to D's c-clockwise bound. The process continues with B's c-clockwise bound; since it falls both within ϕ as currently defined and visible, it is possible for B's c-clockwise bound to be incorporated. Because B is in-the-way, its c-clockwise bound affects ϕ's c-clockwise bound. Furthermore, B's secondary bound affects the clockwise bound of ϕ; however, it has no effect, as ϕ's clockwise bound has already been sufficiently constrained (see Figure 2.16). The next constraint in the sequence is the c-clockwise bound of E. M-NaT E is not in-the-way and its constraint is neither visible nor within ϕ. As a result it is ignored. Finally, F's constraint is considered. However, although its constraint is visible, it is not within ϕ and thus its constraint cannot be incorporated. This terminates the combining process not only because it happens to be the last constraint, but also because F is in-the-way and its clockwise constraint is visible and does not fall within ϕ as currently defined. The resulting immediate navigation objective is

to move between m-NaTs C and B, passing C in a clockwise fashion and passing B in a c-clockwise fashion.



Combining D's constraint

Figure 2.16

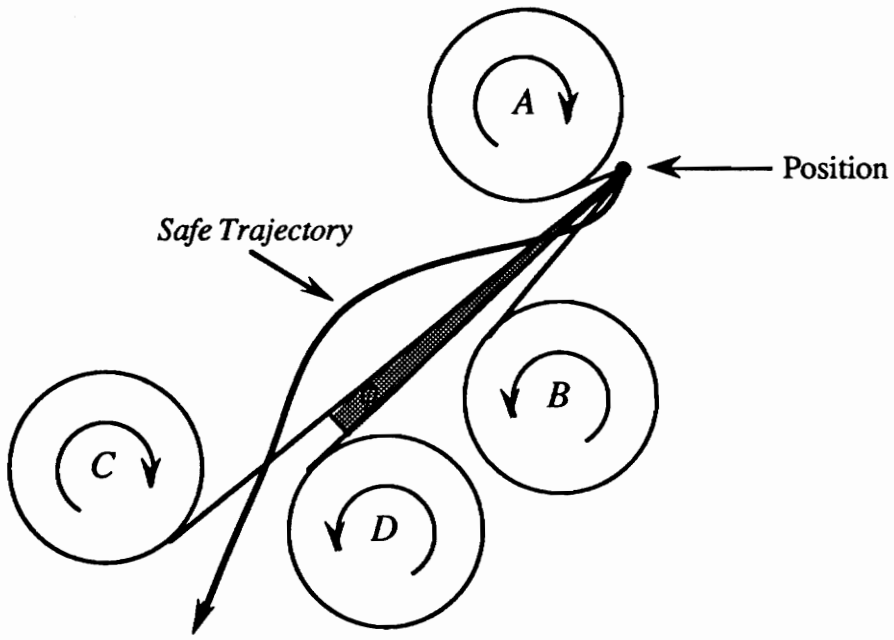
2.3.2. The Gradient

Now that the process of identifying the immediate navigation goal has been presented, there is still the problem of determining the direction of the gradient at a particular position. The process of identifying the immediate navigation objective results in a range of viable angles being defined. One approach to determining the resulting gradient goes as follows:

- If the direction of the s-NaT's gradient at the given position lies in the range of viable angles, then use the direction as determined by the gradient, or else use either the clockwise or c-clockwise bound on the viable angular range, whichever is more closely aligned with the substrate.

However, this scheme is not particularly safe, since moving along one of the bounds of the viable angular range may cause the robot to move too close to the body of an m-NaT. In addition it is possible that m-NaTs other than those defining the immediate navigation objective might have local effects that should take precedence if the robot is to move safely through the environment. Consider the situation depicted in Figure 2.17, in which there are four m-NaTs and the immediate navigation objective has been identified as moving

between m-NaTs *C* and *D*. More critical, however, is the problem of passing between m-NaTs *A* and *B* on the way to *C* and *D*. While the immediate navigation objective may provide a focus for determining the fundamental constraints which must be accommodated in order to accomplish the current navigation task, the effects of the other m-NaTs must play a role in determining the direction of the gradient at a given position. The resulting gradient field should flow smoothly between the m-NaTs at a safe distance (see the dark line in Figure 2.17 as an example).



Bounding constraints are not always the most immediate constraints

Figure 2.17

One technique that would ensure that the gradient does not flow too closely to any of the obstacles represented by the m-NaTs would be to enlarge the bodies of the m-NaTs to provide a margin of safety, similar to configuration space techniques. However, this limits the abilities of a robot using the gradient field in situations where the robot must move very close to obstacles in order to accomplish a navigation task. Rather than growing the bodies of the m-NaTs, the technique developed here associates a danger zone with each m-NaT. The danger zone represents the region of space that is outside the m-NaT's body but within some distance of the m-NaT's body. To accomplish this, each m-NaT must provide a

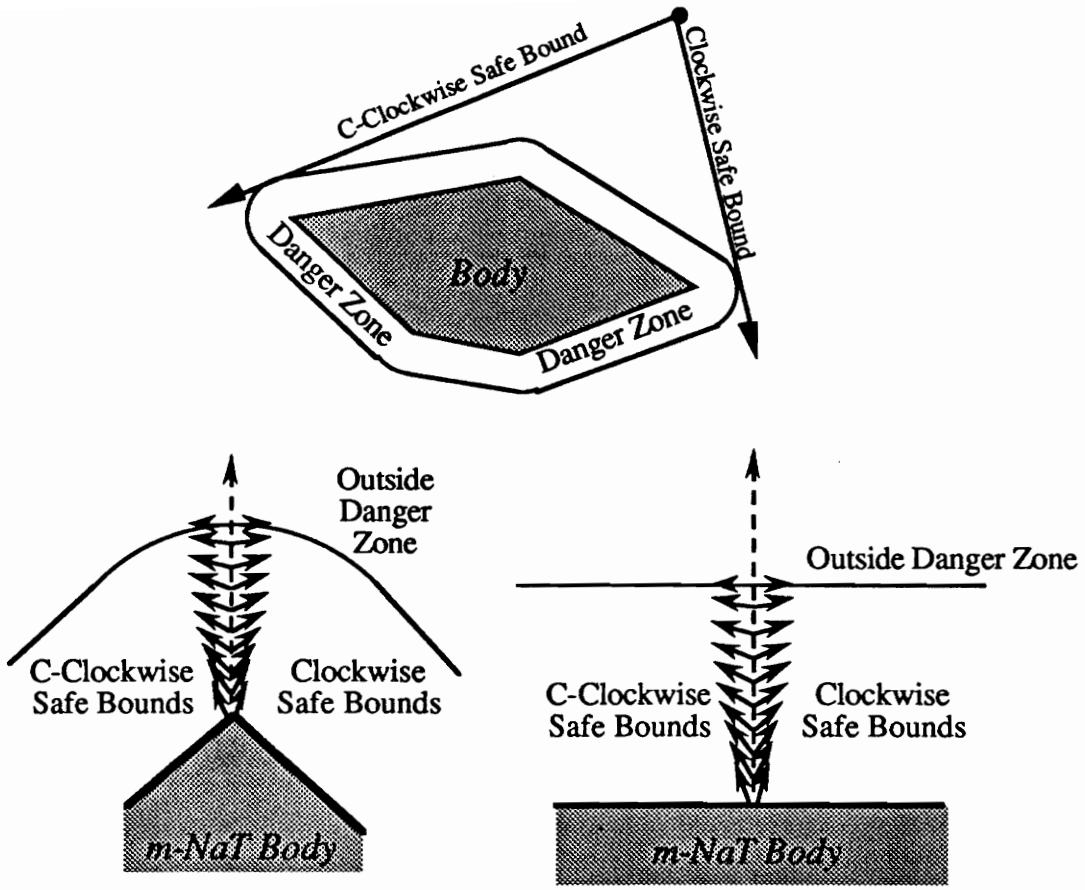
Boolean operation which accepts a position and determines whether it resides inside or outside the m-NaT's danger zone (see the top portion of Figure 2.18). For example, if the body of an m-NaT is a circle, then the danger zone would also be a circle. In order to derive the direction of the gradient at some location, each m-NaT must provide additional angular bounds: the *clockwise safe bound* and the c-*clockwise safe bound.* The values of these two bounds indicate the safe direction of travel around the obstacle in either a clockwise or counter-clockwise direction from any given position. The characteristics of the safe bounds depend greatly on whether the position at which the bounds are being calculated resides inside or outside of the m-NaT's danger zone. When the position is outside of the m-NaT's danger zone, the safe bounds connect the given position to the clockwise and c-clockwise most visible points on the outer edge of the m-NaT's danger zone. When the position is inside the m-NaT's danger zone, the safe bounds point out and around the m-NaT's body (see the lower portion of Figure 2.18). The angles of the safe bounds are tangential to the safe bound as the position approaches the outer edge of the danger zone and becomes nearly perpendicular to the body as the position approaches the body in this case. The safe bounds are used by the final stage of the combining function to determine the direction of the gradient at a given position.

The algorithm used to determine the direction of the gradient at a given position is as follows:

1) Identify the immediate navigation objective using the process described earlier.

2) Define an optimal direction as follows: If the direction of the s-NaT's gradient at the given position falls within the viable angular range, then set the optimal direction equal to the direction of the s-NaT's gradient; otherwise set the optimal direction equal to either the clockwise or c-clockwise bounds on the viable angular range, whichever is more closely aligned with the direction of the s-NaT's gradient at the given position.

3) Identify those m-NaTs that are relevant to the situation. With respect to the given position and the identified optimal direction, all m-NaTs fall into one of four categories:



Defining an m-NaT's safe bounds

Figure 2.18

- not in-the-way with the position outside the danger zone
- not in-the-way with the position inside the danger zone
- in-the-way with the position outside the danger zone
- in-the-way with the position inside the danger zone

Those m-NaTs categorized as not in-the-way where the position is outside the danger zone play no further role in defining the gradient. Any of the remaining m-NaTs whose safe bounds contain the optimal direction are considered relevant to the situation (i.e., if the optimal direction is in the range of angles between the clockwise safe bound and the c-clockwise safe bound moving in a clockwise direction, then include the associated m-NaT). In addition, the m-NaTs which

form immediate navigation objective are also included, if they have not already been included.

4) Each of the m-NaTs identified as being relevant contributes a factor in order to determine the direction of the gradient. Those whose safe bounds contain the optimal direction contribute an influence that points in the direction of the m-NaT's safe bound most closely aligned with the optimal direction and weighted inversely to the contact distance to the m-NaT. There is one special case that must be considered separately. Those m-NaTs that form the immediate navigation objective and whose safe bounds do not contain the optimal direction each contribute a vector that points in the direction of the optimal direction and is weighted inversely to the contact distance. This keeps local effects from causing the gradient to flow too closely to the bodies of the m-NaTs which define the immediate navigation objective.

5) Finally, all of the contributing vectors are summed and a resultant direction emerges. While this might sound exactly like gradient summation, the difference is that only those m-NaTs that are relevant to the situation are included in the summation and their influencing factor is chosen to support the identified optimal direction.

As an example of the manner in which the direction of the gradient is calculated, consider the situation depicted in Figure 2.19, in which the details of each of the five steps are presented. The final value of 62.18 degrees is determined as follows:

1) Using the process presented in the last section, the immediate navigation objective is determined: move between m-NaTs *D* and *E*.

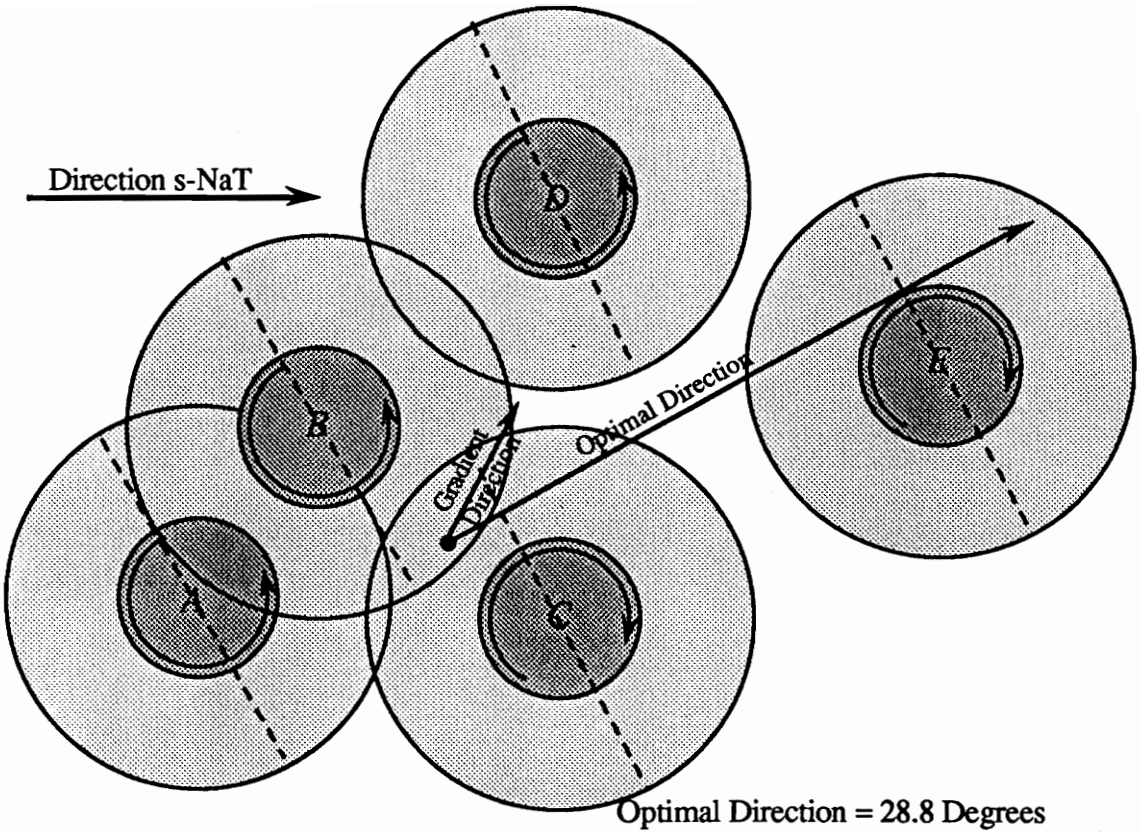
2) The optimal direction from the given position is 28.8 degrees: the angle of the primary clockwise bound of m-NaT *E*.

3) Applying step three, m-NaT *A* is eliminated from further consideration as it is not in-the-way and the given position lies outside of its danger zone. M-NaT *B* is also eliminated because the clockwise range of angles between its c-clockwise and clockwise safe bounds contain the optimal direction of 28.8 degrees. m-NaTs *C* and *E* are relevant and are carried on to the next step in the process because the optimal direction points into the range of angles between their safe bounds.

While the optimal direction is not contained within m-NaT *D*'s safe bounds, due to the fact that it partially defines the immediate navigation objective, it is carried on to four as well.

4) Each of the remaining m-NaTs (i.e., *C*, *D*, and *E*) contributes to the determination of the final gradient a factor weighted inversely to the distance between the given position and the body of the m-NaT. Since the optimal direction falls within the safe bounds of m-NaTs *C* and *E*, these m-NaTs contribute a factor that is inversely proportional to their contact distance with the given position in the direction of their clockwise safe bounds because it is the bound most closely aligned with the optimal direction. M-NaT *D* is handled slightly differently; it also contributes a factor that is inversely proportional to the distance between the given position and its body. However, the direction of that factor is the same as the optimal direction. The reason for D's influence is to ensure that the gradient flows in the direction of the optimal direction in the absence of stronger influences.

5) Calculation of the final gradient direction is shown in Figure 2.19, in which the derivation of the 62.18 degrees is shown. Although the direction of the gradient at that location points into *D*'s body, this direction is intended to serve only as guidance to the robot for a brief period of time. After moving in that direction for a short time, a new value is calculated. The new direction would be influenced less by m-NaT *C* and more by m-NaTs *D* and *E*, causing the gradient to flow between those two m-NaTs rather than toward *D*'s body.



Optimal Direction = 28.8 Degrees

M-NaT	Distance	Direction	1/dist * Cos(dir)	1/dist * Sin(dir)
A	0.8321	NA	NA	NA
B	0.4518	28.8 Degrees	NA	NA
C	0.2672	71.2 Degrees	1.2061	3.4528
D	1.4557	28.8 Degrees	0.6020	0.3309
E	2.2581	43.1 Degrees	0.3455	0.3026
			Sum = 2.1563	Sum = 4.0863

Gradient Direction = arcTan(4.0863/2.1563) = 62.18 Degrees

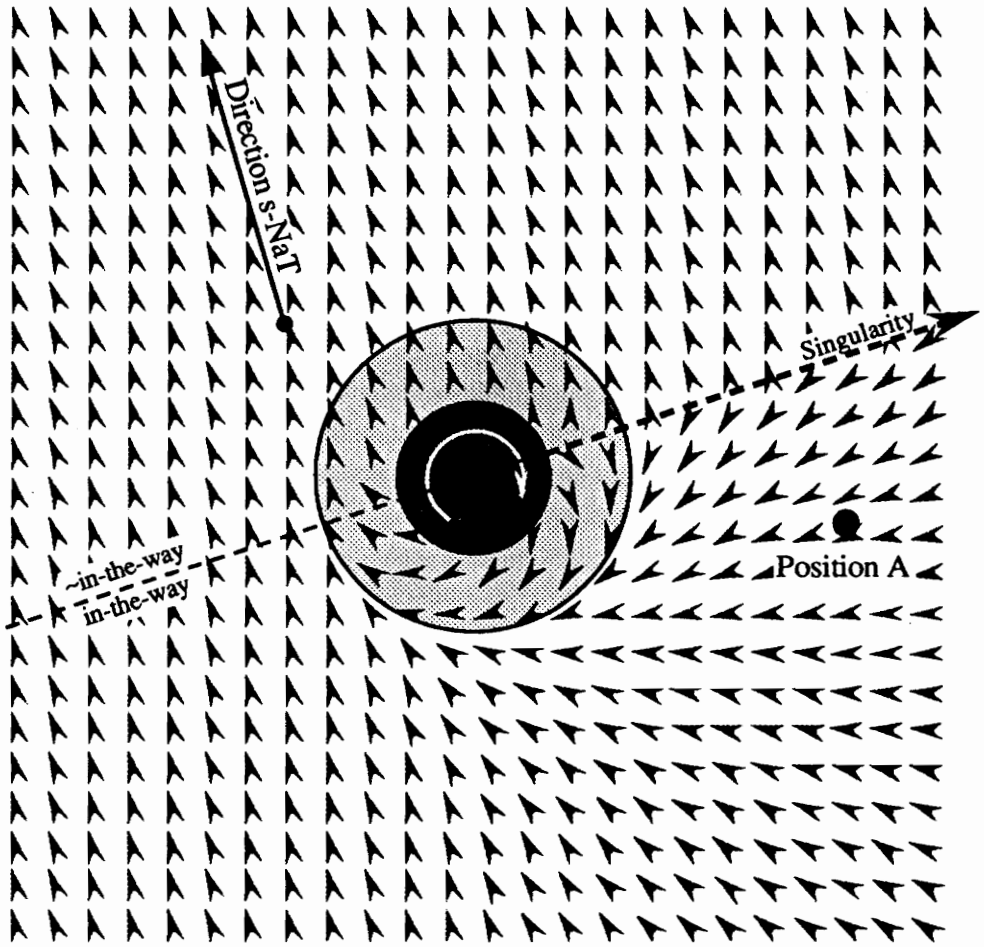
Calculating the direction of the gradient at a given position

Figure 2.19

2.3.3. NaT Plans to Gradients

NaTs form the building blocks for constructing navigation plans. The combining function transforms a NaT-based navigation plan into a preferred direction of travel from a given position in service of the navigation plan. Although the combining function does not actually transform a collection of NaTs into a gradient field (it only computes the preferred direction of travel at a given position), it is convenient to talk as though it did. Figure 2.20 shows a gradient field generated by the combining function transforming a direction s-NaT and a circular m-NaT into a preferred direction of travel at a number of locations. The dashed line separating the figure into two regions represents the line dividing space into two half-planes: one in which the m-NaT is considered in-the-way and the other in which it is not. The gradient in the half-plane in which the m-NaT is in-the-way flows around the m-NaT in a clockwise fashion; where the m-NaT is not in-the-way the gradient flows in the direction of the s-NaT's gradient, except in the m-NaT's danger zone, where the gradient flows away from the m-NaT. For example, consider the direction of the gradient at location A in the figure. Because the m-NaT at location A is in-the-way and the m-NaT's spin is clockwise, the gradient is directed toward the clockwise safe bound of the m-NaT. From location A, the shorter way to get around the m-NaT would have the gradient pointing in the direction of the s-NaT. However, if the system creating the navigation plan had intended for the gradient to flow c-clockwise around the m-NaT at that location, then the spin of the m-NaT should have been reversed. The reason that the influence of the m-NaT's spin is so strong is because it is assumed that the navigation plan has been constructed with knowledge of larger constraints; sometimes it is better to go the long way around something in order to avoid encountering another situation. Finally, note the existence of a singularity in the gradient field, where there is an extreme discontinuity in the flow of the gradient. The singularity is a result of the fact that there is a discrete transition between an m-NaT being in-the-way and not in-the-way. The existence of the singularity simply points out that there are inherent limitations to the use of NaTs. However, unlike those of gradient summation approaches, the limitations are controllable. For example, if the existence of the singularity is a problem, the spin of the m-NaT should be reversed, causing the gradient to circle around the m-NaT in the opposite direction. While it is possible to construct NaT-based navigation plans which will cause the combining function to produce degenerate gradients, the approach allows explicit control over the creation of

the gradient, so that it is relatively easy to construct a navigation plan that results in a gradient which reflects the intentions of that plan.

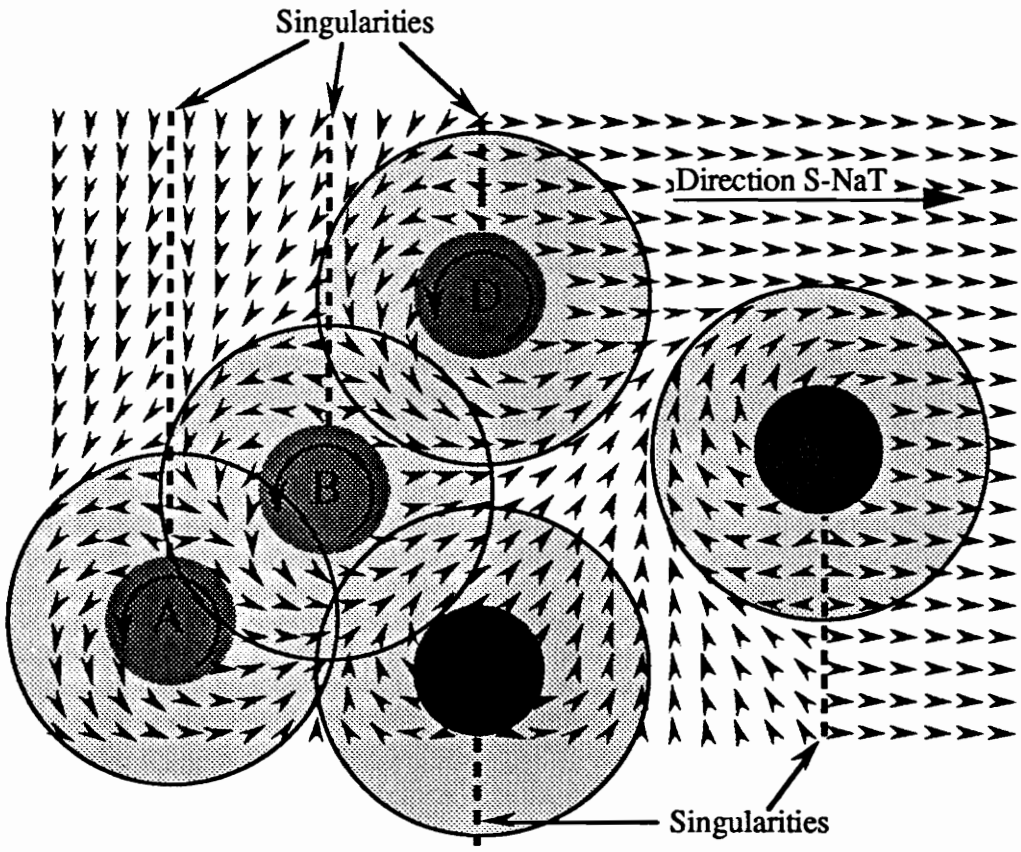


The combining function produces a singularity

Figure 2.20

As an example of the way in which the resulting gradient reflects the intention of the navigation plan, consider the situation in Figure 2.21 which shows a gradient field produced by the combining function operating on a navigation plan constructed from five m-NaTs and a direction s-NaT. The navigation plan built from the NaTs assumes that the robot is located somewhere among the m-NaTs, and that the intended action is to move out from amongst the m-NaTs by proceeding up and to the right. The figure shows the way the gradient flows into the center of the open spaces between the m-NaTs until it is beyond

the influence of the m-NaTs near the top right of the figure. Despite the clear gap between m-NaTs *C* and *E*, the gradient still flows up around *E*. This is because the spin on *E* indicates that the intention of the navigation plan is to move around the top of *E*. If the creator of the navigation plan intended the robot to move out through the gap between m-NaTs *C* and *E*, then the spin on m-NaT *E* should be c-clockwise instead of clockwise. Furthermore, because the intention of the navigation plan is to move up between the m-NaTs, all of the singularities are on the outside, away from the intended region of operation. If the robot is to operate on the top side of m-NaTs *A*, *B*, and *D*, then their spins should be reversed causing the singularities to move to the other side and the gradient to flow up and around the top of m-NaT *D* before proceeding off in the direction of the s-NaT's gradient.



A resulting gradient produced from multiple m-NaTs and a direction s-NaT

Figure 2.21

NaTs are used for building navigation plans and the combining function assumes that the navigation plan has properly encoded the relationships between the obstacles in the environment and the navigation task to be accomplished. NaTs and the combining function do not solve path-finding problems. They do provide a way for a navigation system to construct flexible navigation plans. Furthermore, the combining function allows the navigation system to avoid having to worry about the details of the way the plan provides guidance for controlling the robot's actions.

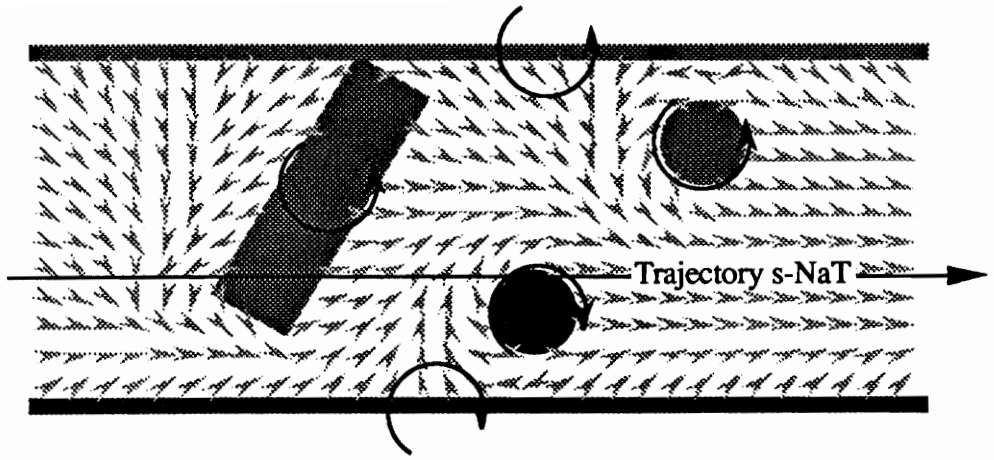
2.4. Building NaTs: A few examples

The previous section introduced navigation templates (NaTs) as a means of constructing navigation plans and showed the way the combining function uses the NaTs to extract tactical information for guiding the robot's actions. This section presents two examples of the way NaTs are used to build a sequence of local navigation plans in service of a global navigation task. The first example shows the way NaTs are used to construct plans for moving around an office setting. The second example presents a situationally driven solution to one of the navigation tasks introduced in Chapter 1.

2.4.1. Getting to Joe Random's Office

Navigation templates are intended to be used for navigating through the local environment in service of a local navigation task. Determining which local navigation task is appropriate at any particular time is resolved at a higher level, where larger navigation issues are reasoned about. Consider once again the problem of getting to Joe Random's office, where the basic global navigation plan is as follows: "proceed up the hallway", "turn right at the intersection", and then "enter Joe's office". This is broken down into three basic local navigation tasks of "negotiate a bounded aisle", "make a transition from traveling along one aisle to traveling along another", and "make a transition from moving along an aisle to passing between two obstacles". The plans used to accomplish any of these tasks must be sensitive to additional environmental constraints that are encountered along the way.

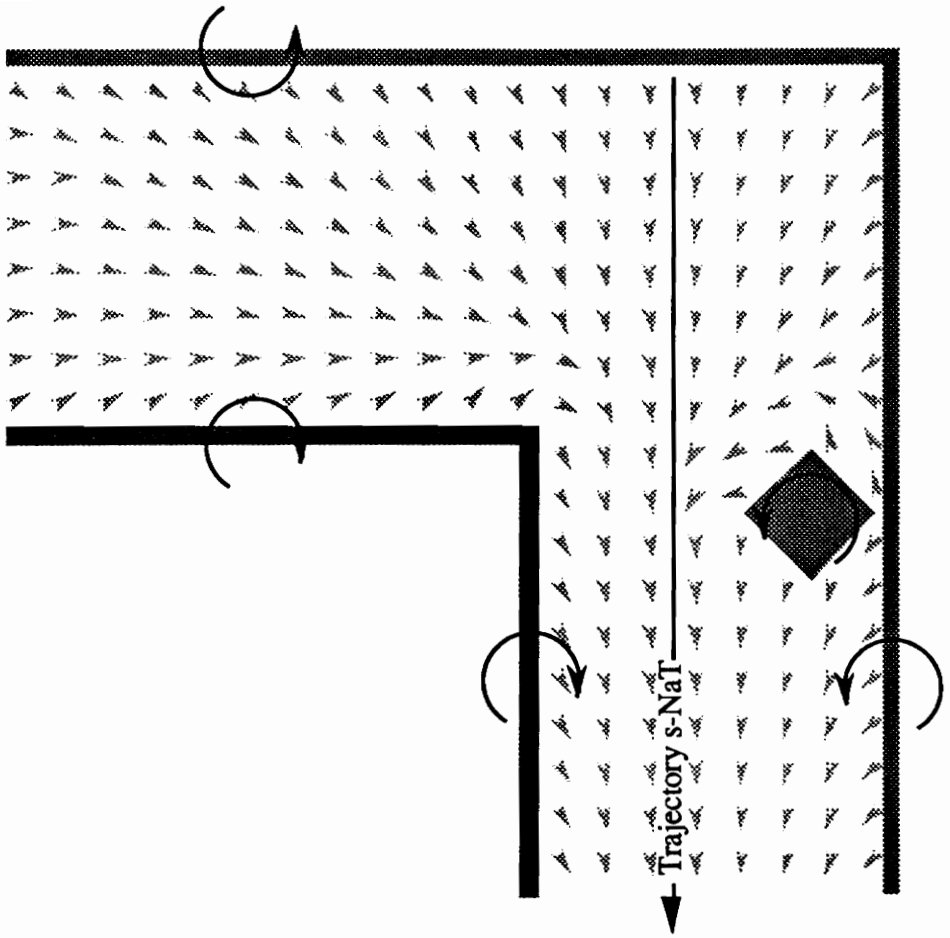
Consider first the task of traversing along a hallway bounded by two walls. Figure 2.22 shows such a situation, in which the current substrate navigation template is a trajectory s-NaT situated slightly to the right of the center of the hall. As expected, the navigation plan indicates that, to traverse the hallway, the robot should move c-clockwise with respect to the top wall and clockwise with respect to the bottom wall. If the s-NaT and the m-NaTs representing the right and left wall were the only NaTs needed to characterize the situation, then the immediate navigation objective at every position within the hallway would be to move between the two walls, and the resulting optimal direction would largely default to the direction indicated by the optimal direction, except when the position is close to one of the walls. However, the situation depicted in Figure 2.22 indicates that there are a number of additional constraints that must be incorporated into the navigation plan. Moving down the hall, the first obstacle encountered is a large barrier which completely blocks the hall except for an opening between the barrier and the bottom wall. Upon encountering such an obstacle, the process maintaining the current navigation plan might determine that, in order to get by this obstacle, it is appropriate to go c-clockwise around it. Once this decision has been made and the new m-NaT has been incorporated into the navigation plan, the combining function begins returning gradient directions that move the robot down and around the bottom of the barrier. Even in the area in front of the barrier next to the top wall, the combining function defines a gradient that circles back down and around the barrier. This particular situation is one in which a gradient summation technique would create a local minimum in front of the barrier, potentially trapping the robot. Once past the barrier, two new obstacles are encountered. The first falls to the right of the trajectory in such a way that the process constructing the navigation plan should decide to move clockwise around the obstacle. The other falls to the left of the trajectory and therefore the plan indicates that the robot should move c-clockwise past it. The resulting gradient flows around the barrier, up towards the top wall, and then down and to the right, flowing almost equally distant between the final two obstacles. Once past the barrier, the m-NaT representing the barrier has only a small effect on the direction of the gradient (i.e., only when inside its danger zone). Once beyond all the obstacles, the gradient defined by the walls and the trajectory s-NaT is resumed.



Navigation plan for moving up a cluttered hallway and the resulting gradient

Figure 2.22

The second local navigation task required in order to get to Joe's office involves a transition from one hallway into another. This can be accomplished by simply constructing a navigation plan that models the intersection and changes the direction from moving along one hallway to moving along the other. Because of the way that the combining function works, the critical issue in determining when to make the transition from one s-NaT to avoid drastically altering the direction of the gradient local to the robot. One way would be to construct a specialized s-NaT for making turns in such a way that making the transition would simply involve replacing the trajectory s-NaT with the one for making a turn when the intersection has been identified. Another way is to define the trajectory so that at a distance greater than the width of the hallway, the gradient flows perpendicular to the objective trajectory. This way, once the intersection has been identified, the s-NaTs can be swapped and the transition will be relatively smooth. The latter scheme is used in Figure 2.23. In the top left part of the hallway, the gradient field flows to the corner of the intersection between the two hallways. This is a result of the function that defines the immediate navigation objective and the resulting optimal direction which indicates that the m-NaT forming the corner is the primary constraint to be accommodated. Once the transition has been made to the hallway on the right, the resulting gradient is similar to the one for traversing a hallway.

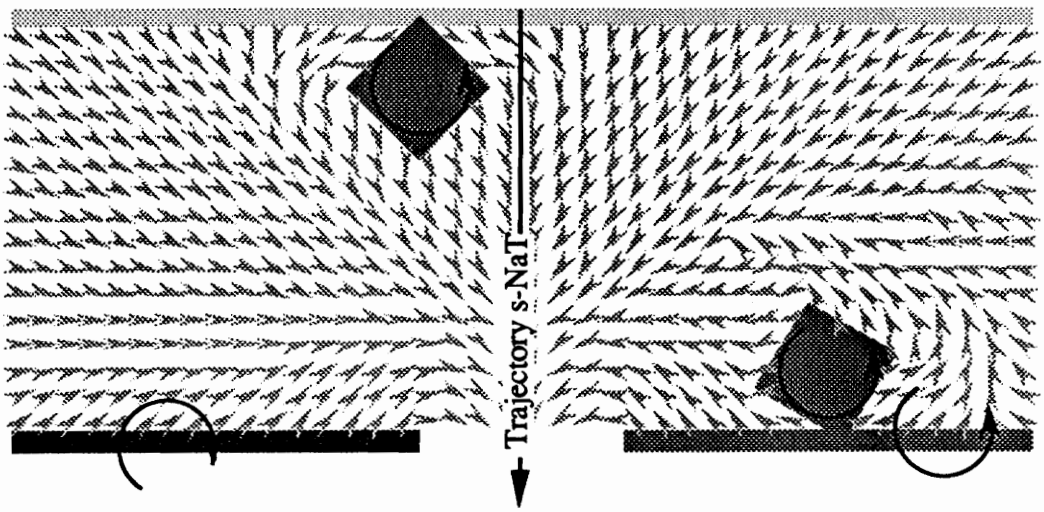


Navigation plan for rounding a corner and the resulting gradient

Figure 2.23

Finally, the doorway to Joe's office comes into view, and a new navigation plan must be constructed in order to direct the robot into the office. This again is simply a matter of changing the s-NaT from one characterizing motion along the hallway to one characterizing motion perpendicular to the hallway and changing the spin on some of the active m-NaTs. Figure 2.24 shows such a situation, in which the navigation task is to pass through an opening in the lower wall. This is accomplished by defining an s-NaT that is directed through the center of the opening and spinning the wall to the right of the opening clockwise (right is determined with respect to the direction of the trajectory) and the wall to the left of the opening c-clockwise. The spin of the top wall in the figure is of no concern,

since it is not in-the-way when attention is restricted to the depicted situation. The spin on the diamond-shaped obstacle to the right of the trajectory is c-clockwise, which is opposite to the intuitive clockwise direction for obstacles to the right of an objective trajectory. The spin on that m-NaT is meant to support the gradient flowing in from the left of the hallway. If the robot entered this scene from the right, then the navigation plan would have spun the m-NaT in the clockwise direction, in order to make the navigation plan consistent with the situation.



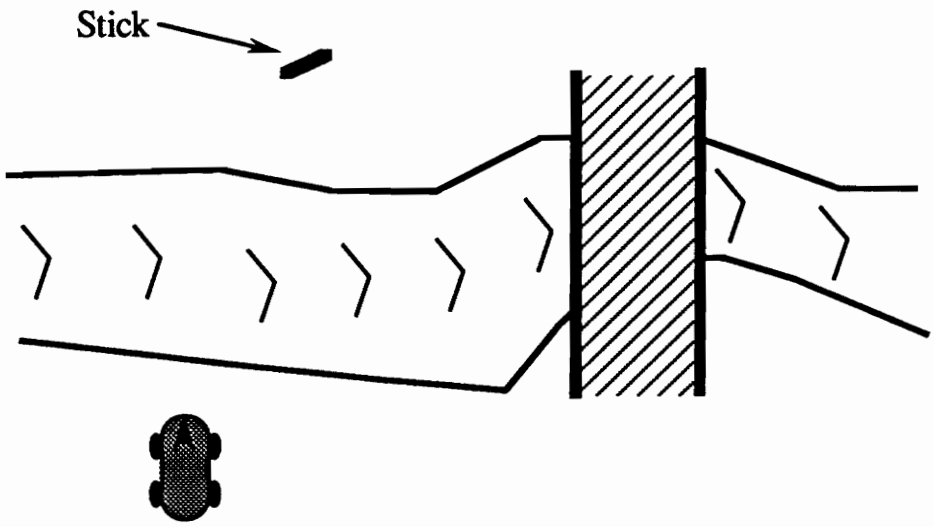
Navigation plan for entering a doorway and the resulting gradient

Figure 2.24

2.4.2. Crossing the Creek

The first chapter posed a navigation problem involving a creek and a bridge where the robot's task is to retrieve the stick from across the creek (see Figure 2.25). The more immediate problem, however, is simply getting to the stick. The problem is used to illustrate the relative strengths and weaknesses of the traditional and reactive approaches to local navigation. Traditional approaches, while goal-directed, are not robust enough in the face of the world's uncertainties to produce the type of smooth behavior that is desired in an autonomous agent. Reactive approaches, on the other hand, are robust in the face of the world's uncertainties; however, they lack the goal-directed behavior required of an autonomous agent. The problem is that neither approach exploits the knowledge available

from an understanding of the relationship between a creek and a bridge. A system that could first capture the essence of the problem, which is to cross the creek via the bridge, can then solve the three simpler navigation problems of getting to the bridge, crossing the bridge, and moving from the bridge to the stick. While this may seem trivial, past navigation research has largely ignored the use of situational knowledge to get a better grasp on the nature of the navigation problem.



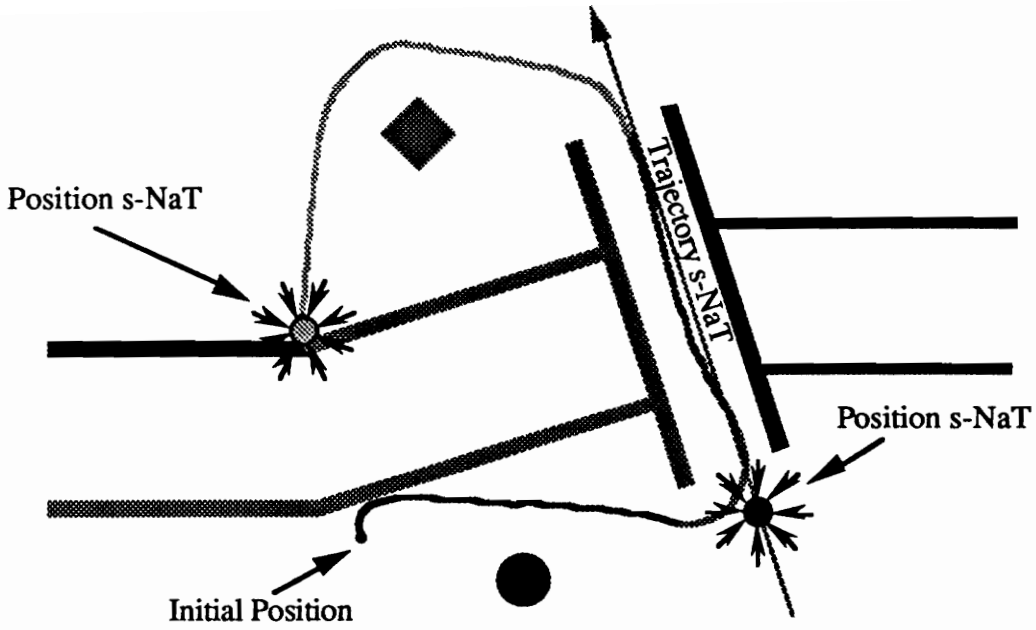
Navigating to the stick

Figure 2.25

The appropriate way to approach this navigation task is to first solve the navigation problem symbolically, without considering the physical interaction between the robot and the world too much. Start by characterizing the nature of the navigation task as simply "needing to get to the other side of the creek". Knowledge of what "crossing a creek" means, with respect to navigation, indicates that creeks effectively split space into two disjoint pieces and that there are basically two techniques for crossing a creek: get wet (assumes that the robot cannot jump) or cross the creek via something that spans the discontinuity. Combining this analysis of the situation with the the fact that there is a known bridge should be enough to direct sensing activity in order to verify that the bridge does, in fact, span the width of the creek. Hence, without ever considering how to drive the motors, the problem of getting across the creek is broken into a sequence of three

separate globally primitive navigation problems: move over to the bridge, cross the bridge, and moving over to the stick.

At this point the navigation problem changes from the intellectual problem of understanding the situation to the physical problem of interacting directly with the environment. Navigation templates are suited for navigation tasks that require the robot to physically interact with the world and when a situational analysis has simplified the general navigation task into a sequence of local navigation tasks. The first navigation task is to move over to the entrance to the bridge. This can easily be accomplished by roughly modeling the bank of the creek as some number of long rectangular m-NaTs, and the rails of the bridge as elongated rectangular m-NaTs (see Figure 2.26). To guide the robot to the entrance of the bridge, a position s-NaT is used to get the robot moving toward the bridge. Once the robot is near the bridge, the position s-NaT is replaced with a trajectory s-NaT which guides the robot down the center of the bridge. Finally, as the robot nears the end of the bridge the, s-NaT is changed once again to a position s-NaT in order to guide the robot over to the stick. Figure 2.26 shows a path followed by a simple gradient following control loop which updated its course based on information provided by the combining function once every ten centimeters (the figure shows an area about ten meters wide). Note that the reason that the path originally moves towards the creek and then proceeds off towards the bridge is a result of two things: the robot is assumed to have a minimum turning radius, and the robot's original orientation has the robot facing the creek. The execution trace is shown in three different shades in order to indicate where the s-NaT was changed in order to direct the actions of the control loop properly.



Trace of a simple gradient following control loop

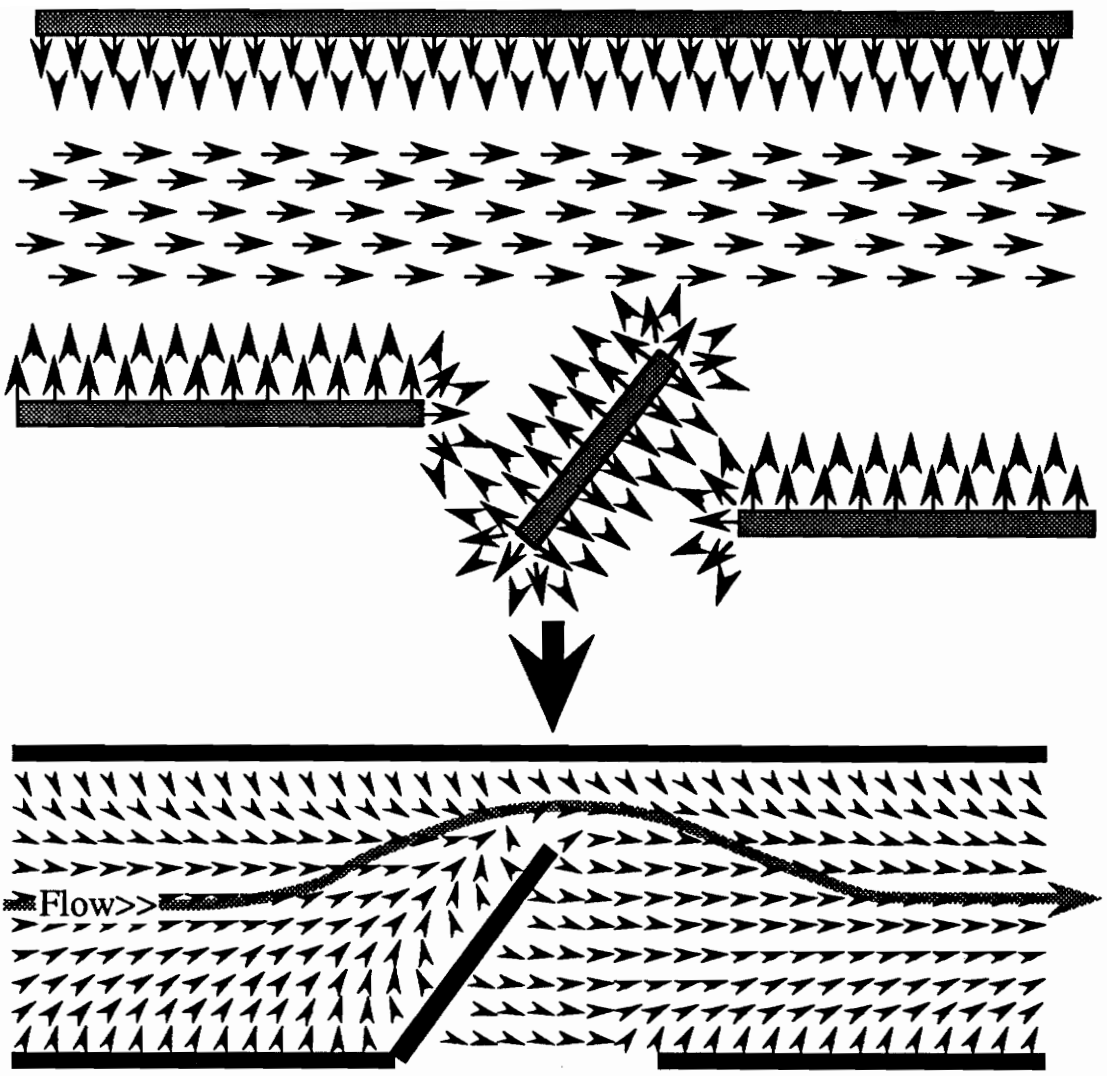
Figure 2.26

2.5. Final Words

There are two basic advantages to the use of *locally* definable gradients for directing a robot's actions:

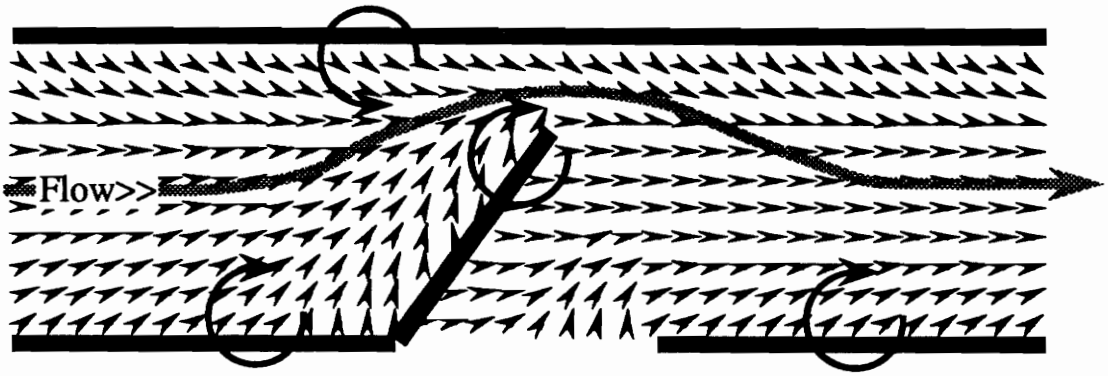
- Because the direction of the gradient at a given position is determined independently of all other positions, the structure upon which the gradient's direction is determined can be modified without affecting subsequent calculation of the gradient's direction.
- Typically, there is very little computation required to determine the direction of the gradient at a given position, thus gradient-based techniques are able to keep pace with the robot's need for real-time guidance.

These advantages provide a compelling argument for the use of gradient-based control of a robot's actions. Previous approaches for creating a locally defined gradient field lack a mechanism for explicitly controlling the flow of the gradient field. As a result, they do not allow the robot to avoid many unfavorable situations. NaTs and the combining function provide a locally defined gradient field. By manipulating the spins of the m-NaTs, the flow of the gradient field is explicitly controlled, thus providing an explicit mechanism for avoiding unfavorable situations. For example, consider a situation where the task is to move along a hallway and avoid a door that has been left open. Using gradient summation techniques a repulsive force is associated with the walls and the door and a gradient is associated with the task of moving along the hallway. Figure 2.27 shows such a situation where the task is to move along the hallway from left to right avoiding the door. The resulting gradient does in fact flow along the hallway from left to right pasting by the door. Creation of a NaT-based navigation plan for the same situation is accomplished by associating an m-NaT with each of the walls and the door and associating an s-NaT with the task. The combining function also generates a gradient that flows along the hallway from left to right past the door (see Figure 2.28). The two approaches are nearly indistinguishable when the configuration of the obstacles in the environment cooperate with the formation of the gradient. Consider how the situation changes when the task is changed from moving from left to right along the hallway to the task of moving from right to left. In this situation, the gradient summation plan is formed by simply reversing the direction of the task gradient. However, because the task gradient is the only active portion of the plan, the resulting gradient does not flow along the hallway as desired; rather, the resulting gradient flows out the door due to the influence of the of the repulsive gradient associated with the door (see Figure 2.29). Formation of a NaT-based navigation plan for the task of moving from right to left along the hallway is easily accomplished by changing the direction of the s-NaT's objective direction and the spins on the m-NaTs associated with the objects in the world. Unlike the gradient summation approach, the NaT-based approach does provide enough situational information to create a gradient which flows from right to left along the hallway past the door (see Figure 2.30).



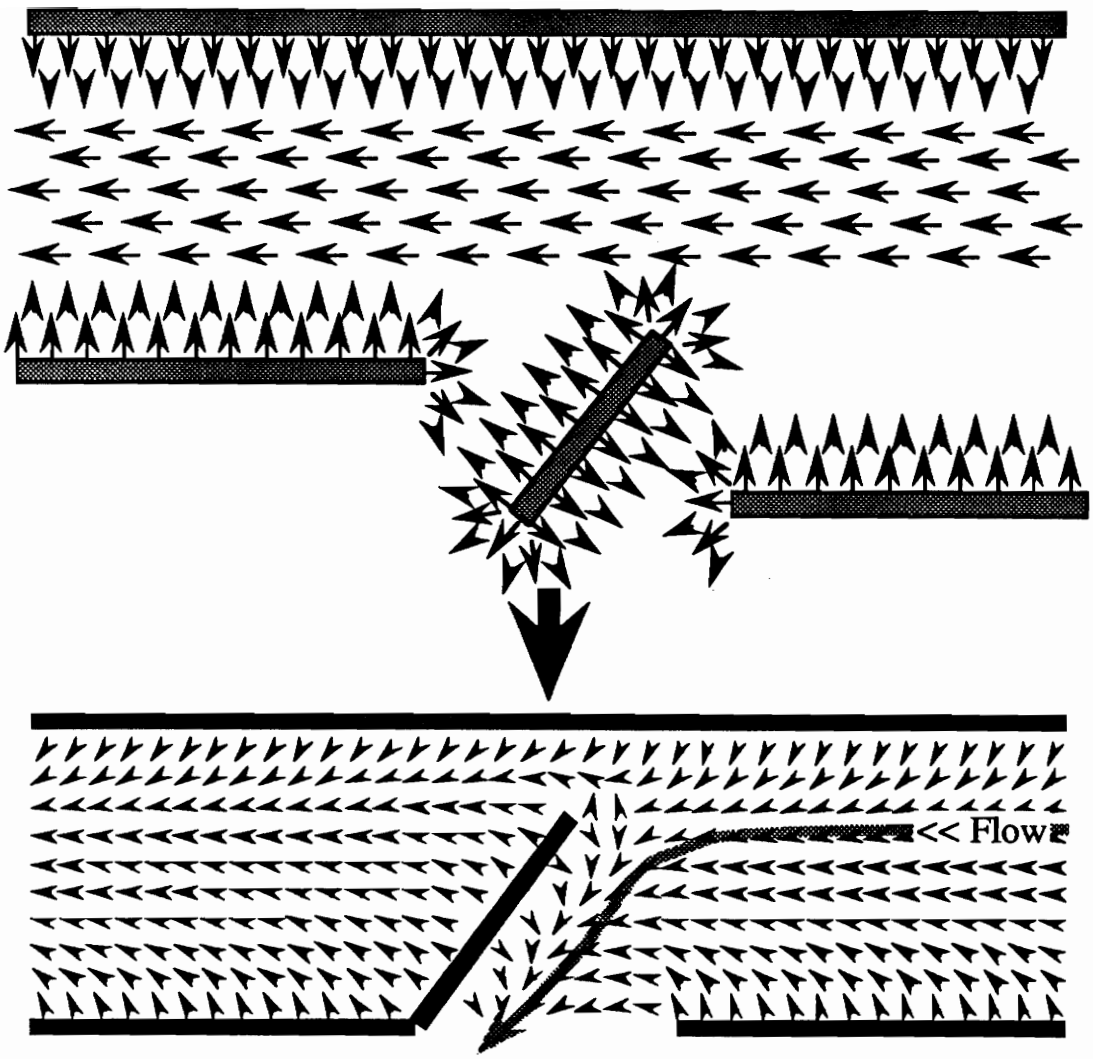
Gradient summation navigation plan for moving to the right along the hall past the door

Figure 2.27



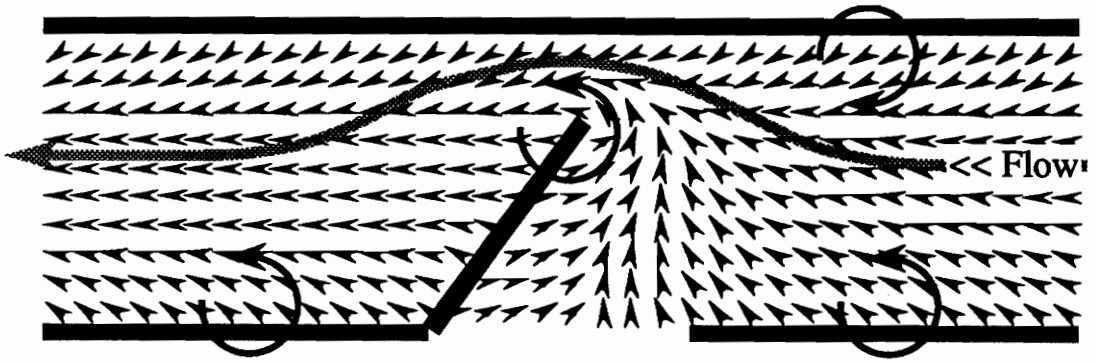
NaT-based navigation plan for moving to the right along the hall past the door

Figure 2.28



Gradient summation navigation plan for moving to the left along the hall past the door

Figure 2.29



NaT-based navigation plan for moving to the left along the hall past the door

Figure 2.30

Chapter Three

3. Navigation Templates II

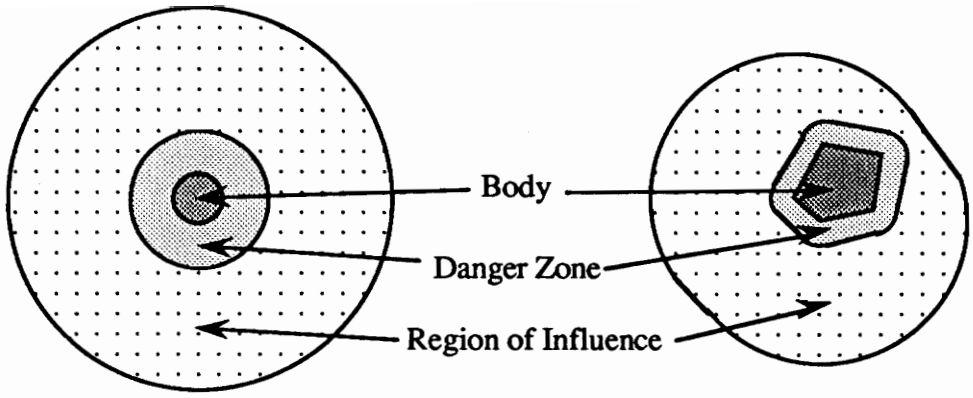
Navigation Templates (NaTs) are used to construct rough navigation plans by characterizing the relationship between obstacles in the environment and the current navigation task. The utility of NaTs is associated with the combining function, which quickly transforms a NaT-based navigation plan into guidance for a robot's low level control loop. There are some extensions, applications, and issues related to the NaTs and the combining function that must still be covered. One such extension provides a mechanism for limiting the range at which a modifier NaT (m-NaT) can influence the combining function. As defined, the combining function will not generate a gradient that flows directly toward an object. However, to accomplish many navigation tasks such activity is often necessary. To alleviate this problem, NaTs and the combining function are altered. Another extension involves supporting a mechanism for determining the spin of the NaT at the time that the combining function is run. Also addressed is the way in which NaTs and the combining function aid in the control of the actions of a robot whose physical dimension cannot be ignored. Finally, a summary of the operations needed to construct a NaT and an outline of the combining function are presented.

3.1. Regions of Influence

As defined, the combining function must consider all the modifier NaTs (or m-NaTs) comprising the current navigation plan to determine the direction of the gradient at a given position. Even if an m-NaT has no influence on the gradient at that position, several geometric operations must be performed on the m-NaT. This raises a performance issue: how to keep the number of NaTs that the combining function must consider at any one time

down to a reasonable number. One technique for limiting the number of m-NaTs that the combining function must consider is to place a strict limit on the number of of m-NaTs that can be used to construct a navigation plan. Such a restriction has the desirable characteristic of placing an upper limit on the amount of computation required to determine the direction of the gradient at any position. However, unless the upper limit is sufficiently large, such an approach places a considerable burden on the system maintaining the navigation plan by requiring it to limit its plans to only those m-NaTs that are relevant to the situation at hand. To ease this burden, a computationally efficient mechanism to limit the number of relevant m-NaTs is needed. This need is addressed by associating a region of influence with each m-NaT. The region of influence places a limit on the range at which an m-NaT is relevant to a particular situation. Hence, while it is ultimately the responsibility of the system maintaining the navigation plan to limit the number of m-NaTs that are active at any one time, an m-NaT's region of influence helps to reduce the number of m-NaTs that must be combined at a given position.

To support the region of influence, each m-NaT must provide a Boolean operation which determines whether a given position resides inside or outside the m-NaT's region of influence (see Figure 3.1). While there is no restriction to the definition of an m-NaT's region of influence, a simple definition might include those points that are within some specified distance from the body of the m-NaT. In order to incorporate this new feature into the combining function, a simple preprocessing step is added. The preprocessing step eliminates from further processing those m-NaTs whose region of influence does not include the position for which the combining function is being run. Consider a hallway scenario, in which four NaTs are needed to construct the navigation plan: two m-NaTs to represent the walls, one m-NaT to represent an obstacle located along the right side of the hallway, and an s-NaT which guides the robot up the center of the hallway. When the combining function is applied to these four NaTs without the added preprocessing step, the interaction between the three m-NaTs results in a gradient field that flows up the hallway and around the obstacle. The resulting gradient would be nearly indistinguishable if the effects of the m-NaT representing the obstacle are ignored until the obstacle was within 5 or 10 meters. If used properly, the gradients produced with the introduction of the region of influence are almost identical to those produced without limiting the range of an m-NaT's effectiveness, but the computational load of the combining function is reduced.

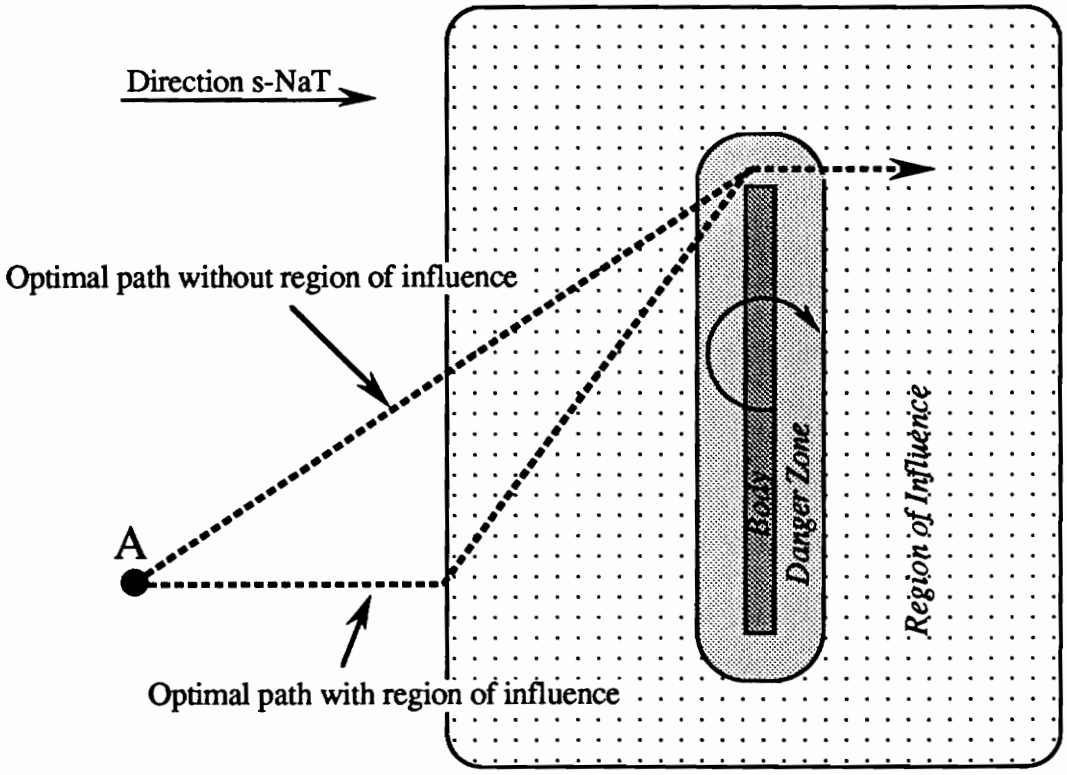


An m-NaT's region of influence

Figure 3.1

While adding a region of influence to each m-NaT facilitates the computation of the gradient at a given position, the reduction in computational requirements comes at the cost of placing more responsibility on the system maintaining the navigation plan. Consider the situation depicted in Figure 3.2, in which there is a single m-NaT and a direction s-NaT configured as shown. The optimal path proceeds directly from location A to the corner of the m-NaT and then continues in the direction of the s-NaT's gradient. However, when the m-NaT's region of influence is considered, the path from location A proceeds first in the direction of the s-NaT's gradient until the m-NaT's region of influence is encountered. Next the path leads to the corner of the m-NaT and then in the direction of the s-NaT's gradient again. If the extent of the m-NaT in Figure 3.2 is relatively small in comparison to the distance traveled, then the inefficiency resulting from the addition of the region of influence would be of little consequence. However, if the extent of the m-NaT is considerable (e.g., representing an extended fence line), then the inefficiency might be significant. In the latter case there are two ways in which the inefficiency can be addressed. The first would grow the m-NaT's region of influence to the point that the inefficiency is reduced to an acceptable level. The more appropriate solution is to select a new s-NaT that properly considers the implications of this constraint with respect to the overall navigation task. Rather than relying on the combining function to direct the robot around the corner of the m-NaT, the current navigation task of heading in an easterly direction should be broken down into two sub-tasks: first moving the robot to the corner of the m-NaT, and then simply restoring of the original navigation task. This example points

out that, at the level of the combining function, there is no explicit knowledge of the situation and it is therefore the responsibility of the system maintaining the navigation plan to define an m-NaT's region of influence appropriate to the current navigational task.



The effects of an m-NaT's region of influence on the optimal path

Figure 3.2

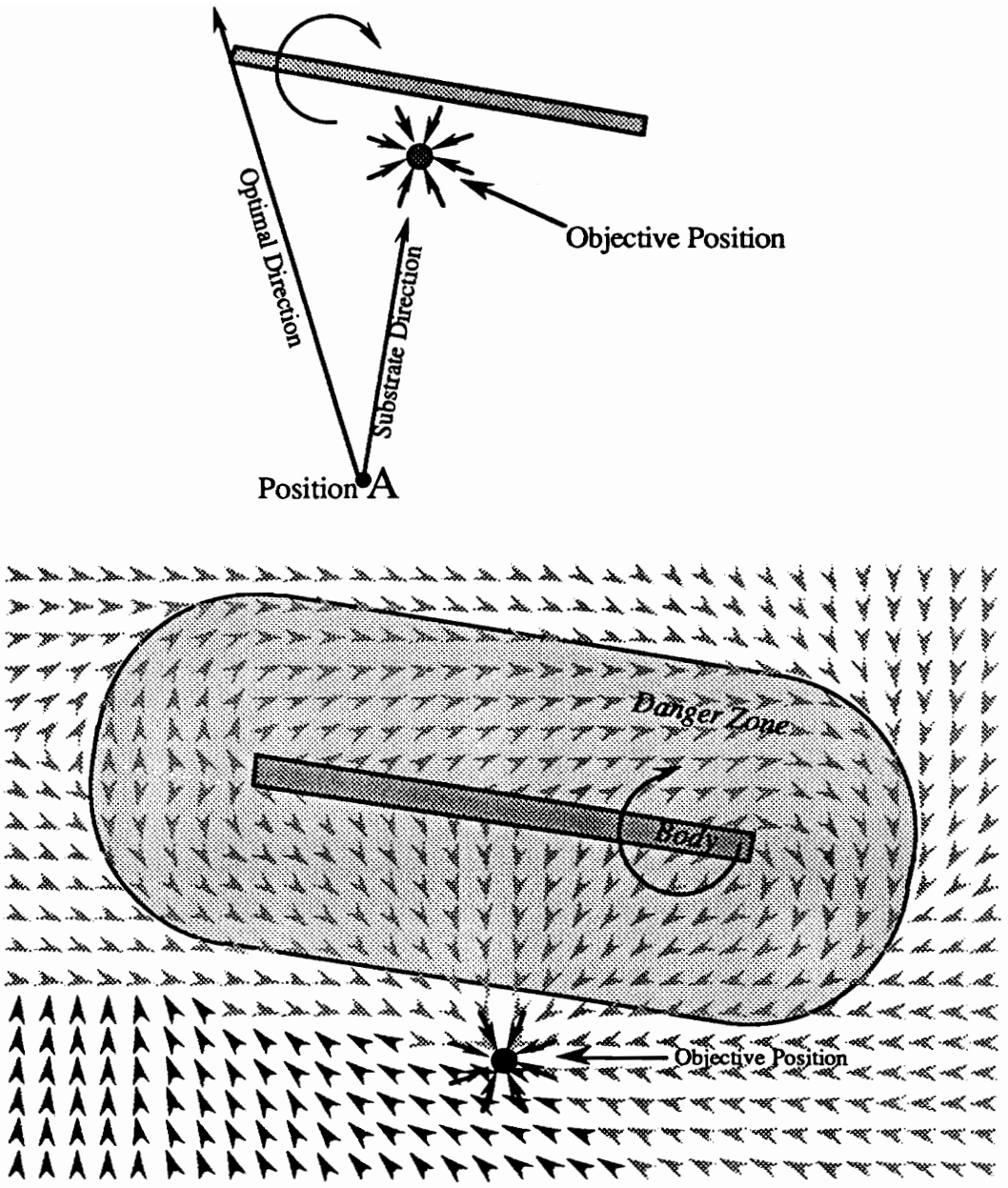
3.2. Approaching m-NaTs

Consider the navigation task of moving to a specific location. There is a subtle difference between such a navigation task and that requiring the robot to move in a direction or along a trajectory. The difference between the two is that when the navigation task is to move in a direction or along a trajectory, objects encountered along the way are obstacles to be avoided; whereas, when moving to a particular location, objects must sometimes be

approached. For example, consider the problem of moving to a position next to an object. This is a common occurrence, as ticket counters, elevator buttons and the like are all located in or on walls. In such situations, the object being approached is not an obstacle to be avoided, as it would be if the navigation task were to move in a particular direction. Rather, the object can be ignored entirely unless the position to be attained lies on the other side of the object or requires the robot to travel along the length of the object.

The NaTs and the combining function, as defined, cannot produce a gradient field that will flow smoothly to a location in front of an m-NaT. Consider the situation depicted in the upper portion of Figure 3.3, in which there is an m-NaT with a clockwise spin and a position s-NaT. After applying the combining function at position A, in this situation, the m-NaT is considered to be in-the-way and the resulting optimal direction points to the m-NaT's clockwise corner rather than toward the objective position. If the robot followed the gradient, moving off toward the m-NaT's corner, eventually the robot would enter a portion of space in which the m-NaT is not in-the-way, allowing the combining function to produce a gradient that flows back toward the objective position (see lower portion of Figure 3.3). The portion of the gradient indicated in the Figure by the darker arrows is directed to the left corner of the m-NaT in an attempt to satisfy the local constraint of moving c-clockwise around the m-NaT. In many cases this will keep the robot from ever accomplishing the navigation task. Therefore, a modification is needed in order to allow the gradient field to flow smoothly up to an objective position located next to an m-NaT.

The inability to approach an m-NaT is a result of the fact that the combining function is strictly a local computation, taking no account of the nature of the navigation task being accomplished. Because the s-NaT is the only element of a NaT-based navigation plan containing any information about the navigation task, one solution to the problem is to allow the s-NaT to play a larger role in the combining process by allowing it to make decisions as to the relevance of an m-NaT. Thus, in the case of the position s-NaT, the s-NaT must limit the effect of the active m-NaTs so that the objective position can be approached, while ensuring that the combining function is otherwise unaffected.

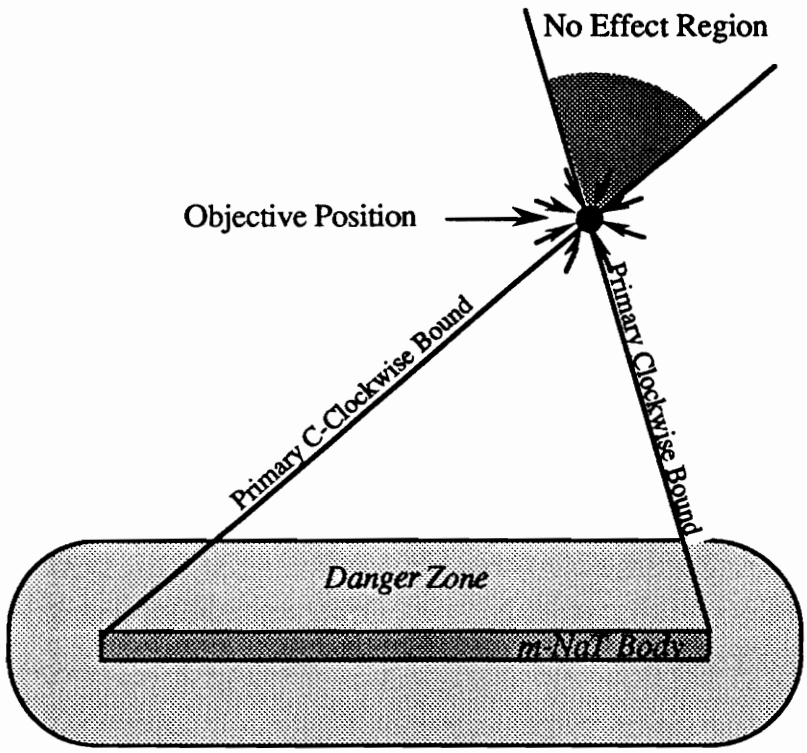


Attempting to reach an objective position

Figure 3.3

The root of this problem lies in the fact that an m-NaT's primary clockwise and c-clockwise bounds are used to identify the immediate navigation objective at a given position. The result is that if the objective position of the position s-NaT lies within the

clockwise range of angles between the c-clockwise primary bound and the clockwise primary bound, then the optimal direction cannot point towards the objective position. This has the result that the gradient will not flow directly towards the objective position when there is an m-NaT on the other side of the objective position as seen from the position where the combining function is being applied. In other words, if the objective position as seen from a particular position lies inside the clockwise angular range between an m-NaT's primary clockwise and c-clockwise bounds, then the objective position cannot be approached directly from that position. To alleviate this situation, the primary clockwise and c-clockwise bounds of each m-NaT as seen from the objective position are computed. These angles form a region of space inside which the effects of the m-NaT are ignored when calculating the direction of the gradient (see Figure 3.4). This allows the optimal direction within this region to be directed at the objective position.



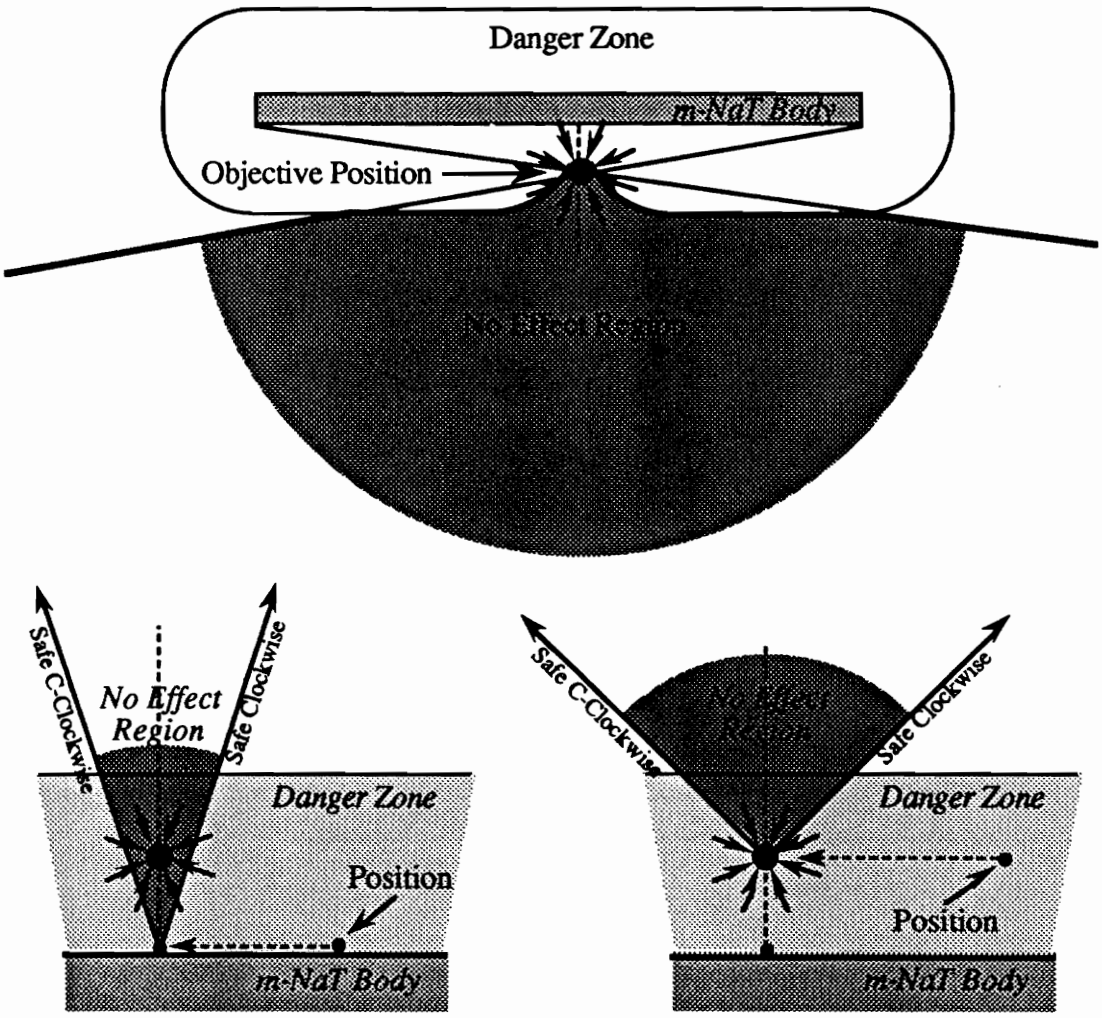
Limiting the effects of an m-NaT in the presence of an objective position s-NaT

Figure 3.4

This solution works when the position at which the gradient is being calculated lies outside of the m-NaT's danger zone. However, consider a situation in which the objective position is very close to a wall being represented by a long rectangular m-NaT. If the objective position were in fact very close to the wall, then the primary bounds with respect to the objective position would run parallel to the side of the m-NaT, cutting space into two half planes. This has the undesirable property that the m-NaT would be ignored in an entire half-plane, and a side of the m-NaT would run along the plane separation, creating a situation in which the robot could easily bump into an obstacle. Thus, a more robust solution is needed when the objective position lies inside of an m-NaT's danger zone. Figure 3.5 indicates the method used to handle this situation, in which the previously described scheme is used to limit the m-NaT's effectiveness for positions that are outside of the m-NaT's danger zone. Inside the m-NaT's danger zone, the safe clockwise and c-clockwise bounds are used to define the region in which the m-NaT is not effective. A line which passes through the objective position and contacts the body of the m-NaT at the point that is closest to the objective position is used as the basis for defining the region in which the m-NaT is ignored. For positions within the m-NaT's danger zone, the portion of the danger zone in which the m-NaT is ignored is defined as follows: The clockwise range of angles between the safe c-clockwise and clockwise bounds of the m-NaT defined at a location on the aforementioned line passing through the objective position that is at the same distance from the m-NaT's body as the given position. For example, if the position is in contact with the m-NaT's body, then the ineffective region is defined as the clockwise range of angles between the safe c-clockwise and clockwise bounds defined at the position on the m-NaT's body that is closest to the objective position (see lower left portion of Figure 3.5). If the position is at the same distance from the m-NaT's body as the objective position, then the region in which the m-NaT has no effect is defined as the clockwise range of angles between the safe c-clockwise and clockwise bounds as defined at the objective position (see the lower right portion of Figure 3.5).

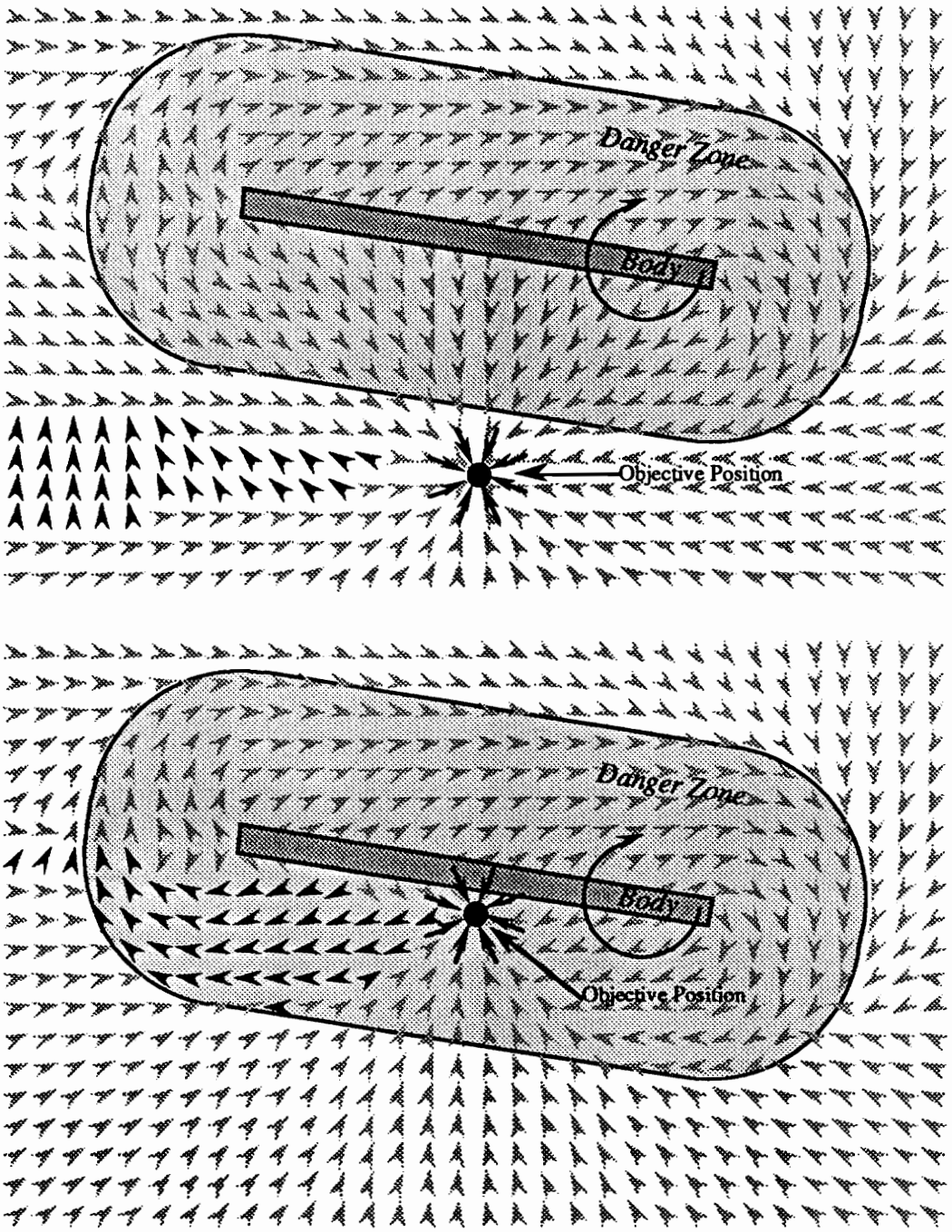
Figure 3.6 shows an example of a gradient created using position s-NaTs augmented with the ability to limit the effect of the m-NaTs being combined. The upper portion of the figure depicts a situation in which the objective position lies outside of the m-NaT's danger zone, while in the lower figure the objective position lies inside the m-NaT's danger zone. Using this modified technique, the resulting gradient field is much closer to the type of gradient field required to approach an m-NaT. However, there is still a minor problem,

indicated by the small region of dark arrows in the upper figure directed toward the left end of the m-NaT. This is a result of the fact that the navigation plan indicates that the m-NaT should be negotiated in a clockwise direction, while to reach the objective position the m-NaT must be negotiated in a c-clockwise fashion. This problem is partially solved by reversing the m-NaT's spin; however, this will only move the problem to the other side of the objective position. The key to a more robust solution lies in the fact that the m-NaT's spin is irrelevant when the m-NaT is being approached. If the spin of the m-NaT being approached is allowed to float using a default scheme, then the problem with the defined schemes for approaching an m-NaT can be eliminated.



Case when the objective position lies inside an m-NaT's danger zone

Figure 3.5



Allowing an s-NaT to make contextual relevancy decisions about the m-NaTs

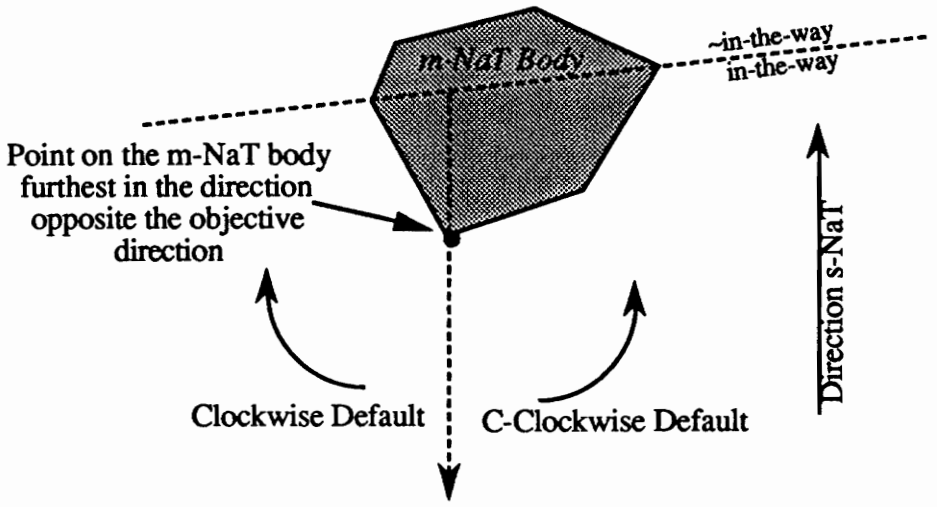
Figure 3.6

3.3. Spinning NaTs Dynamically

One of the fundamental principals behind NaT-based navigation plans and the combining function is the notion that there are two ways in which the robot can move past an obstacle. This is the reason a spin is associated with each m-NaT. The spin of each m-NaT is determined by the system which maintains the navigation plan, and may at any time be changed by that system. Occasionally, however, the best way around an obstacle cannot be determined until m-NaT combining time, and at other times it simply does not matter in which way the robot proceed around the m-NaT. In such situations, it is appropriate to allow an m-NaT's spin to be determined at combining time. To support this, there are three types of m-NaTs: *spun*, *unspun* and *dynamically-spun*. *Spun* m-NaTs are those whose spin is set explicitly by the system managing the navigation plan, M-NaTs discussed up until this point have been of this type. *Unspun* m-NaTs are assigned no spin by the system managing the navigation plan; instead, the spin of an unspun m-NaT is determined the first time that the m-NaT's spin is needed in order to combine it with the other m-NaTs. However, once the spin of an *unspun* m-NaT is determined it becomes fixed, changing the m-NaT to a *spun* m-NaT. *Dynamically-spun* (or *d-spun*) m-NaTs determine their spin each time it is needed by combining function. Hence, a *d-spun* m-NaT can potentially spin in more than one direction during the time that it is part of a navigation plan.

The scheme used to assign a spin to an *unspun* m-NaT is a function of the current s-NaT, the characteristics of the m-NaT, and the position that the combining function is operating on. For example, a scheme to determine the spin of an m-NaT in the presence of a direction s-NaT is shown in Figure 3.7. The scheme splits space into four regions. One region represents all positions that are outside of the m-NaT's region of influence. If the position lies in this region, then the m-NaT's spin is of no importance and therefore not determined. Another region represents the portion of space in which the m-NaT is not in-the-way. If the position at which the combining function is being employed lies in the space in which the m-NaT is not in-the-way, then the spin of the m-NaT is of no consequence and therefore not determined. The remaining two regions split that region of

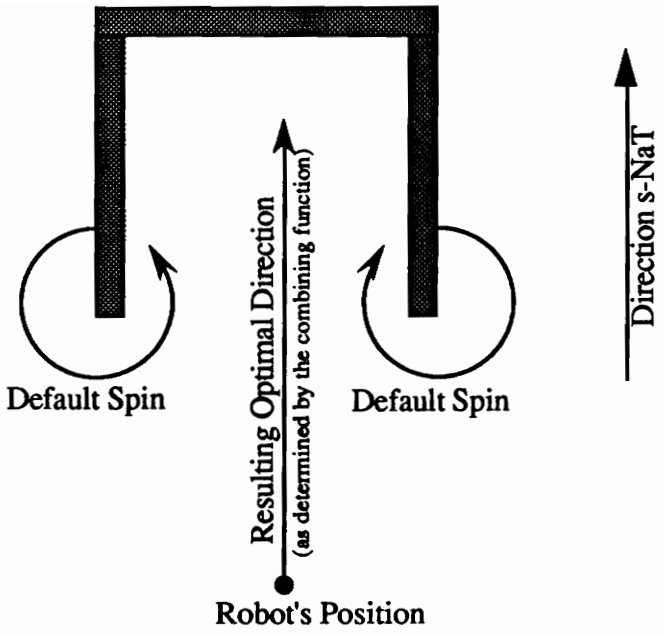
space in which the m-NaT is considered to be both in-the-way and within the m-NaT's region of influence; one represents the space of positions in which the m-NaT's default spin is clockwise and the other represents the space in which the default spin is c-clockwise. The line which separates the half plane into two regions is parallel to the objective direction and runs through the point on the body of the m-NaT furthest in the opposite direction of the objective direction (see Figure 3.7 for an example). In the case where this defines a line rather than a single point, a dividing line passes through the midpoint of that line.



Scheme for determining an m-NaT's default spin in the presence of a direction s-NaT

Figure 3.7

A system designing a navigation plan should not allow the spin of an m-NaT to be automatically determined in situations where an m-NaT's spin is of consequence. Consider the situation depicted in Figure 3.8, in which three m-NaTs are configured to form a dead end with respect to a direction s-NaT. If the m-NaTs used to model the right and left walls forming the dead end are *unspun*, then it is possible for the robot to become trapped and unable to continue moving in the direction indicated by the direction s-NaT. This would arise if the robot encounters the situation in a way that results in the default spins of the right and left walls forcing the robot into the dead end. Therefore, *unspun* m-NaTs should be used only in those situations in which it is believed possible to pass an obstacle in either direction and still accomplish the navigation task.



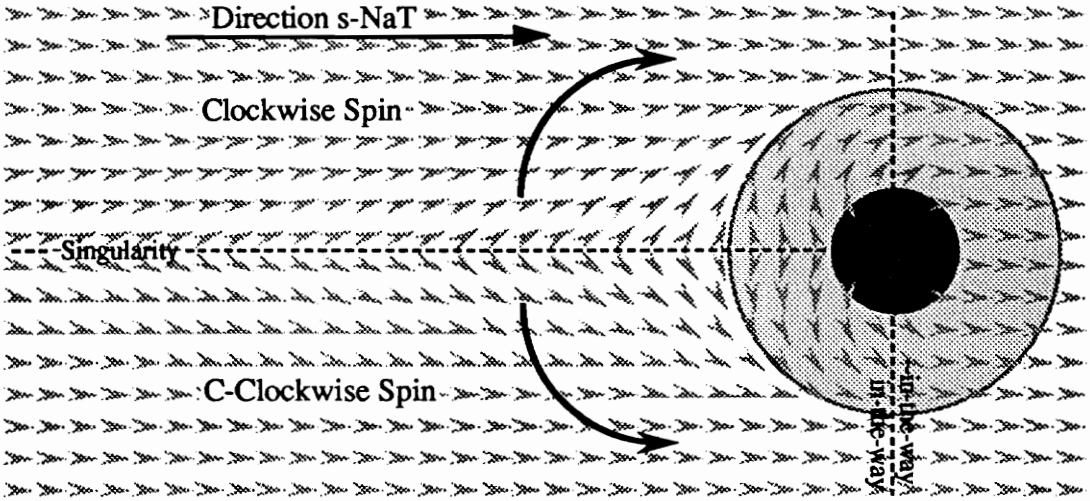
Care must be taken when using *d-spun* m-NaTs

Figure 3.8

The spin of a *d-spun* m-NaT is determined each time the m-NaT's spin is needed by the combining function. *D-spun* m-NaTs are particularly useful when there are forces in the environment that might cause an obstacle to change locations or cause the robot to follow a less-than-ideal trajectory through the gradient field. Figure 3.9 shows an example of a gradient field resulting from a *d-spun* circular m-NaT in the presence of a direction s-NaT. The singularity in the gradient field runs along the line which splits the gradient from a clockwise flow to a c-clockwise flow. Because the m-NaT is *d-spun*, this m-NaT type must be used carefully. Unlike *spun* or *unspun* m-NaTs, the use of *d-spun* m-NaTs may result in the robot oscillating between going clockwise and c-clockwise around the m-NaT.

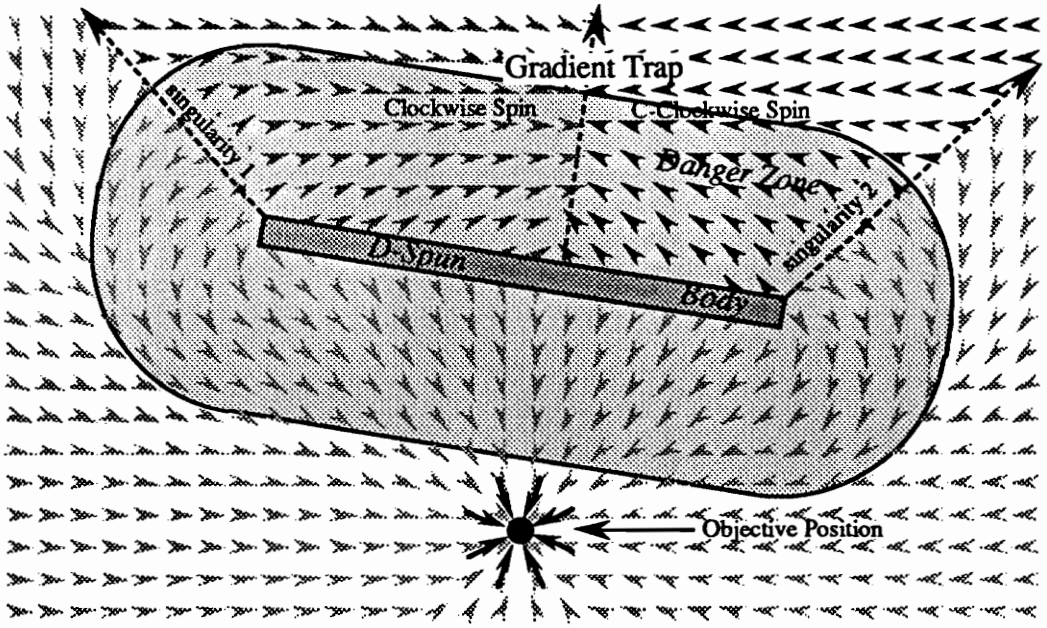
When using d-spun m-NaTs it is important to ensure that the scheme used to determine the default spin of an m-NaT will not result in the formation of a gradient trap. For example, consider what happens if the scheme defined above for determining the default spin of an m-NaT is used to determine the m-NaT's spin when a position s-NaT is active (see Figure 3.10). The trap is a result of the fact that the direction of the substrate in the upper right portion of the figure points in a direction indicating that the default spin is c-

clockwise around the m-NaT, while in the upper left portion of the figure the resulting default is clockwise around the m-NaT. The overall effect is a gradient which flows into a line on the far side of the obstacle from the objective position. Such a situation would not occur if the m-NaT were either *spun* or *unspun*, because once the spin is set the gradient would direct the robot to move around the obstacle in a consistent fashion.



D-spun m-NaT in the presence of a direction s-NaT

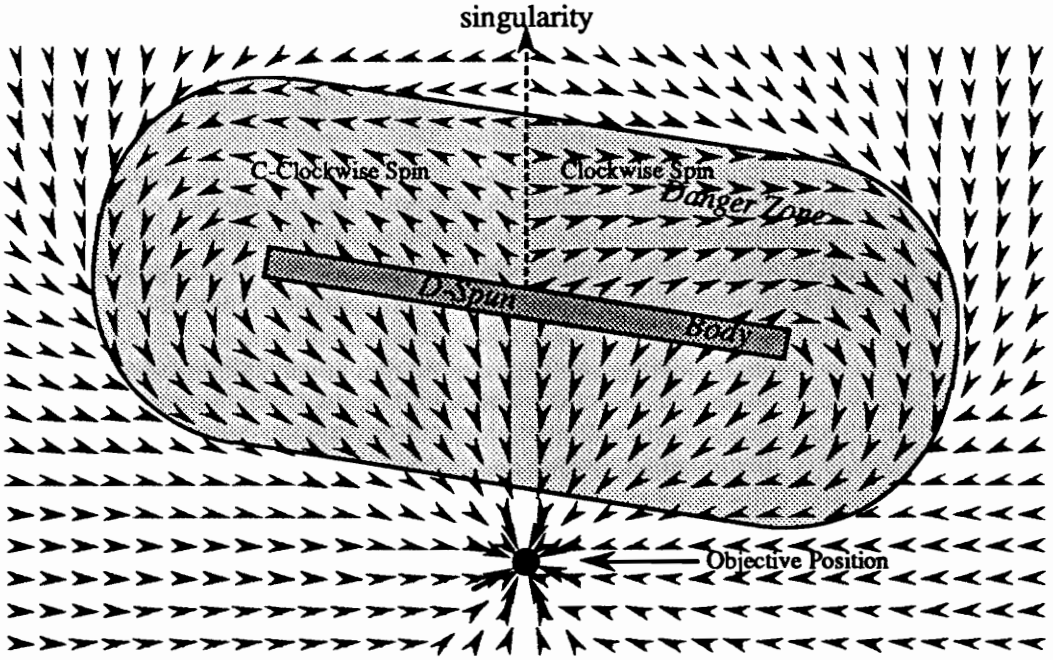
Figure 3.9



Existence of a local minimum

Figure 3.10

To eliminate the existence of the gradient trap, a different technique for determining the default spin of a *d-spun* m-NaT in the presence of a position s-NaT is required. Figure 3.11 shows the results of a scheme which uses the distance around the m-NaT to reach the objective position to determine the m-NaT's default spin (i.e., spin in the direction that minimizes the distance to the objective position). This scheme requires each m-NaT to be capable of determining which direction around its body defines the shortest distance between a given position and an objective position. This scheme works well for determining the default spin of both *unspun* and *d-spun* m-NaTs. The example in the figure is similar to those introduced in the last section, in which the problem of approaching an m-NaT is addressed. Furthermore, the arrows in the aforementioned problem region all point towards the objective position. The gradient is able to flow towards the objective position from every location because, when an m-NaT is being approached, its spin simply does not matter as the object is not an obstacle to be negotiated.



A *d-spun* m-NaT in the presence of a position s-NaT

Figure 3.11

3.4. Real Robots

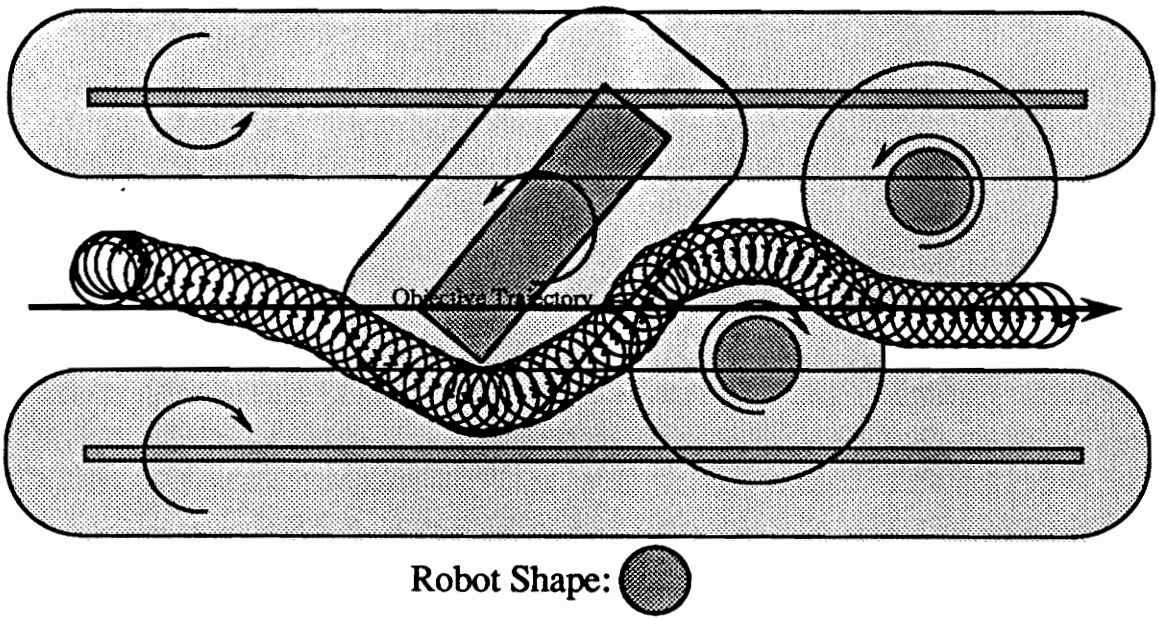
It has been shown that NaTs can be used to construct navigation plans and how the combining function transforms the NaTs into a gradient. However, little has been mentioned about the way in which NaTs and the combining function can aid in the process of controlling the actions of a robot. This section presents some simple control loops which employ NaTs and the combining function to control the actions of a robot with physical extent.

A number of schemes have been developed for determining the way to control a robot's action. By far the most common approach adopted in the literature involves the use of configuration space {Rosen70, Udupa77, Lozano-Perez79}. Essentially, configuration space trades the general problem of moving a robot with physical dimension through space for the problem of moving a point through the space of possible robot configurations. One way that NaTs can be used to control the actions of a robot is by "growing" each of the m-NaTs' bodies to its configuration space equivalent. By growing the m-NaTs' bodies to account for the physical extent of the robot, the robot's actions can be controlled based solely on the robot's location and orientation. Such an approach works well if the robot is circular or is not allowed to rotate. However, when the robot is non-circular and allowed to rotate freely, the scheme of growing the obstacles to produce the CSpace breaks down and the dimensionality of the CSpace increases beyond two dimensions. This results in the notion of an m-NaT's spin loosing all meaning due to the fact that there are no obstacles, only viable multi-dimensional configurations that the robot can assume.

As defined, the m-NaTs' danger zones already provide partial provision for accommodating the robot's dimensions by causing the gradient field to flow around the m-NaTs at a designated distance. In the case of a circular robot, the danger zones can be used in very much the same way as configuration space, allowing the robot's physical extent to be ignored. This is accomplished by ensuring that the danger zone of all the m-NaTs extends away from an m-NaT's body to a distance equal to at least the radius of the robot. In general, the danger zone should extend beyond the robot's radius to provide a margin of error, but not so far as to make the resulting gradient flow too widely around obstacles.

Consider the situation depicted in upper portion of Figure 3.12, in which the earlier hallway example is revisited. In this instance, the navigation plan and the combining function are being used to control the action of a circular robot. The robot is equipped with synchro-drive, allowing it to turn in place as well as follow circular arcs of an arbitrary radius. The danger zones on the m-NaTs are shown by the surrounding shaded regions and were chosen to be slightly larger than the diameter of the robot. The hallway in the figure is two meters wide; the robot is about 0.4 meters in diameter. The algorithm used to control the actions of this robot set its maximum linear speed to be 0.50 meters/second and its maximum angular velocity to be 0.4 radians/second. The control loop iterates once every 0.2 seconds. Each time through the loop, the combining function is run to determine the direction of the gradient at the robot's location. The local direction of the gradient is combined with the robot's current state to determine the linear and angular accelerations to be applied by the robot's actuators until the next iteration through the control loop. In the example shown, the linear and angular velocities are controlled to maintain an inversely proportional relationship with each other (see the lower portion of Figure 3.12). This way, when the robot is out of alignment with the gradient, the robot is slowed in order to turn more quickly to become realigned. When in alignment with the gradient, the robot stops turning and accelerates towards its maximum linear velocity.

If the same control scheme is employed to control the actions of a rectangularly shaped synchro-drive robot, the results are not quite so desirable (see Figure 3.13). In this situation, the robot's center follows nearly the same path as the circular robot; however, its overall trajectory is much rougher and results in a collision with one of the obstacles in the environment. This is a result of the fact that, unlike that of a circular robot, rotation of a rectangular robot changes its physical relationship with the environment. This illustrates that controlling the actions of more complicated robots using the information provided by the combining function cannot be based simply on the direction of the gradient at a single location.

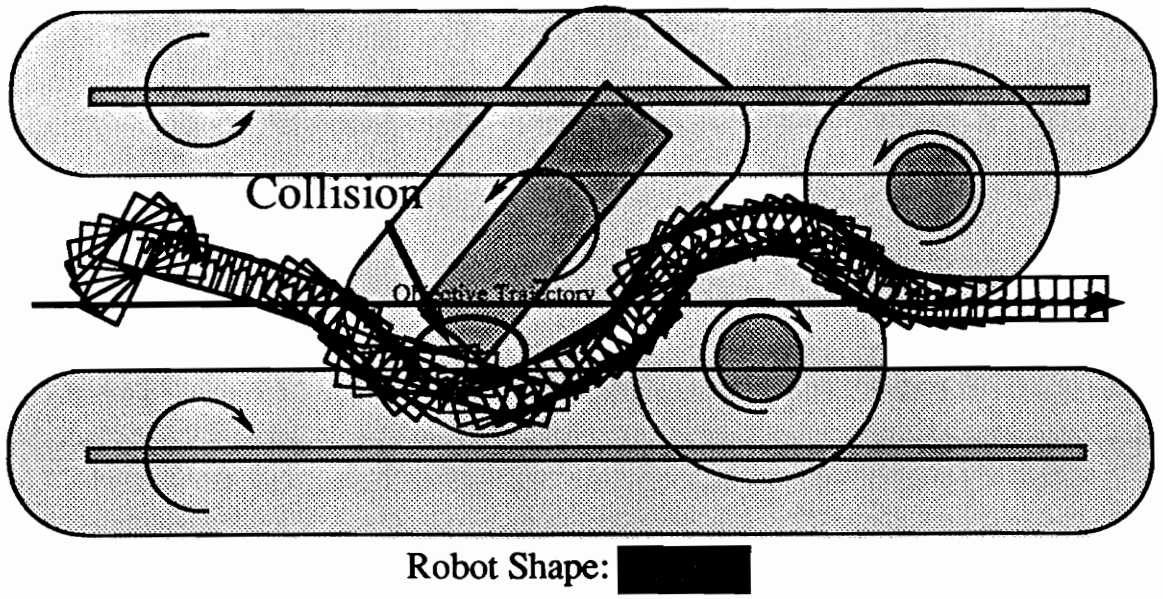


Code used to determine the linear and angular accelerations for the robot to apply for the next dtime seconds given the angular disparity between the robot's current orientation and the preferred direction of travel

```
(define (crbot>incremental-accelerations self dtime delta-angle)
  (let ( (objective-linv
          (if (< (crbot-max-ang-disparity self) (abs delta-angle))
              0.0
              (* (crbot-max-linv self)
                 (/ (- (crbot-max-ang-disparity self)
                       (abs delta-angle))
                    (crbot-max-ang-disparity self))))) )
    (let ( (lina (/ (- objective-linv (crbot-linv self)) dtime))
           (anga (/ (* 2.0 (- delta-angle
                              (* dtime (crbot-angv self))))
                    (* dtime dtime))) )
      (values (min lina (crbot-max-lina self))
              (max (- (crbot-max-anga self))
                   (min (crbot-max-anga self) anga)))))))
```

Simulation trace for a circular robot moving down a cluttered hallway

Figure 3.12



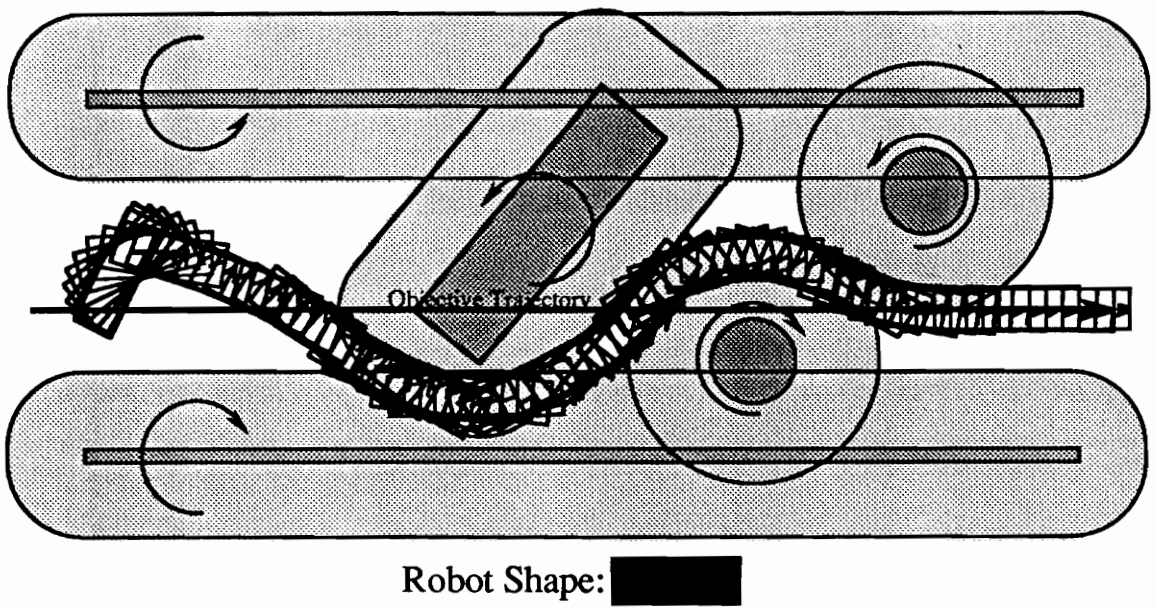
Simulation trace of a simple control scheme used to control the actions of a rectangular robot

Figure 3.13

A way to address this problem is to use the value of the gradient field at a number of locations along the boundary of the robot as input to a slightly more complicated control algorithm. To support this approach, the combining function returns the distance and direction to the body of the m-NaT closest to the given position[1] in addition to the direction of the gradient at the given position. Figure 3.14 shows the results of applying such a scheme to the control of a rectangular robot where, in addition to the gradient at the center of the robot, the directions of the gradient at the robot's four corners were also calculated each time through the control loop. The direction of the gradient and the distance to the nearest obstacle at the four corners were used to bias the direction of the gradient at the center of the robot. The simulation trace shown in Figure 3.14 was produced with an identical control loop as the one which produced the trace shown in Figure 3.13, except for the preprocessing step which biased the gradient direction each time through the control

[1]Note that the combining function has all of this information available so no significant computation is required to return this information.

loop. While this is by no means a general solution for controlling the actions of an arbitrarily shaped robot, it does show that NaTs and the combining function provide substantial aid for controlling the actions of such non-trivial robots operating in reasonably complex domains.



Simulation trace of a rectangular robot negotiating the hallway example

Figure 3.14

3.5. Summarizing NaTs

Navigation Templates (or NaTs) come in two basic types. The first type of NaT is the substrate NaT (or s-NaT), which is used to characterize the basic nature of the navigation task on the order of "go in that direction", "move alone this path", or "move over to that location". Each s-NaT provides a gradient field indicating at each position in space the preferred direction of travel in service of the associated navigation task, assuming that there are no environmental constraints that could impede the robot's progress. The second type of NaT is the modifier NaT (or m-NaT), which is used to model the relationship between

obstacles in the environment and the navigation plan. Each m-NaT must provide a number of geometric operations in order for the combining function to consider the relative effects of the m-NaTs and determine the preferred direction of travel from a given position. Figure 3.15 shows the basic structure of an m-NaT. Each m-NaT has or provides each of the following:

- A body. This is a convex geometric object used to represent the physical object in the environment represented by the m-NaT in the navigation plan.
- A spin. This, in part, designates the relationship between the m-NaT and the navigation task and forms the basis for the navigation plan. The spin is either clockwise or c-clockwise, and indicates that it is the intention of the navigation plan that the robot move with respect to the m-NaT's body as specified by the spin.
- A safe distance. This is used to define the m-NaT's danger zone (i.e., those positions that are within the the safe distance away from the m-NaT's body).
- A region of influence. The combining function will attempt to accommodate the navigational constraints imposed by the m-NaT for any position that resides within this region.
- Functions to compute the right and left most extremes of the m-NaT's body with respect to a given direction. These points define a line cutting space into two half-planes. In one half plane the m-NaT is considered to be in-the-way, and in the other half-plane it is not. The combining function treats those m-NaTs that are in-the-way differently from those that are not in-the-way.
- The clockwise and c-clockwise most visible points on the body of the m-NaT with respect to a given position. These define the angle of occlusion produced by the m-NaT's body as seen from the given position. These values are used by the combining function to define the clockwise and c-clockwise primary bounds and to determine the distance between a given position and the clockwise and c-clockwise most visible extent of the m-NaT's body.
- A function to compute the closest point of the body of the m-NaT from a given position. This is used by the combining function to define two quantities: the

contact distance, which is the distance between the m-NaT's body and the given position, and the secondary bound, which is the ray from the closest point on the body of the m-NaT through the position at which the combining function is being run. When the contact distance is less than or equal to the safe distance, the position falls within the m-NaT's danger zone. The secondary bound is used with the primary bounds to define angle ranges from the given position that are relatively clockwise and c-clockwise with respect to the m-NaT's body.

- The right-most and left-most visible points of the m-NaT's danger zone as seen from any given position outside the m-NaT's danger zone. These are used by the combining function to determine the way the resulting gradient should flow around the m-NaT's body.

In addition, m-NaTs come in three types: *spun*, *unspun*, and *dynamically spun*. These indicate the way in which the spin of the m-NaTs is determined. *Spun* m-NaTs have their spin determined by the system constructing the navigation plan. *Unspun* and *dynamically spun* m-NaTs have their spins determined as a function of the robot's position, their own position, and the characteristics of the current s-NaT.

Given an s-NaT and a collection of m-NaTs (i.e., a NaT-based navigation plan), the combining function employs these properties and operations in the following way to determine the preferred direction of travel at any given position:

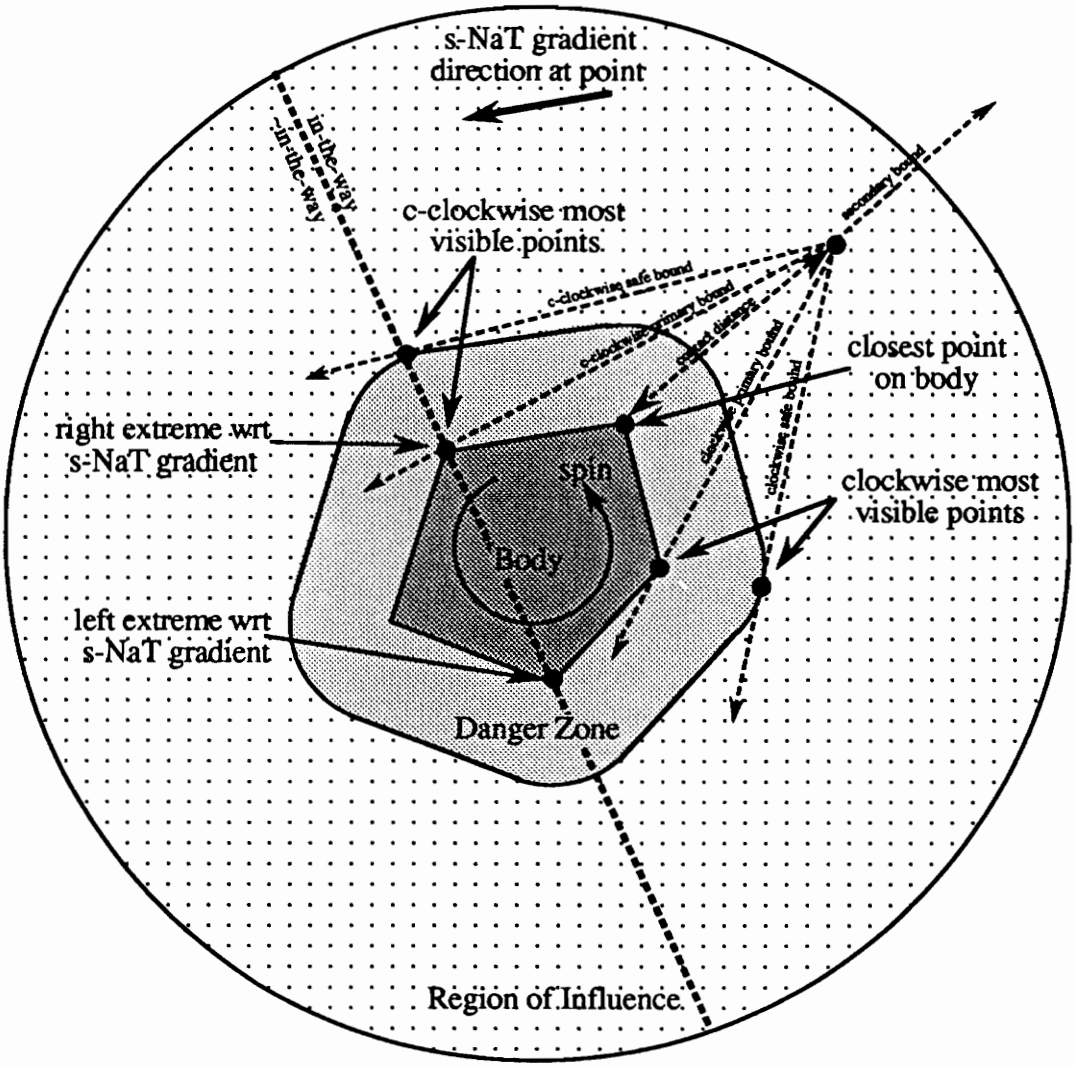
1) Eliminate from the collection of m-NaTs those m-NaTs whose region of influence does not contain the given position.

2) Set the spin of any m-NaT whose type is either *unspun* or *d-spun* by employing the techniques outlined in section 3.3.

3) Allow the s-NaT a chance to eliminate those m-NaTs that are not relevant to the navigation task at the given position, as shown in section 3.2.

4) Isolate the immediate navigation objective as defined by applying the process defined in section 2.3.1 to the s-NaT and remaining m-NaTs.

5) Calculate the preferred direction of travel at the given position using the process defined in section 2.3.2.



Structure of an m-NaT

Figure 3.15

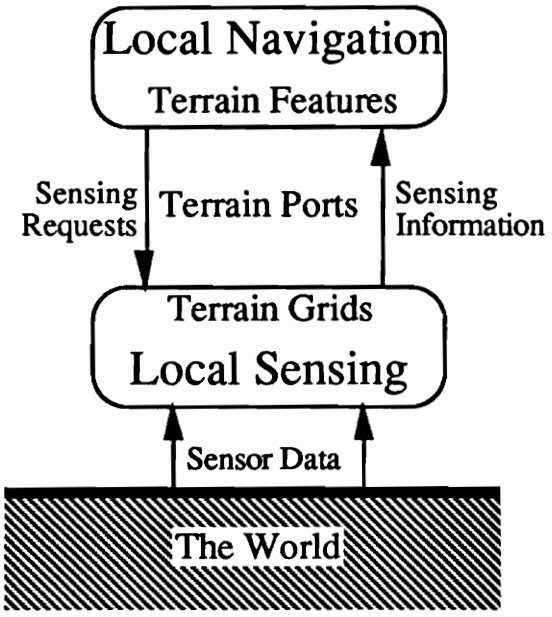
NaTs are used to describe navigation plans. The navigation decisions required to construct a NaT-based plan involve choosing both an appropriate s-NaT to characterize the navigation task and an appropriate set of m-NaTs to model the environmental constraints. The combining function handles the details of determining the way in which the plan is transformed into guidance for the robot's low level control. Because the combining function can quickly calculate the preferred direction of travel, the robot's actions can remain well connected to the navigation plan. However, for the actions to be well

connected to the physical world, the NaTs must be well connected to the robot's perception of the world. In addition, the robot's perception of the world must in turn match reasonably with the physical world or even though the robot's plan is connected to the robot's perception the resulting action will not produce the desired results. The following chapter presents a spatial modeling system that allows a navigation system to direct the actions of the sensing system, thus providing the navigation system with a mechanism of ensuring that the robot's perception of the world is up-to-date.

Chapter Four

4. Sensing the World

This chapter describes an interface between a local navigation system and a local sensing system. The interface is bidirectional, allowing the sensing system to supply the navigation system with updated information as well as allowing the navigation system to direct the sensing system in service of the navigation task. To support the interface, three structures are employed: terrain grids, terrain ports, and terrain features (see Figure 4.1). The sensing system compiles sensor data into a set of terrain grids which provide an unstructured representation of the robot's local environment. Terrain ports are windows which provide the navigation system with access to terrain grid information as well as a mechanism for directing the activities of the sensing system. Terrain features are specific observations about the terrain that are both created and used by the navigation system to form a structured representation of the robot's local environment. The transformation of sensor data from an unstructured terrain grid representation, maintained by the sensing system, to a structured terrain feature representation, maintained by the navigation system, is fundamental to the local navigation system because it requires a structured representation to create NaT-based navigation plans. Only by bringing the representation to a structured level is it possible to take advantage of situational knowledge to facilitate the navigation process. The subsequent chapter discusses the way in which these mechanisms are combined with NaTs to form an overall navigation system.



Flow of sensor data between the world and the local navigation system

Figure 4.1

4.1. Motivation

At some level, any robotic system acting in the world transforms raw sensor information into actuator commands. Whether explicitly or implicitly, this transformation at some point creates a model of the robot's local operating environment upon which actions are based. Among the more popular of the explicit models, especially for modeling natural terrain, are the grid-based[1] representations of the robot's local environment. The attraction of a grid-based representation is its conceptually clean mapping from sensor input to internal world model {Elfes87}. Many local navigation systems have employed grid-based representations and used them as a basis for planning actions to move a robot

[1]Grid-based spatial models typically refer to arrays of information used to represent a two dimensional surface. Grid-based representations also refer to quadtrees or other hierarchical data structures.

through its local environment {Moravec82, Thorpe84, Fujimura86, Kambhampati86, Puri87, Slack87, Slack88, Oskard90}. Planning is generally accomplished by embedding a standard graph search algorithm into the fabric of the representation, in which the nodes of the graph are represented by the elements of the grid and the vertices are created by connecting each node with either its four or eight immediate neighboring nodes. However, this approach has some serious limitations. One such limitation is that environmental constraints are only implicitly examined by the navigation system operating on the representation. For example, while it is possible using a grid-based elevation map to find a path that will move the robot around a boulder in an otherwise flat plane, there is no explicit information available which can explain why the path is not straight. Without the ability to manipulate explicit information about the current environmental constraints, it is impossible to exploit experience for the purposes of navigation , since there is no way to map the current situation into those with known solutions. A second limitation involves a lack of continuity in the actions generated from planning processes that have been embedded into a grid. Although there is conceptually clean mapping from sensor data into an internal grid-based representation, the mapping from grids to the type of fluid actions required of a robot operating in the world is particularly difficult (e.g., traversing a minimum radius turn). This problem has typically been referred to as stair-stepping and is a result of the connectivity scheme used to create the graph employed by grid-based search techniques to find paths. While efforts have been made to eliminate stair-stepping by either post-processing the path {Thorpe84} or employing a more sophisticated search technique {Slack88}, the resulting paths, although smoother, represent only a partial solution. In general, navigation systems relying on grid-based representations and plans must abandon any real chance of accounting for the kinematic and dynamic constraints of the robot, ignoring them entirely or restricting the robot's motions in such a way as to fundamentally limit its capabilities. However, if the representation can be brought to a feature-based level {Levitt87, Keisey88}, control techniques that rely on a structured model of the world can be employed.

A further problem with navigation algorithms embedded in a grid is that they must assume that the world does not change while the algorithm is running. The reason for this assumption is that there is no good way of dynamically propagating the effects of a change in the representation through the system without rerunning the algorithm from the start. Synchronous interaction between a robot's planning and sensing systems is inadequate if a

new plan must be constructed from scratch each time a sensor update is received, assuming plan generation is non-trivial. These problems are particularly pronounced when the issues of local navigation are addressed. The more general solution provides explicit support for disjoint operation between the sensing and navigation portions of a robot, in which the navigation system maintains a navigation plan that can be quickly modified to incorporate the implications of sensory updates.

The world is a vastly complex place, potentially providing the sensing system with a very high bandwidth of raw data that requires transformation into meaningful information. One way to reduce the workload of the sensing system is to provide it with a focus of attention. This allows the sensing system to direct its resources in service of the current navigation task, rather than expending resources generating information that is not used to accomplish the navigation task. This process of directing the activities of the sensing system in support of the current navigation task is an instance of task-directed sensing. Typical uses of grid-based models do not provide a controlled mechanism for directing sensing activity; instead they rely on the assumption that the sensing system automatically gathers all the information needed to accomplish the task without knowledge of that task. This assumption results in systems that become "bogged down" as they attempt to sense the entire local environment, or in systems that fail because information critical to the robot's success has not been made available. Another reason that both the sensing and navigation systems must support task-directed sensing is to ensure that the navigation system's current plan remains synchronized with the state of the world. For example, if the current navigation plan indicates that the robot should proceed to the right around rock#3, then as the robot moves towards the rock, the sensing system should be directed to refine the rock's perceived location so that the robot's perception of the rock is as close to ground truth as possible.

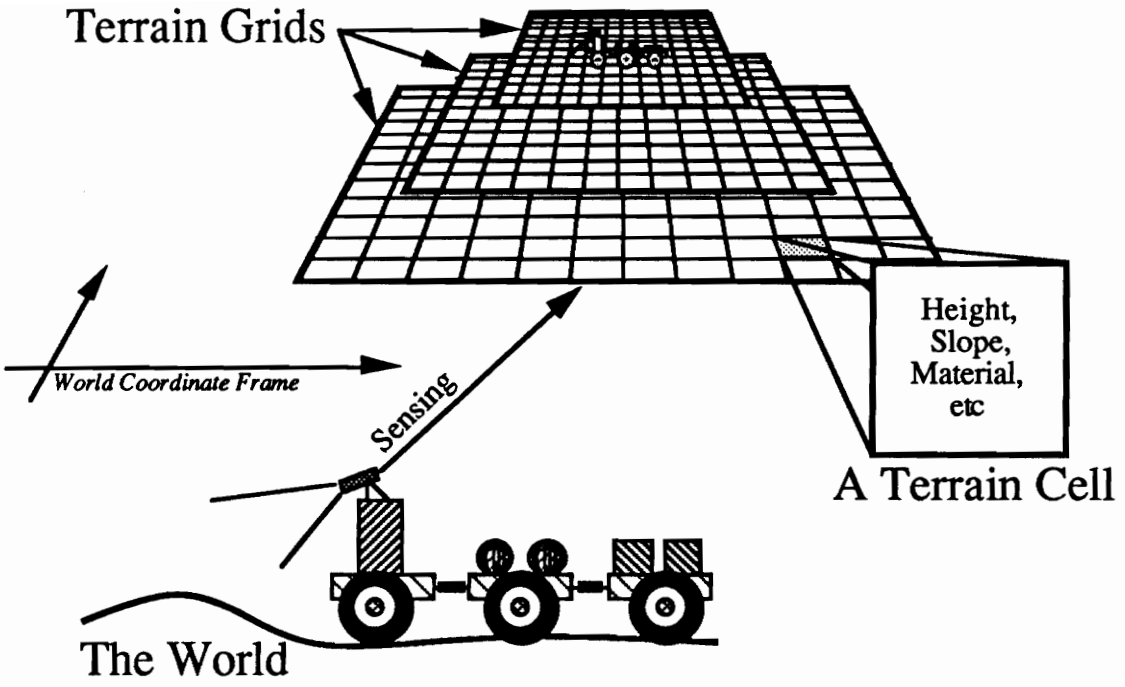
The work reported in this Chapter is directed specifically at supporting the modeling issues associated with local navigation of a mobile robot moving through the world where its only view of the world is provided by an unstructured grid-based model (or terrain grid). In particular, a spatial modeling system for creating and maintaining a structured model of the robot's local environment is presented.

The spatial modeling system has two levels of representation: an unstructured level created and maintained by the sensing system in the form of terrain grids (i.e., a grid-based

representation), and a structured model maintained by the navigation system in the form of terrain features (i.e., measurable characteristics of the terrain). For example, a terrain feature might represent the average slope of a surface patch or a ridge line extending out in front of the robot. Terrain features are created in two ways: by forming a meaningful collection of other terrain features, or by transforming the sensing data in the terrain grid into a symbolic format using a filtering function. To connect terrain grids and terrain features in a useful way, terrain ports are introduced. Basically, a terrain port is a window into the terrain grids that is used by the navigation system to examine information and allow the sensing system to notify the navigation system when new sensor information has been acquired. A communications infrastructure is introduced which allows these three elements (terrain grids, ports and features) to form a bidirectional interface between the sensing and navigation systems of a mobile robot.

4.2. Terrain Grids: A reservoir of spatial information

The foundation of the spatial modeling system is an unstructured representation of the local environment built and maintained by the sensing system as a set of terrain grids (see Figure 4.2). A terrain grid is an explicit representation of the terrain surface encoded as square arrays of square terrain cells. Each grid has a fixed cell resolution and each cell of the grid has a fixed size. For example, a terrain grid might be defined as a 10 by 10 array, each cell of the array being 1 meter on a side, resulting in a grid with a world extent of 10 meters by 10 meters. The dimension of the grid and the size of the cell are determined by the needs of the navigation system using the information and the limitations of the sensing system providing the information. The sensing system maintains a set of grids at varying resolutions in order to model the robot's local environment. For example, the sensing system might maintain three terrain grids: one with 10 cm by 10 cm cells covering a 10 meter by 10 meter patch, another with 20 cm by 20 cm cells covering a 20 meter by 20 meter patch, and one with 35.3 cm by 35.3 cm cells covering a 35.3 meter by 35.3 meter patch {Moravec87, Gat90b}. In order to maintain flexibility, there is no explicit restriction on the way in which the resolutions of the terrain grids relate to each other; any such restriction is solely a requirement of the sensing system or any other system using the information supplied by the terrain grids.



Sensor data is compiled into terrain grids by the sensing system

Figure 4.2

Each terrain cell is a container for storing sensor information about that portion of the terrain surface onto which the cell maps. Each cell of the grid has slots which store quantitative and qualitative information characterizing its particular portion of terrain surface. With respect to modeling natural terrain, the cell slots most typically contain quantities, such as the elevation, slope, and/or roughness of the surface {Wilcox87}. When the terrain is manmade, the cells contain information relating the height of the terrain as well as the composition of the material found in the cells (e.g., carpet, concrete, paper, plaster). For natural settings, neural networks have been used to label the composition of the terrain surface (e.g., grass, leaves, dirt, etc.) {Marra88}. Another classification scheme uses information from neighboring cell slots to generate hybrid information by identifying regions of cells with a similar quality (e.g., identification of contiguous terrain cell patches with a slope between 0.0 and 0.25) {Doshi88}. Because of the clean mapping from the surface into the cells of the grid, terrain grids are useful for fusing information from different sensor modalities. If, however, it is inconvenient or impractical to fuse the

information into one terrain grid, then multiple terrain grids can be used, one for each sensor modality. Thus, any sensing modality capable of transcribing its information into a grid aids in the process of sensing the world.

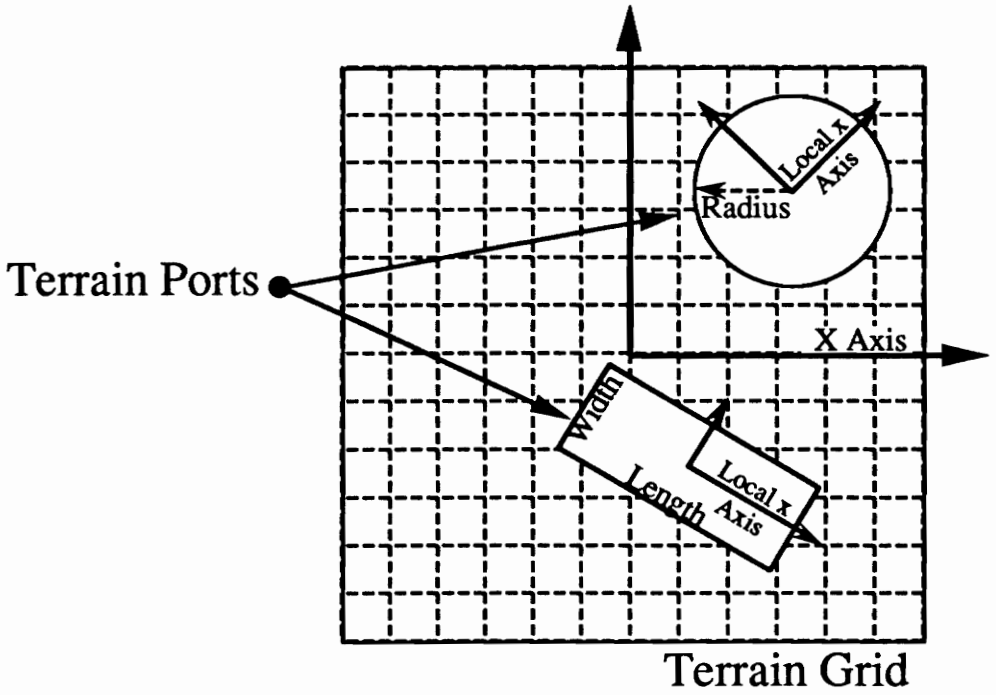
In order for the terrain grids to maintain a more or less fixed position relative to the robot as it moves about the world, the grids respond to a translation event. Translation of a grid is handled in an incremental fashion, storing the robot's relative translations until the grid can be scrolled by an integer number of cell-widths at a time. This is done so that most of the information contained within the grid remains more or less unaffected by the translation, except for the information on the leading and trailing edges of the grid.

4.3. Terrain Ports: Windows on the World

Terrain ports form the communications link between the sensing and navigation systems. Terrain ports are used to allow the flow of information from the sensing system to the navigation system as well as to provide the navigation system with a mechanism for directing the activities of the sensing system. Simply speaking, a terrain port is a window through which the cells of a terrain grid are made visible to the navigation system. Terrain ports allow the navigation system to view the information available in the terrain grids selectively as needed to support the current navigation task. More specifically, the navigation system accesses information in the terrain grids by opening a terrain port on a particular grid; any particular port can be bound to only one grid. Although more general types of port geometries are possible, two types of terrain ports are implemented in this system: circular and rectangular (see Figure 4.3). The location and orientation of a terrain port is defined with respect to the local coordinate system of the terrain grid[2]. Each terrain port has its own coordinate frame which allows the terrain grids to be accessed in the coordinate frame that is most natural for the algorithm using the information. There is no restriction on the number of terrain ports that can be opened on a single terrain grid; furthermore, because terrain ports are independent of one another, there is no restriction as

[2]Because terrain grids are non-rotatable, they provide a fixed frame of reference with respect to the world.

to the way in which the ports opened on a particular terrain grid spatially relate to one another.



Terrain ports opened on a terrain grid

Figure 4.3

Terrain ports form the communications link between the sensing and navigation systems. As such, there are four events to which they must respond so that the two systems can intercommunicate: translation of the terrain grid to which the port is bound; updates to the cells in the terrain grid visible through the port; requests by the navigation system for sensing activity; and requests by the navigation system to translate, rotate and/or scale the terrain port.

When the sensing system determines that a terrain grid must be translated in order to stay more or less centered with respect to the robot, the sensing system sends a grid translation event to all of the terrain ports opened on that terrain grid. The terrain ports respond to this event by translating their position the same amount in the opposite direction. This allows the terrain ports to maintain the same relative location in world coordinates. Sometimes, however, this translation causes a terrain port to translate past the bound of the

terrain grid. When this occurs, the terrain port notifies the navigation system of the situation so that the system can handle the implications.

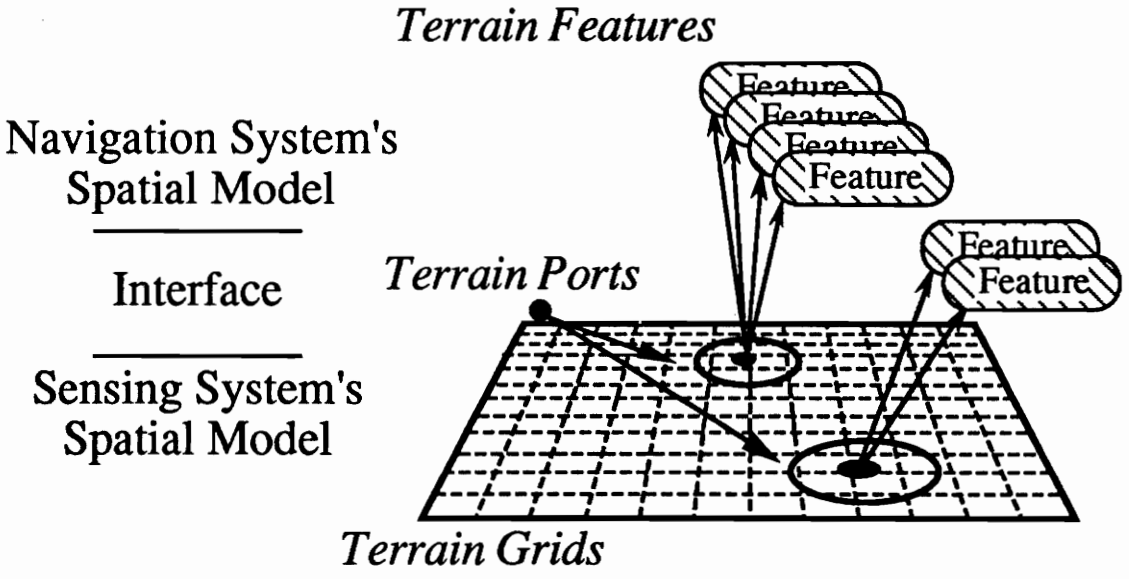
When the sensing system updates the information in the cells of a terrain grid, the sensing system notifies all of the terrain ports affected that an incremental update has occurred. The terrain port in turn notifies the navigation system of the change so that it can update its current navigation plan in order to remain synchronized with the world. While it might seem inefficient to send a message every time the cells associated with a particular port are updated, the scheme assumes that the majority of the sensing system's activities are in response to specific requests made by the navigation system.

When the navigation system requires new information or an update of old information, a terrain port is signaled to notify the sensing system that it should update the terrain grid associated with that terrain port. This does not mean that the request is instantly processed and the information is returned. Instead the request is handed to the sensing system which schedules the request for processing {Miller85}. The navigation system is notified that its sensing task has been accomplished by signaling the associated terrain port that the terrain grid visible through the port has been updated.

In order to allow the navigation system to scan the terrain and/or change the field of view, terrain ports can be translated, rotated, and/or scaled. For example, a terrain port might be used to survey the terrain in front of the robot or enlarged to increase the field of view. There are two sources of such events, resulting in two types of terrain ports: standard and floating. Standard terrain ports accept motion and scaling events from the navigation system and change their configuration appropriately. Floating terrain ports are sensitive to the robot's motions and remain relatively fixed with respect to the robot. So that floating terrain ports do not consume computational resources every time the robot moves a millimeter or rotates a microradian, they are made sensitive to the robot's actions at some granularity. This is accomplished by storing their relative translations and rotations until some threshold is reached. Floating ports are particularly useful for ensuring that the terrain grid information directly in front of the robot is up-to-date or that a port is always associated with the robot's current location.

4.4. Terrain Features: Observing the world

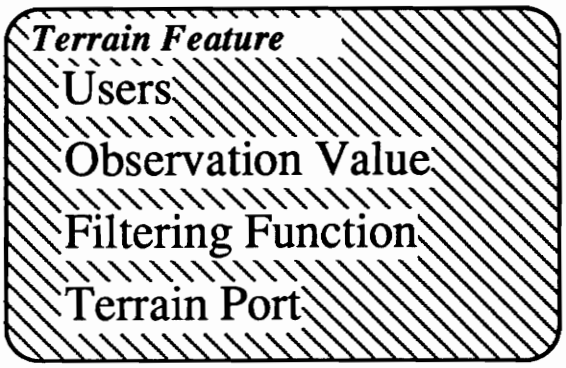
Terrain ports form the communications link between a sensing system and a navigation system. On the sensing side of the interface, terrain ports are connected to terrain grids. Terrain ports provide the navigation system with a way of viewing the terrain grid information as well as a mechanism for directing the actions of the sensing system. However, because an unstructured spatial model provides no real basis for situationally-driven control of the robot's actions, there is a need to transform the unstructured model of the terrain grids into a structured model to facilitate the navigation process. This section discusses the way terrain ports facilitate the construction and maintenance of a structured model of the world by isolating features in the terrain. For example, the value of a calculation which estimates the average slope of the terrain visible through a particular port might form a terrain feature. Terrain ports are connected to the navigation system via terrain features, so that whenever a message is sent to the navigation system from a terrain port, the message is actually sent to a terrain feature (see Figure 4.4).



Terrain features forming the navigation system's view of the world

Figure 4.4

Terrain features come in two types: primitive and composite. Primitive terrain features can be characterized as objects containing four basic slots (see Figure 4.5). One slot points to the terrain port to which the feature is bound. Each feature is bound to only one port; however, each port can be associated with more than one feature. Features are extracted by running a filtering function over the terrain grid data visible through the primitive feature's associated port. For example, a filtering function might classify the composition (e.g., grass, dirt, etc.) of the surface patch visible through its associated terrain port. The result of the filtering function is stored in the observation slot of the feature so that the value can be accessed by the navigation system at a later time. A primitive feature is primarily a pointer to a port, a filtering function, and a value slot. In addition, each primitive feature has a list of users which are tracking the feature's state. This allows the feature to signal the appropriate portions of the navigation system when an event that affects the primitive feature occurs. For example, when a feature receives a message from its associated port indicating that the terrain grid information visible through its terrain port has been updated, the primitive can signal its users that its observation value may no longer be valid.



A terrain feature object

Figure 4.5

Because a terrain feature is bound to a terrain port, it must know how to respond to the events that can be generated by that terrain port. In particular, a feature must respond to events generated when its port has translated beyond the boundary of the associated terrain grid, port translations, rotations, and/or scaling events, and terrain grid update events. When the port translates beyond the boundary of the grid, it responds by signaling all the terrain features that are bound to the port that the port no longer provides a valid window

on the terrain grid. Primitive features respond to such signals by relaying them to their users. Eventually, the signal propagates to a level in the navigation system at which the information is relevant in order to allow the appropriate corrective action to be taken. When a user of a feature ceases to function as such, it is removed from the feature's users list. If this causes the feature's user list to become empty, then the feature removes itself from the list of users associated with the terrain port and the feature evaporates. In the event that the port has moved with respect to the terrain grid or the information in the terrain grid visible through the port has been updated, the feature signals its users of the change. If one of the users requires that the feature update its observation slot as a result, it sends an update event to the feature causing it to run its filtering function and update its observation slot. Finally, updating of the observation slot causes the feature to send a message to all users of the feature indicating that its observation value has been updated.

In addition to primitive terrain features, the system supports the creation of composite terrain features. These are similar to primitive features, with the exception that they are not bound to a terrain port. Instead, composite features have a list of components (primitive and composite) that are used to construct the composite feature. The filtering function of a composite feature operates on the values in the observation slot of its components. For example, a composite feature might be formed from a combination of two primitives: one characterizing the roughness of the terrain to the right of the robot, the other the roughness of the terrain to the left of the robot. The filtering function of such a feature might relate the relative roughness of the surface to the sides of the robot, or maintain an indication as to the preferred direction of escape in the event of an emergency. Composite features can also be used as a method for integrating the value of a primitive feature over time or distance. For example, one might imagine using a floating terrain port to record the average roughness of the surface over which robot has traversed.

4.5. Simple Example: Following a sidewalk

As a simple example of the way in which the elements of the interface presented in this chapter are intended to function in support of the local navigation process, consider a situation involving a robot and an extremely primitive sensing system capable of filling the

cells of a single terrain grid with either true or false. Assume further that cells filled with true indicate that the terrain associated with the cell is composed mostly of concrete and false that it is mostly not-concrete. Assume that the resolution of the terrain grid is significantly smaller than the scale of the robot. The task presented to the navigation system is to follow a concrete sidewalk. The job of the navigation system is to locate and build a model of the sidewalk as the robot moves along it[3]. Figure 4.6 shows a simple scheme that is used by the navigation system to direct the actions of the sensing system in order to locate and track the sidewalk. The navigation system starts by opening a terrain port at some distance in front of the robot and requests that the sensing system update the terrain grid in that region. A filtering function is then used to locate the sidewalk as it relates to the terrain port[4]. The filtering function samples the terrain grid along the width of the terrain port and identifies the two locations where a transition in surface material from concrete to other is made. The result of the filtering function is the creation of two terrain features; one represents a point on the right edge of the sidewalk and the other a point on the left edge.

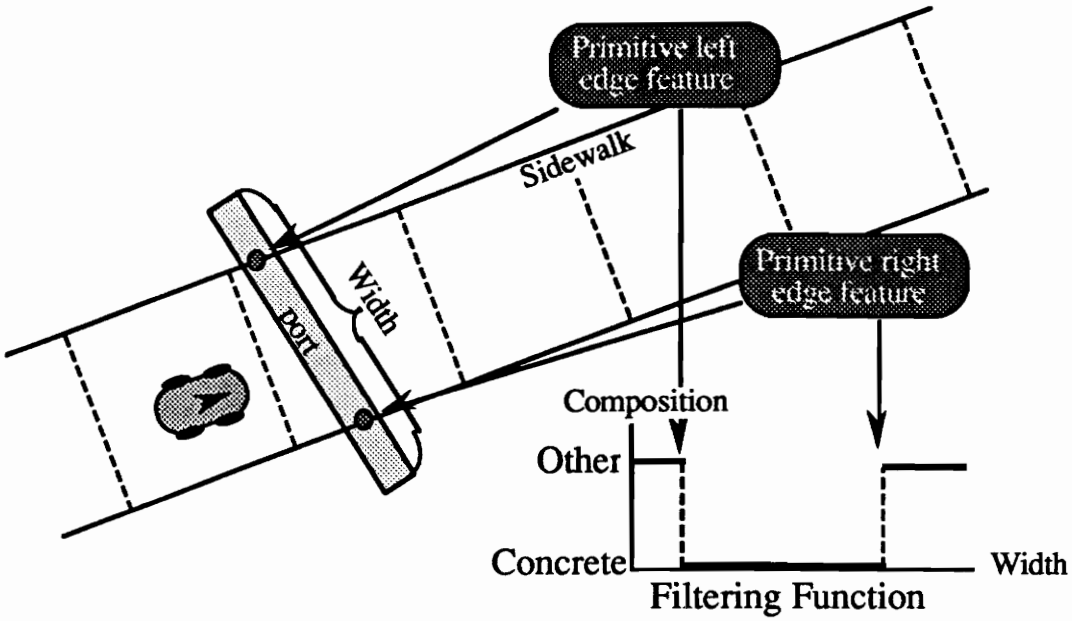
Having identified two points on the edge of the sidewalk, the navigation system requires more points in order to determine the local orientation of the sidewalk and more accurately align the port with the sidewalk. This is accomplished by rotating the terrain port slightly and using the filtering function to identify the location of two new points on the edge of the sidewalk. The resulting four points are used to refine the orientation of the terrain port and to begin the formation of a number of composite features characterizing the right and left edges of the sidewalk as well as the sidewalk as a whole (see Figure 4.7). The navigation system uses the information contained in these composite features to further translate and rotate the terrain port and acquire more edge features. Once the model of the sidewalk has been built up to a sufficient level, the navigation system starts the robot in motion down the sidewalk, using the feature-based model of the sidewalk to drive the navigation process. As the robot travels along the sidewalk, the navigation system continues to push the terrain port out in front of the robot in order to direct the sensing system and extend the representation of the sidewalk. One way to reduce the burden on the

---

[3]Navigation issues involved in controlling the robot's actions are not explicitly addressed here.

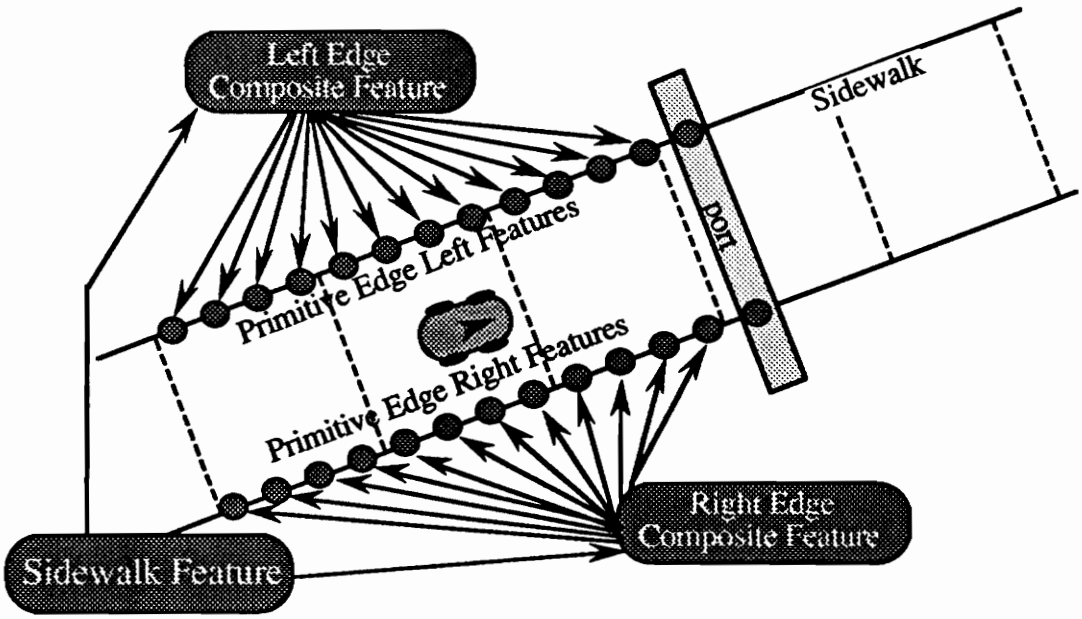
[4]Assume that the terrain port crossed the sidewalk on the first try.

sensing system at this point would be to break the single port into two smaller ports, one to track the left edge of the sidewalk and another to track the right edge. This scheme, however, would not allow the system to detect obstacles occluding its path along the sidewalk and is therefore less robust.



Using a ports to track a sidewalk

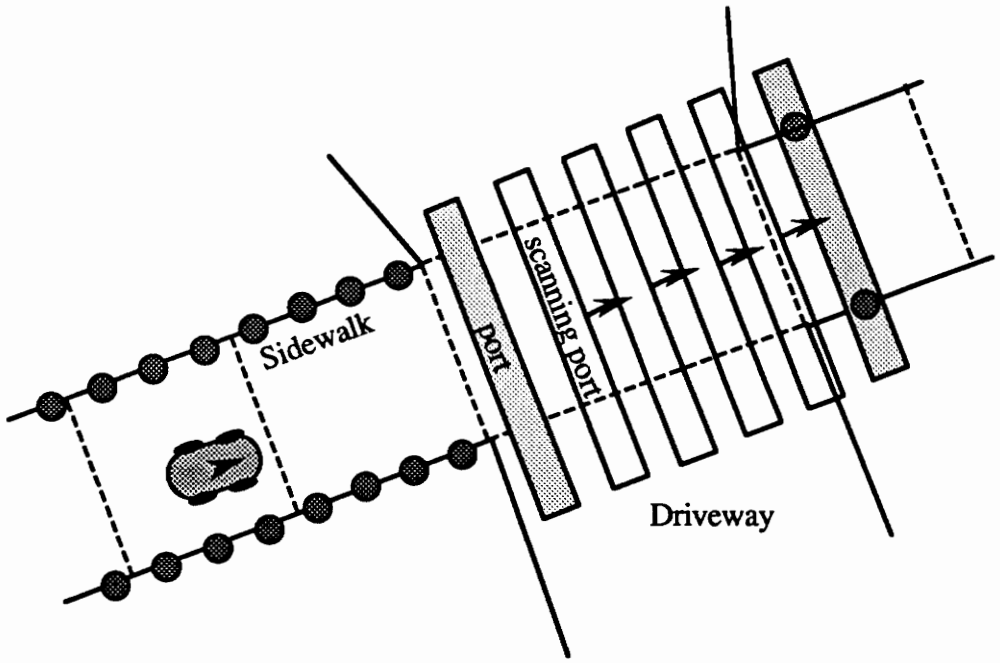
Figure 4.6



Building a model of the local situation

Figure 4.7

The situation is more complex if the sidewalk crosses a driveway or similar disruption in the status quo, in which the filtering function is unable to identify the edges of the sidewalk. In this case, the navigation system could open an additional terrain port and use it to direct the sensing system to scan the terrain further ahead in order to determine where the sidewalk resumes (see Figure 4.8). While more complicated situations could be imagined in which this basic scheme would fail to locate the sidewalk, in most situations such simple techniques are adequate. In situations where they fail, a more detailed analysis of the situation is required. One approach would be for the navigation system to direct the sensing system in order to facilitate the construction of a feature-based representation of the driveway; this would provide the navigation system with a basis for determining where the sidewalk resumes. Without the ability to direct the actions of the sensing system, it is difficult or impossible for the navigation system to ensure that the data required to accomplish the current task is available.



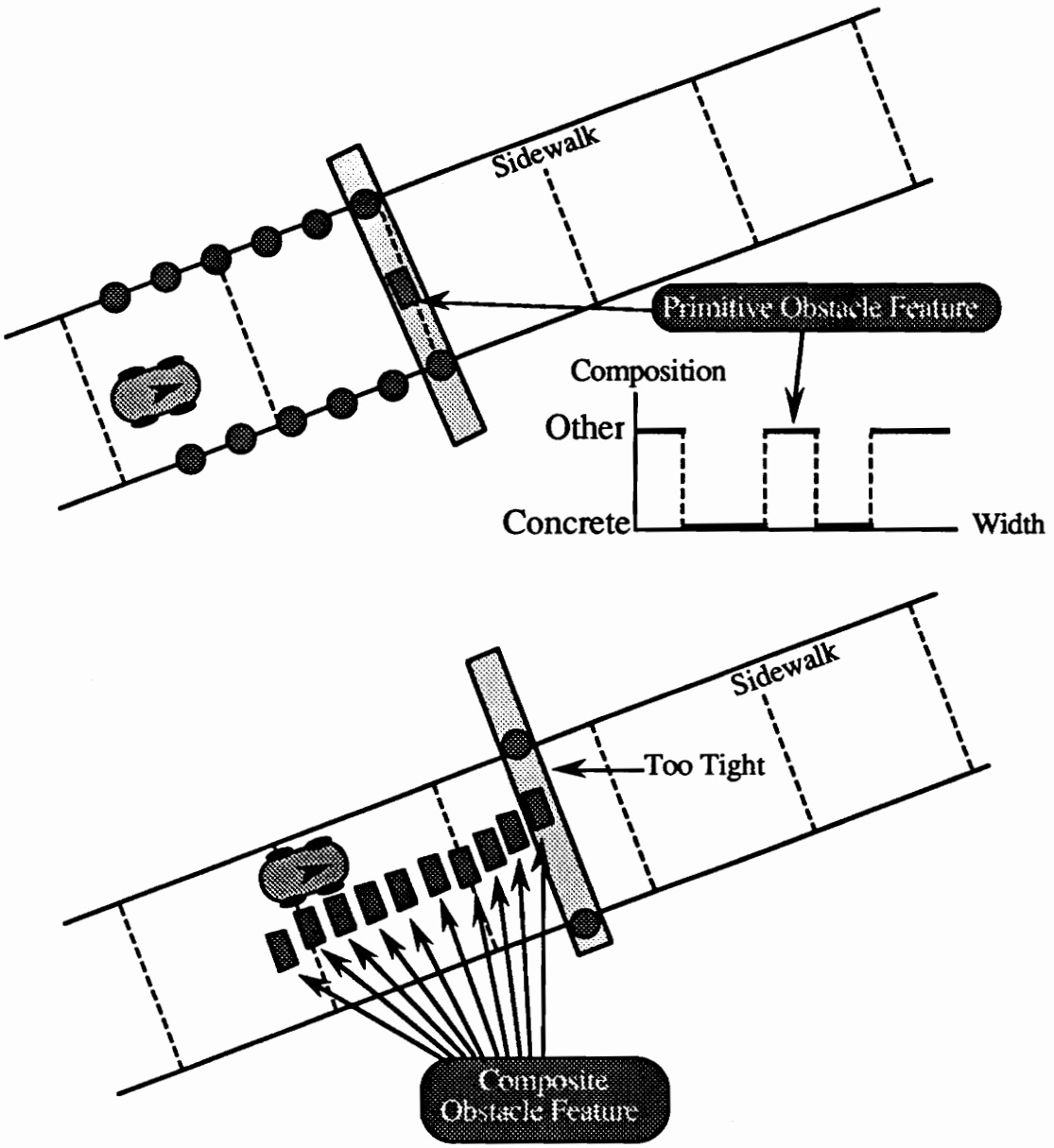
Encountering a driveway

Figure 4.8

To further illustrate the need and use of a feature-based model, consider what happens when the robot encounters an obstacle on the sidewalk (i.e., anything that is not concrete).

The top portion of Figure 4.9 shows such a situation. The result is interpreted as an obstacle occluding the sidewalk and results in the creation of a new composite obstacle feature used to map the obstacle's extent. Using a naive navigation scheme, assume that the navigation system chooses to move to the left of the obstacle where the distance between the obstacle and the edge of the sidewalk is the greatest. The navigation system then proceeds, with the added complexity that the obstacle is being tracked and its representation is being augmented as more observations about it are acquired. The bottom portion of Figure 4.9 shows the situation after it has been allowed to develop further. The composite feature representing the obstacle has been formed, and the navigation system has determined that the obstacle is occluding its current path. At this point the navigation system has an adequate spatial representation to reason that, in order to make further progress, the robot must back up and proceed to the right around the obstacle.

It is important to note in the above example that having a symbolic model of the situation allows the navigation system to know not only that it must back up in order to continue, but also how far to back up. While this might not seem significant, the ability to obtain such knowledge is critical in situations where a navigation problem extends beyond the robot's current sensor range and/or in situations where there are limits on the amount of terrain grid data that can be stored in the robot's memory system (features-based representations are substantially more compact). Consider a slightly different situation occurring on a much larger scale, where a robot has been moving along a canyon floor. Now suppose that the navigation system has determined that the canyon floor is impassible. At this point a navigation system without a model of the "big picture" might come to the conclusion that further progress requires that the robot move back up the canyon. However, once the impassable region is beyond the robot's sensor range, the navigation system would direct the robot back down the canyon toward the blockage. When a navigation system capable of constructing a feature-based representation of its environment is presented with the same situation, there is enough of a model of the situation to reason about the "big picture" when the impasse is encountered.



Obstacle Detection

Figure 4.9

4.6. Final Words

This chapter has presented a spatial modeling system intended to connect a navigation system to a sensing system that is capable of providing only an unstructured view of the world. The system provides a mechanism for use by the navigation system to direct the actions of the sensing system and to create and maintain a structured representation of the local environment. By making an explicit separation between the world model built by the sensing system and the representation of the world used by the navigation system, it is possible for the navigation and sensing systems to operate asynchronously. Thus, information in the terrain grids can be updated independently of the navigation system's current activities, while at the same time notifying the navigation system that a change has occurred.

Chapter Five

5. The LNav System

LNav is a local navigation system which directs the actions of a mobile robot in service of given local navigation tasks. LNav employs terrain ports to direct the activities of a sensing system in acquiring the information necessary for it to build a structured model of the robot's local environment. This model provides the necessary information needed to construct and maintain a NaT-based navigation plan for accomplishing the current navigation task. LNav demonstrates that by coordinating sensing and acting via a flexible plan for navigation it is possible to produce both smooth and goal-directed behavior in a mobile robot.

5.1. Overview

Figure 5.1 shows a minimal support architecture for LNav. The architecture includes a "High Level" system that is responsible for issues of global navigation and provides LNav with a stream of local navigation tasks, the most recent of the tasks representing the current navigation task. LNav's interface with the world is buffered by two low-level systems, which are responsible for the robot's direct interactions with the physical world. The sensing system collects raw sensor data and transforms it into a grid-based unstructured model of the local environment. LNav interfaces with the sensing system via terrain ports, allowing LNav to both access the sensing system's model of the world and direct its actions. The action system drives the robot's actuators and causes the robot to act in the world. LNav controls the robot by continually providing the action system with a locally preferred direction of travel. Because there are a number of low-level reactions

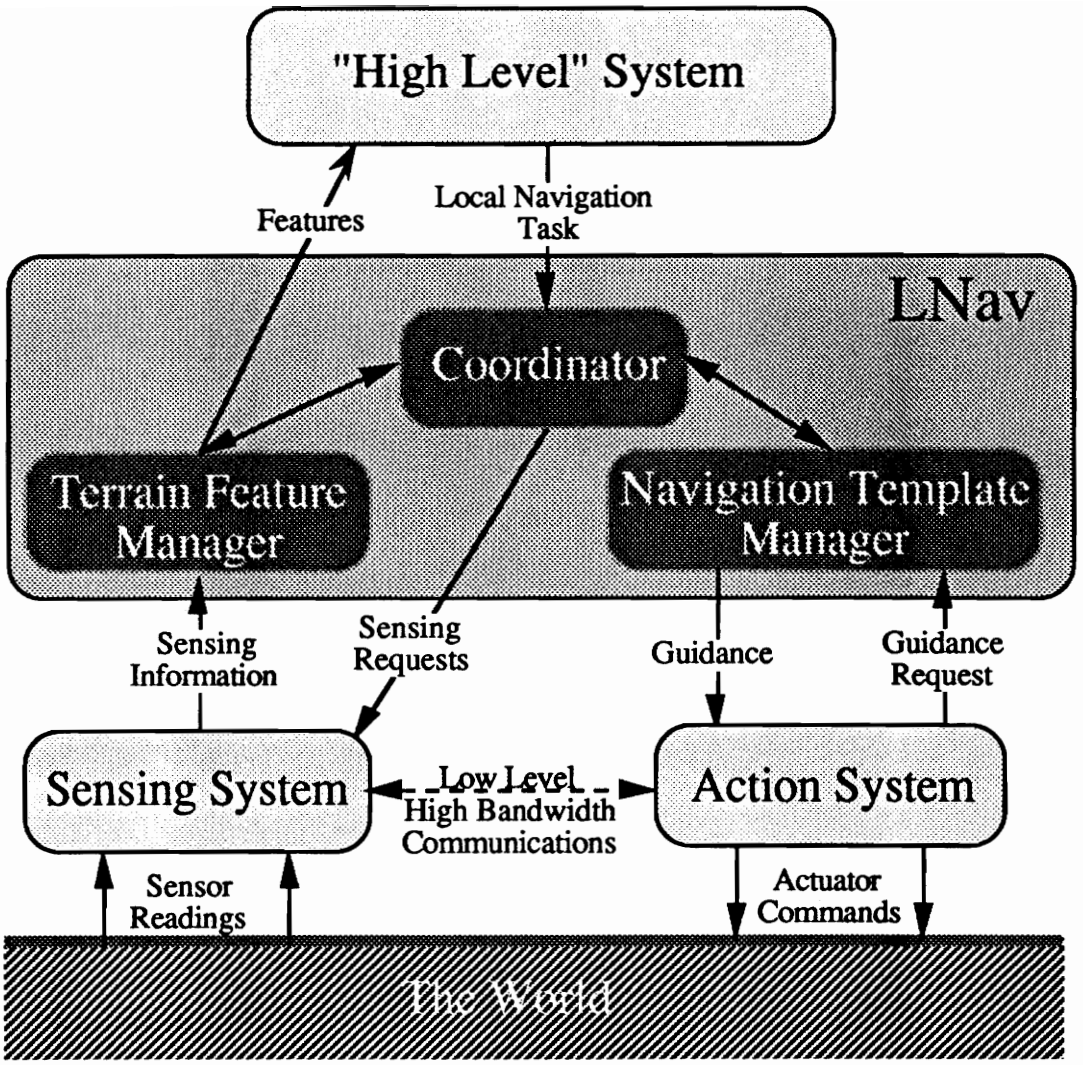
which a robot operating in the world must be capable of in order to survive, a high-bandwidth communications link is assumed to connect the sensing and action systems.

Internally, LNav is constructed from three components: the Feature Manager, the Coordinator, and the NaT Manager. The Feature Manager maintains LNav's current perception of the local environment by transforming the unstructured model of the world provided by the sensing system into a structured model of the world. The NaT Manager maintains the current navigation plan and provides guidance to the action system by transforming the current plan into a preferred direction of travel from the robot's current position. The Coordinator is responsible for directing the actions of the sensing system and maintaining the NaT-based navigation plan.

LNav has been implemented and controls the actions of a simulated robot moving around in Boulder World. To demonstrate utility of the ideas presented in this dissertation, Boulder World is filled with boulders that the robot cannot roll over and therefore must go around. The view of the world provided by the sensing system consists of a single terrain grid whose cells contain either a sampled average of the elevation of the terrain associated with the cell or no information at all. The cells of the grid only become filled if the sensing system is directed by LNav to sense the associated portion of the world. Figure 5.2 shows an example of the type of data provided by the sensing system. The cells that are shaded represent locations that have been filled with sensed information; the others contain no information[1]. For LNav to construct NaT-based navigation plans it requires a structured representation of the local environment. The Feature Manager extracts boulder features from the terrain grid to create and maintain a feature-based model of the local environment. LNav's Coordinator uses the resulting model to construct a navigation plan. This is accomplished by associating an m-NaT with each boulder feature in the current model. The Coordinator's job is to keep the navigation plan up-to-date with the changing perception of the location of the boulders near the robot in order direct the robot's actions, avoiding the boulders and accomplishing the navigation task.

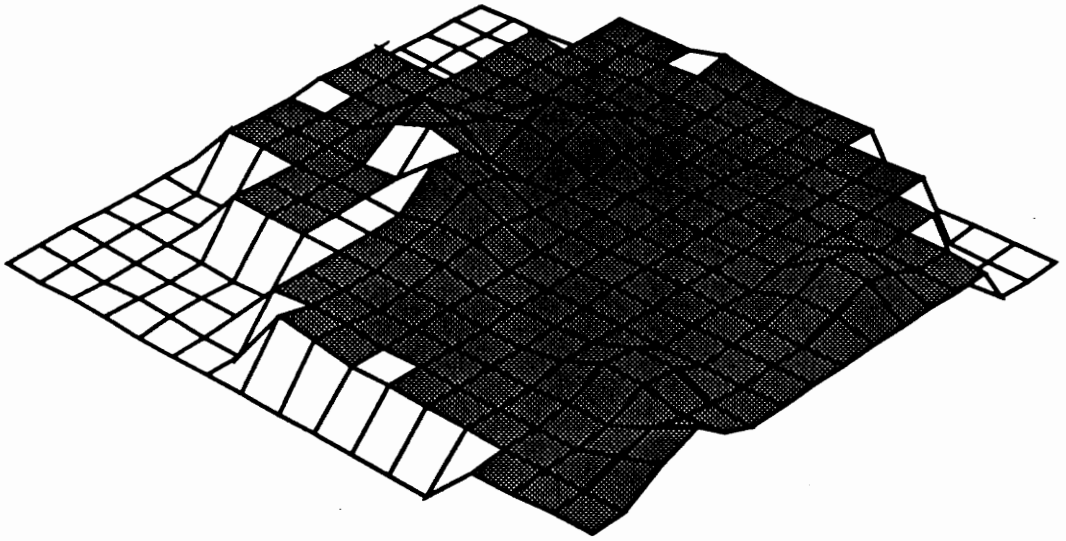
---

[1]Some of the cells containing no information appear to have elevations only as a result of the display program which generated the picture.



LNav Architectural Setting

Figure 5.1

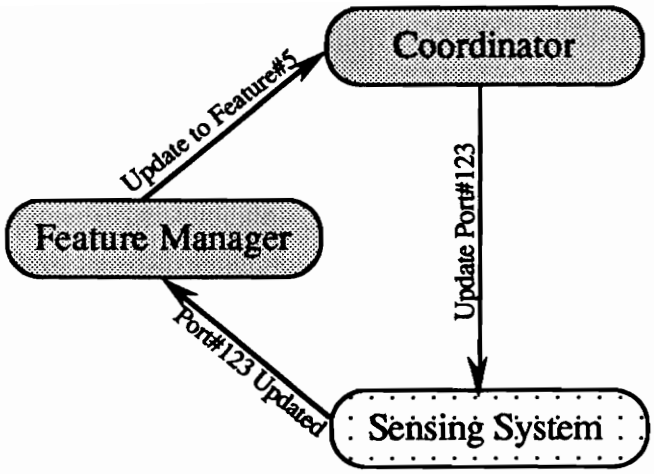


Sample of the information provided by the sensing system

Figure 5.2

5.2. Sensing the World and Extracting Features

The Coordinator is responsible for manipulating terrain ports and directing the sensing system in service of a navigation task. The sensing system, upon being directed to update the portion of its terrain grid associated with a particular terrain port, employs sensors to acquire the necessary data, and then updates the terrain grid. Once the update is complete, the sensing system notifies the Feature Manager via the terrain port that the terrain grid local to the port has been updated. The Feature Manager then runs appropriate filtering functions over the terrain grid information to extract any relevant terrain features and update its feature-based model of the local environment. If the update results in a modification to the feature-based model, then the Feature Manager notifies the Coordinator of the change so that the Coordinator can consider the impact of the change on its navigation plan (see Figure 5.3).

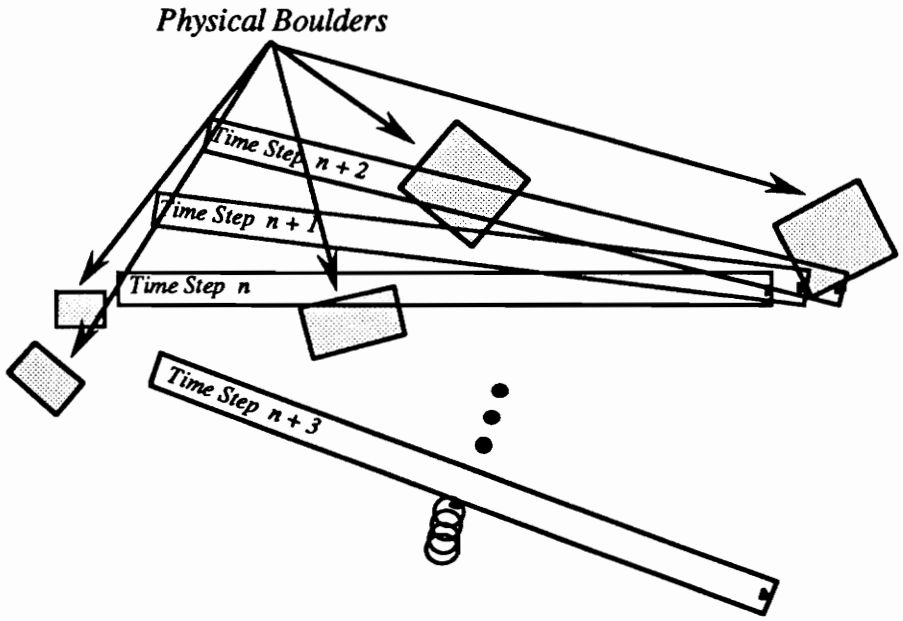


LNav's sensing cycle

Figure 5.3

To construct and maintain navigation plans in Boulder World, LNav requires a model of the boulders that are local to the robot's current position. One difficulty is that that measurements and actions in the world introduce uncertainty into that model of the world. Rather than attempting to reason explicitly about the uncertainty, LNav manages the uncertainty by repeatedly sensing the environment, keeping the uncertainty to a level which can be effectively ignored. Thus, for purposes of navigating through Boulder World, LNav requires a scheme keeping the information in the terrain grid up-to-date as it forms the basis of LNav's perception of the world. To accomplish this, a simple sensing scheme has been implemented which continually updates the region of space in front of the robot. The scheme connects LNav and the sensing system via a single, wide rectangular terrain port which is continually swept out over the region of space in front of the robot. The Coordinator sweeps the terrain port in a step-wise fashion, stepping the port's position further in front of the robot, pausing long enough for the sensing system to update the information and the feature manager to update the model of the boulders, then stepping the port's position ahead to the next location in front of the robot. The process is continued until the terrain port has been swept out in front of the robot to some maximally useful distance at which point the process is restarted with the terrain port located immediately in front of the robot and repeating the sweeping cycle. Figure 5.4 shows a portion of the robot's simulation trace overlaid with the terrain port's position and orientation at the

associated time steps. Since sensing the world, updating the terrain grid, and extracting features from the world all require finite amounts of time, it is likely that the robot will have moved between steps of the terrain port's position. The sensing scheme accommodates the robot's motion by ignoring it, always repositioning the terrain port relative to the robot and not to its last position on the terrain grid. As a result, during any single pass of the sensing scheme, portions of the terrain in front of the robot will not be sensed. However, the rate at which the port is swept out in front of the robot is significantly faster than the robot's velocity. This allows potential holes in the data to be filled, and the terrain in front of the robot to be sensed multiple times. Sensing the world multiple times is the basis by which LNav manages the world's uncertainty.

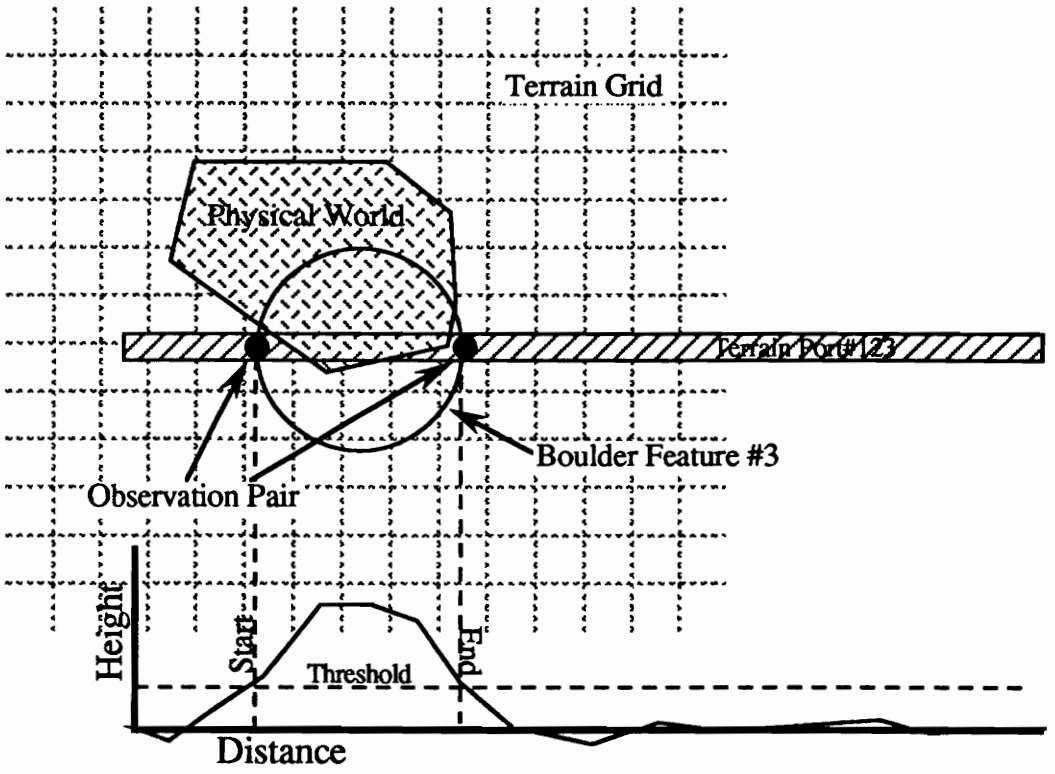


Sensing the world

Figure 5.4

Each time the sensing system notifies the Feature Manager that the terrain grid visible through the port has been updated, the Feature Manager runs a filtering function over the data and updates its model based on the results. The particular filtering function used to allow LNav to move about in Boulder World simply identifies discontinuities in the surrounding terrain that are too large for the robot to traverse. Discontinuity transitions are paired with each other forming a *boulder observation.* Figure 5.5 provides an example of

this process, showing the way in which the filtering function maps height against distance along the length of the terrain port to isolate a boulder observation. In situations where an end of the terrain port is "resting" on a boulder, the end is considered either the start or termination of an observation. Each boulder observation represents a primitive terrain feature which serves as data to the formation of composite *boulder features*. A boulder feature models the physical boulder as a circle located at the midpoint of the line segment connecting the two points forming the observation and with a radius that is equal to half the distance between the two points. The unstructured data provided by the sensing system is transformed into a structured representation of the boulder field. When the Feature Manager identifies a boulder, it notifies the Coordinator of the event so that it can use this new information to update its plan for accomplishing the navigation task.

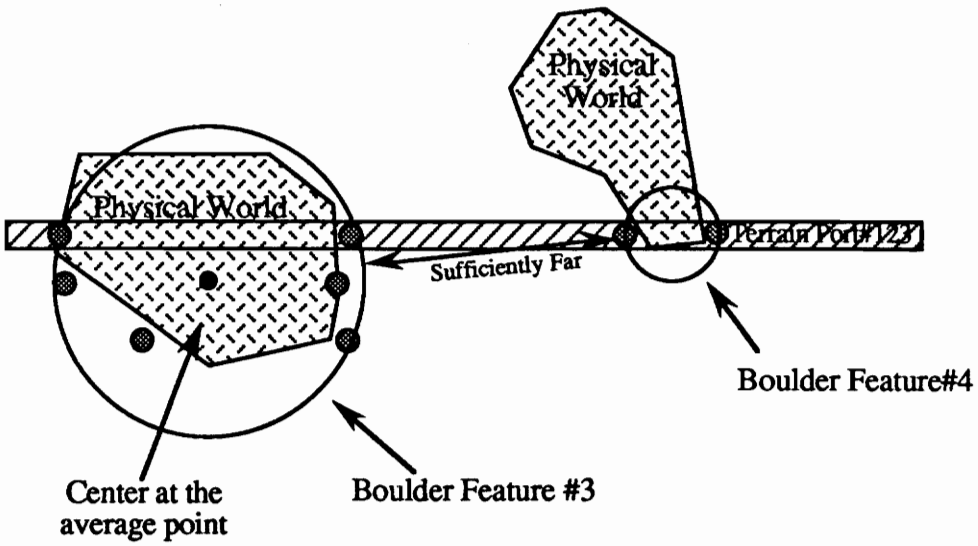


Observing a boulder and forming a boulder feature

Figure 5.5

As time passes, the Coordinator relocates the terrain port according to the sensing scheme and the Coordinator directs the sensing system to update the portion of the terrain

grid associated with the port. When the terrain grid has been updated, the Feature Manager is once again notified of the update and employs the filtering function on the updated information, attempting to extract additional boulder observations. Often a new observation is located sufficiently close to an established boulder feature so that the observation is considered to be of the same boulder. In such a situation, rather than creating a new boulder feature to accommodate the observation, the observation is integrated into the model of the closest boulder. The addition of the observation causes the center of the boulder to be changed to the average position of all the observations that currently constitute the boulder. The radius of the boulder feature is set to the minimum distance needed to encircle all of the observations. Figure 5.6, depicts a situation where the filtering function returned two boulder observations. One observation is sufficiently close to an established boulder feature that is is simply considered part of that boulder and integrated into that boulder feature. The other observation is sufficiently far from the established boulder feature that the Feature Manager creates a new boulder feature to accommodate the observation.



Integrating new boulder observations into the current model of the surroundings

Figure 5.6

This scheme for constructing a feature-based model of the robot's surroundings has a few shortcomings. First, due to the way in which boulder observations are incorporated

into the existing model, it is possible for a single physical boulder to be modeled as multiple boulder-features. This turns out to provide a better space-wise estimate of the portion of the environment actually occupied with boulders, since a single circle encompassing all the observations of the multiple boulder features may include large amounts of traversable space. However, it does not reflect the true structure of the environment. Second, modeling physical boulders with circles can result in a gross overestimation of the physical space occupied by the boulder (e.g., a long narrow physical boulder). While in many situations this approximation can unnecessarily limit the robot's ability to travel through the world, it is important that any scheme for transforming observations of the environment into an internal model be computationally efficient or the robot will spend too much time "thinking" and not enough time acting. Rather than employing a more sophisticated modeling technique (e.g., convex hulls), to ensure the overestimation does not get out hand, a limit is placed on how large a boulder feature can become. Thus, if incorporating an observation in a boulder feature would cause it to exceed the set limit, than a new boulder feature is formed with that observation. To ensure that new observations do not cause boulders to become too large, a limit is placed on how long[2] a boulder observation can be before it is split into multiple shorter observations. Finally, multiple physical boulders can be modeled as a single boulder feature. This occurs when observations of two different physical boulders are close enough so that they result in the formation of a single boulder feature rather than two independent ones, limiting the robot's ability to perceive the true structure of the environment. However, because boulder features drive the Coordinator's construction of navigation plans, this is only a problem when viable navigation options are lost when two physical boulders are modeled as a single boulder feature. To control this shortcoming the distance used to determine when two observations are close enough that they are assumed to be part of the same boulder is based on the distance between physical boulders required for the robot to pass safely between them.

---

[2]The distance between the points defining the observation.

Because the robot's actions in the world and perceptions of the world are uncertain, each boulder observation is given an expiration date[3] when it is first formed. The expiration date allows the Feature Manager to remove old boulder observations so that LNav's perception of the world remains up-to-date. When an observation has expired, it is removed from the associated boulder feature, causing its position and radius to be updated and the Coordinator to notified. If the observation is the last one comprising the boulder feature, then the boulder feature vanishes and the Coordinator is notified of the event. The use of expiration dates also serves to eliminate boulder features that are no longer relevant to the navigation task, since if the boulder were relevant the sensing scheme should have acquired more recent observations of it. In this way, observations of the world whose validity has been compromised by the accumulation of the world's uncertainties do not affect the robot's actions, allowing LNav to handle the world's uncertainties implicitly rather than having to reason explicitly about them.

In summary, the Feature Manager informs the Coordinator when its state has changed via one of three message types:

- Boulder feature add. This event occurs when the Feature Manager must create a new boulder feature to accommodate a new observation paring.
- Boulder feature delete. This event occurs when the Feature Manager finds all of the observations used to define a boulder have expired.
- Boulder feature update. This event occurs whenever the location and/or radius of a boulder feature is updated (i.e., an observation was added or deleted from the boulder feature).

---

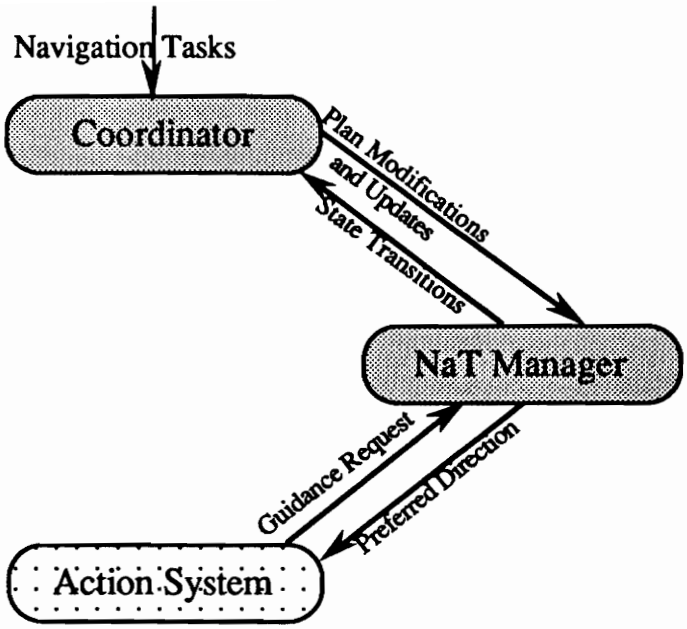
[3]The word "date" is used loosely as a typical expiration date will be only a few tens of seconds into the future.

5.3. Coordinating Features and NaTs

The heart of LNav is located in the coordinator process. This is where local navigation tasks are received by LNav and navigation plans for accomplishing those tasks are maintained. The NaT manager supports the Coordinator by holding the current navigation plan and repeatedly combining the NaTs to provide guidance to the robot's action system (see Figure 5.7). The NaT Manager does not make planning decisions, rather it supports the Coordinator's need to modify the existing plan by allowing the Coordinator to change the active s-NaT, add an m-NaT, delete an m-NaT, and/or modify an existing m-NaT. The Coordinator has three basic responsibilities:

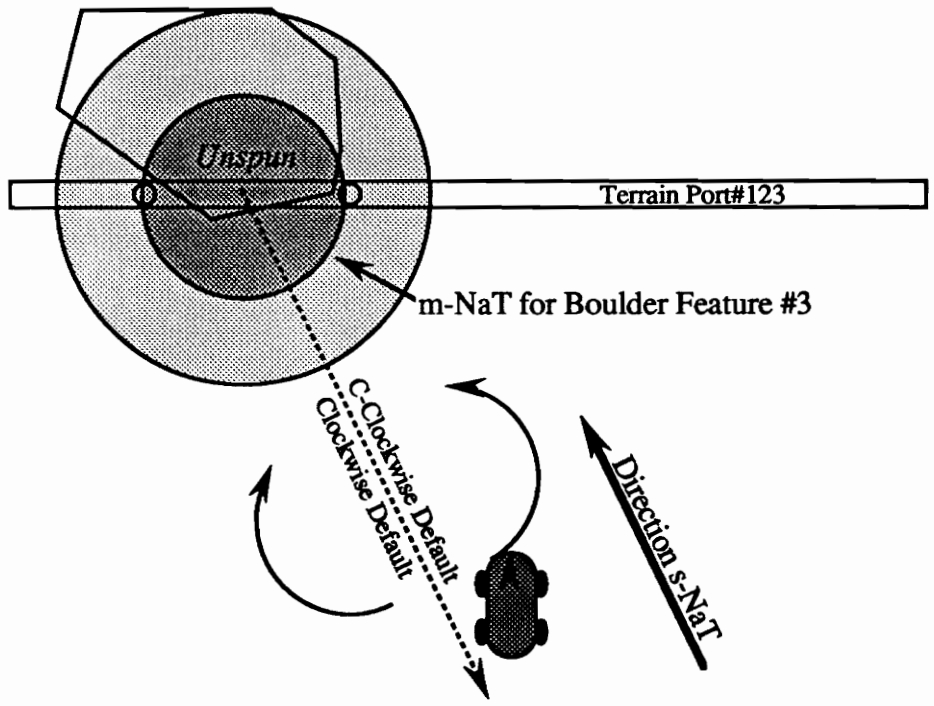
- Accept local navigation tasks.
- Direct the activity of the sensing system.
- Modify the current navigation plan to accommodate signaled changes in the feature-based model.

When the Coordinator receives a new navigation task, it associates an s-NaT with the task and delivers it to the NaT Manager, forming the basis of the current navigation plan. Each time the Coordinator receives a message indicating that a boulder feature has been identified, it establishes a new *binding* between the feature and a new *unspun* circular m-NaT used to account for the obstacle in the navigation plan (see Figure 5.8). The new m-NaT is passed to the NaT Manager; thus, becoming part of the current navigation plan.



LNav's direction of the robot's actions

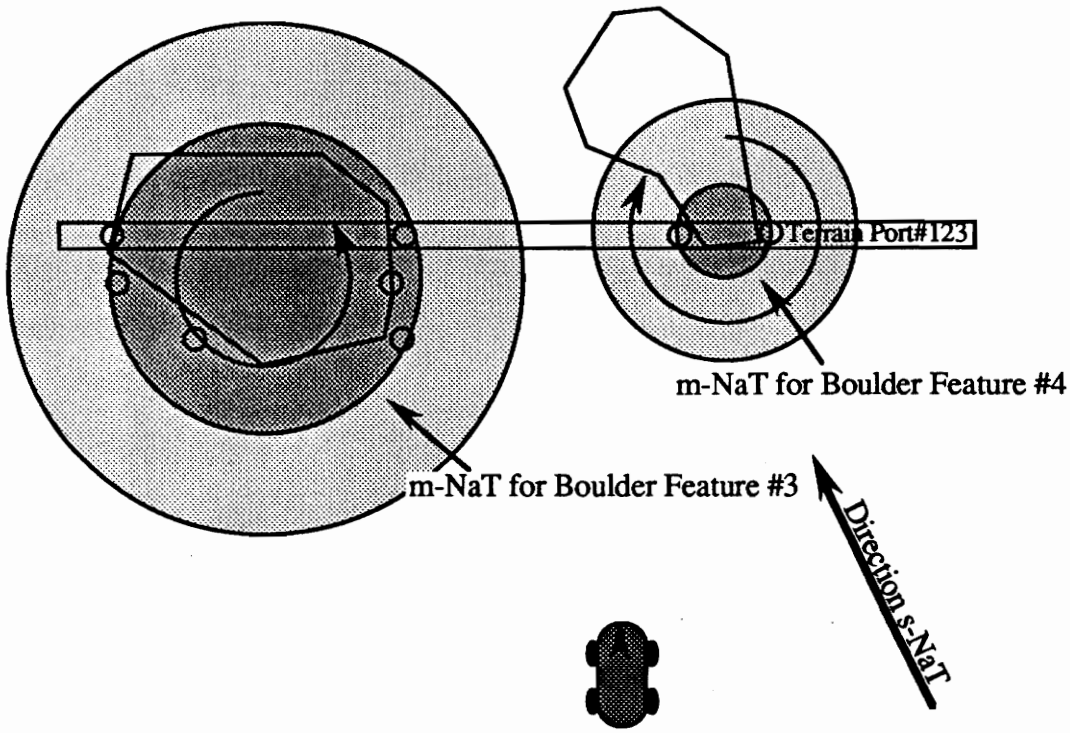
Figure 5.7



Associating an m-NaT with a new boulder feature

Figure 5.8

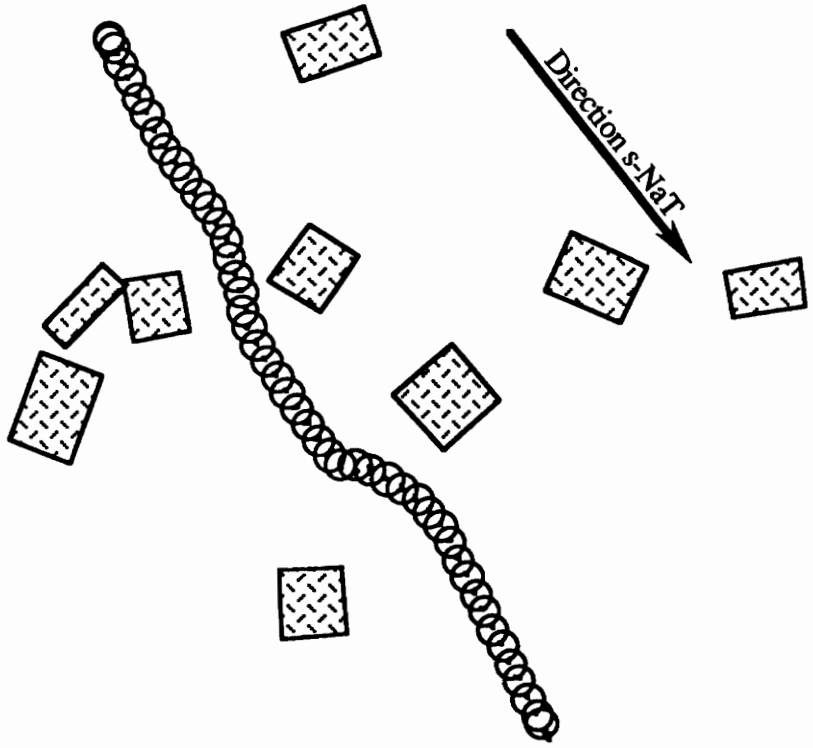
By maintaining a binding between the features in the world and the m-NaTs in the navigation plan associated with the features, the Coordinator can update the appropriate portion of the current navigation plan when it receives a message indicating that a particular boulder feature has been updated or deleted. When the Coordinator receives an update, due to a boulder observation either being inserted or deleted from a particular boulder feature, the associated m-NaT is modified to reflect the new position and radius of the boulder (see Figure 5.9). When the Coordinator receives a message indicating that a boulder has been removed from the local model, the Coordinator directs the NaT Manager to remove the associated m-NaT from the current navigation plan. The Coordinator makes the assumption that because the boulder has not been actively sensed for some period of time, it is not relevant to the navigation task. How long an observation should persist is partially determined by the relevance of boulders that the robot has passed (i.e., not in the actively sensed region of the terrain).



Updating an m-NaT to reflect changes in the perceived state of the world

Figure 5.9

This simple scheme of inserting *unspun* m-NaTs into the navigation plan produces reasonable results in situations where the robot can pass easily to either side of the boulder and still accomplish the current navigation task. A simulation trace of a circular synchro-drive robot using this scheme to move through a randomly created boulder field is shown in Figure 5.10.



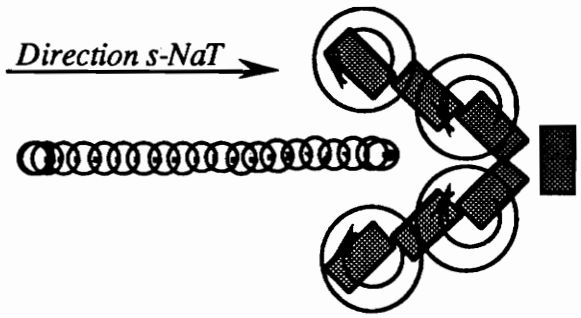
Simulation trace using a default scheme to coordinate features and m-NaTs

Figure 5.10

5.4. Avoiding Traps in Boulder World

The default scheme of using *unspun* m-NaTs to produce a navigation plan actually lacks any real notion of the word "plan" and can result in causing the robot to attempt to move between two boulders that are too close for the robot to fit (including those in contact

with one another). For example, Figure 5.11 shows a simulation trace of a circular robot controlled by LNav when the default navigation scheme is employed. Unlike the simple example shown in Figure 5.10, the boulders in this situation form a trap with respect to the direction of the s-NaT's gradient. Upon encountering this situation, the robot creates two boulder features, one corresponding to the left side to the trap and the other corresponding to the right side. The Coordinator associates an unspun m-NaT with each identified boulder feature and inserts it into the navigation plan. When the combining function is run next, the spins of the two m-NaTs are assigned. This results in a navigation plan that directs the robot between the two m-NaTs demarcating the entrance to the, as yet unforeseen, trap. As the robot advances, the sensing scheme moves the robot's sensing port forward, new boulder observations, are identified, and the local model updated. Eventually, the collection of m-NaTs used to model the boulder features forming the trap results in there being insufficient space for the robot to pass between m-NaTs with a clockwise spin and m-NaTs with a c-clockwise spin. At this point the robot either strikes a physical boulder or stops because the robot is positioned inside of an m-NaT's body (i.e., the robot has "struck" a boulder feature).



Default navigation plans can lead the robot into problems.

Figure 5.11

The reason that traps are created in the first place is that the NaTs are independent of each other, insensitive to the existence of other m-NaTs. As a result, it is possible for two m-NaTs to overlap and have different spins. However, when two m-NaTs are in contact or sufficiently close that the robot cannot pass between them, then they should be considered a single obstacle and assigned a uniform spin direction. This observation has been incorporated into the Coordinator by allowing the Coordinator to group collections of

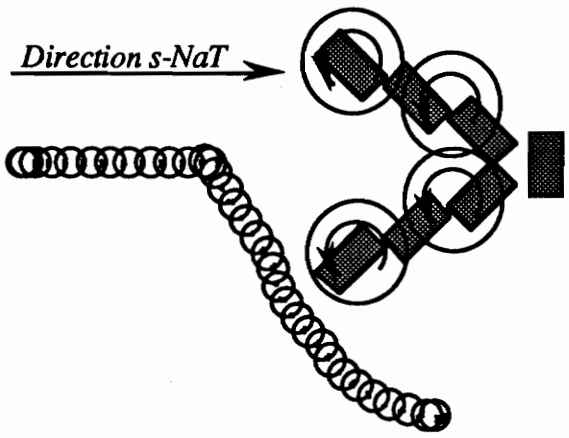
bindings between boulder features and m-NaTs. The groupings are made of boulder features that are too close to at least one other boulder feature in the group for the robot to pass between them.

When a new boulder feature is created, the Coordinator first responds by creating a new m-NaT and binding the m-NaT and the feature together. Then, rather than simply making it an *unspun* m-NaT and passing it to the NaT Manager, the Coordinator checks to see if there are any existing boulder features that are sufficiently close to the new feature to prohibit the robot's safe passage between the them. If no such boulder exists, then the associated m-NaT is made *unspun* and a new boulder group is formed with the boulder as its sole member . If, on the other hand, other boulders had been identified as being sufficiently close, then the groups associated with the boulders identified as being too close are located and combined with the new boulder forming a single group. The spin of all the m-NaTs in the conglomerate group are set to the spin direction associated with the m-NaTs of the original group containing the largest total area (i.e., the sum of the areas of the boulder features). If all of the m-NaTs forming a group are unspun, then default spins are assigned prior to the formation of the group.

The process for updating a boulder is very similar to the process for adding one. First, the m-NaT associated with the boulder feature is updated to reflect the change. Second, all boulders that are not part of the updated boulder's group are checked to determine if the update to the boulder causes it to become too close for the robot to pass safely. If no such boulders are identified, then things remain as they are. However, if there are boulders of other groupings that are too close, then all associated groups are identified and combined using the same heuristic as the one described above. It is possible that, when a boulder is updated or deleted, a gap in a boulder grouping could be formed that is sufficiently wide for the robot to pass through. However, rather than splitting boulder groups when this occurs, the groups persist until either the group is combined with another, or all the boulders of the group have been deleted. This way as boulder features are deleted the navigation plan does not suddenly change causing the robot to move back into a situation which it had previously avoided.

Consider the robot's behavior when presented with the boulder trap (see Figure 5.11) when LNav has been augmented with this ability to isolate boulder groupings and manipulate m-NaT spins based on this information. A simulation trace of such a situation

is shown in Figure 5.12. This time LNav is able to discern that all the m-NaTs associated with the trap form a single logical boulder grouping, and, as a result, are all assigned a uniform spin; thus allowing the robot to avoid the trap. The location in the trace where the robot makes a hard right turn is where updates from the Feature Manager prompted LNav to group all of the associated boulder-m-NaT bindings to form a single group and adjust the spins of the m-NaTs. This illustrates the way in which NaTs make it possible for relatively simple spatial reasoning mechanisms to handle relatively complicated navigation situations.

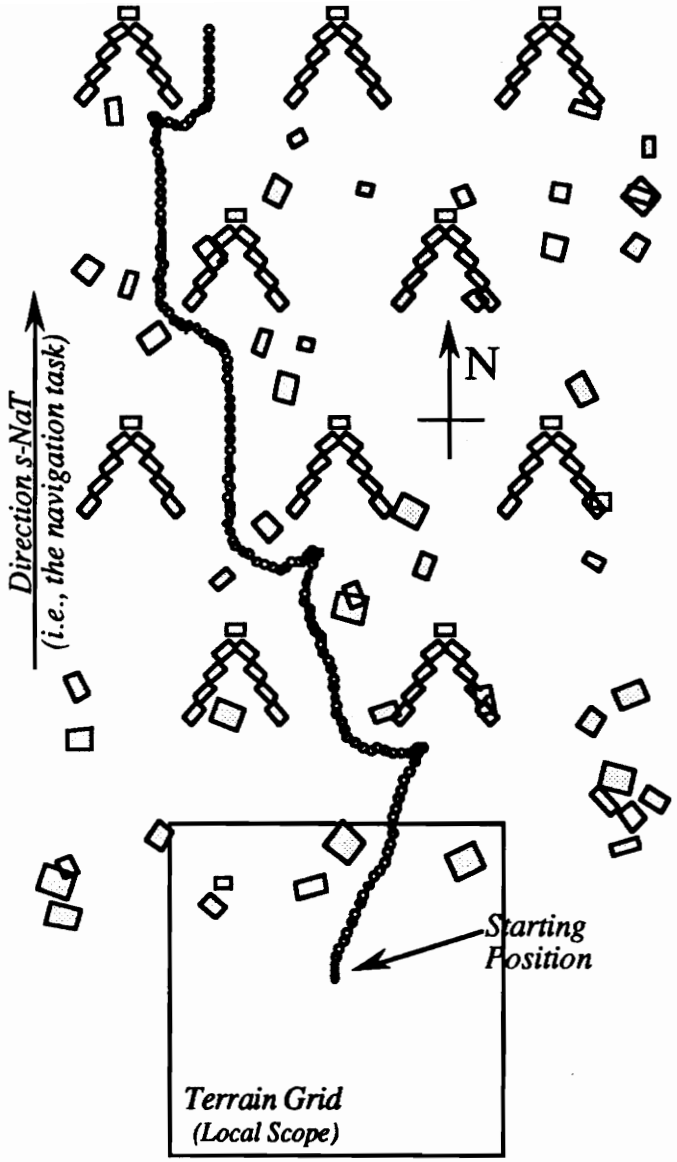


Avoiding Local Minima

Figure 5.12

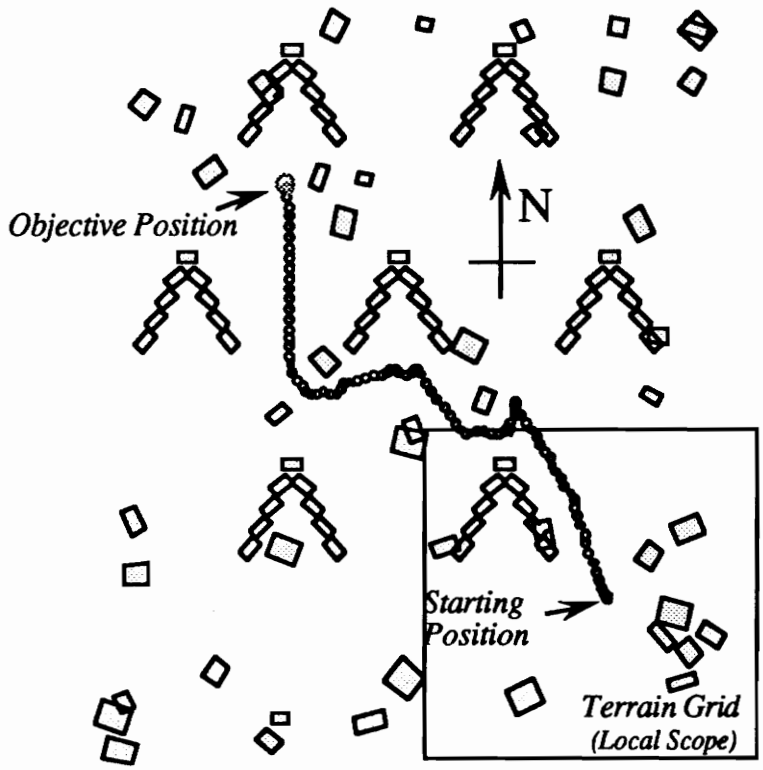
LNav is able to reliably negotiate a boulder field containing many such traps. Figure 5.13 depicts a situation where LNav was presented with the navigation task of moving north. Proceeding north, the robot moves to avoid some obstacles, forcing the robot a little to the east. This results in the default navigation scheme initially directing the robot into a trap. However, as the robot approaches the trap, more of the surroundings are illuminated resulting in the coordinator grouping all of the boulders forming the trap and assigning them all a clockwise spin. This change in the navigation plan causes the robot to avoid the trap and continue to move northward. Along the way the robot encounters a number of other such situations and avoids them all. For the interested reader, details associated with the generation of this execution trace are presented in Appendix A. A similar example showing the robot moving to an objective position is shown in Figure 5.14. Figures 5.15 and 5.16 also show LNav directing the actions of a robot moving through a boulder field. Unlike the traces of Figures 5.13 and 5.14, the traces of Figures 5.15 and 5.16 were

generated in the presence of significant amounts of sensing and acting uncertainty. In both of these traces (i.e., 5.15 and 5.16) LNav employs a sensing system that can only identify boulder observations to within 20 percent of the real distance between the robot and the observation, and an action system only accurate to within 5 percent. In spite of the uncertainty the LNav is able to reliably accomplish the tasks. LNav's ability to handle the uncertainty, in spite of the simplicity of the mechanisms used to construct and maintain the navigation plan, is indicative of the way a NaT-based navigation plan absorbs much of the world's uncertainties without explicitly reasoning about the uncertainty.



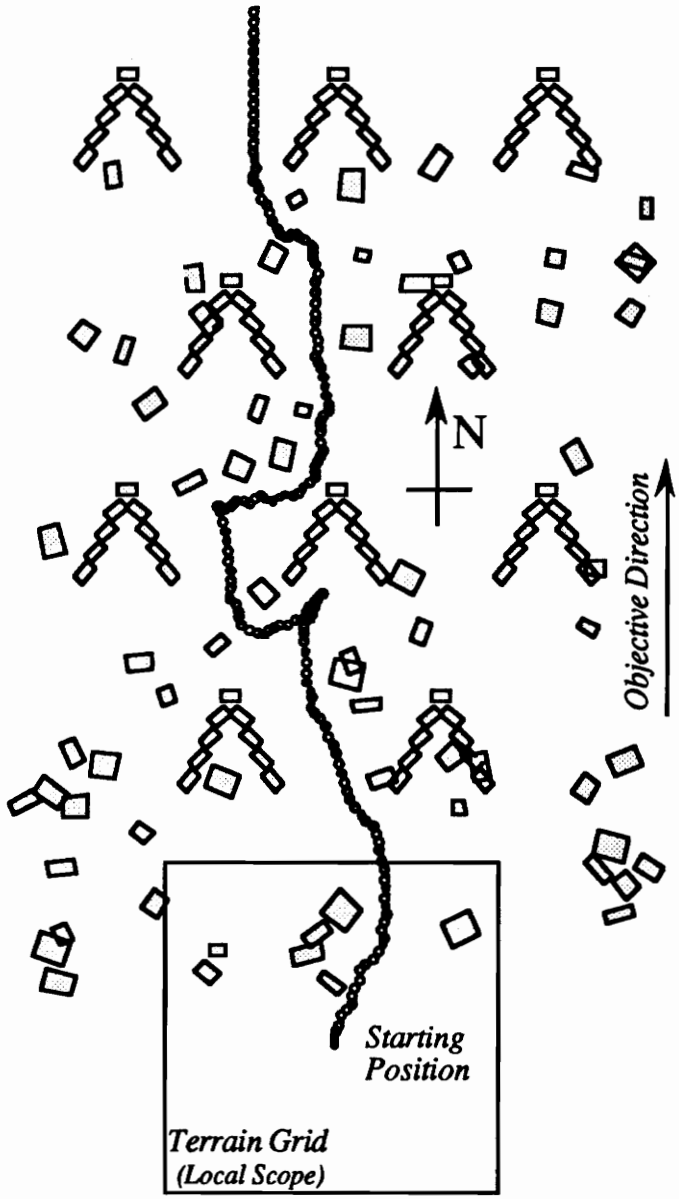
LNav directing a robot to in an objective direction

Figure 5.13



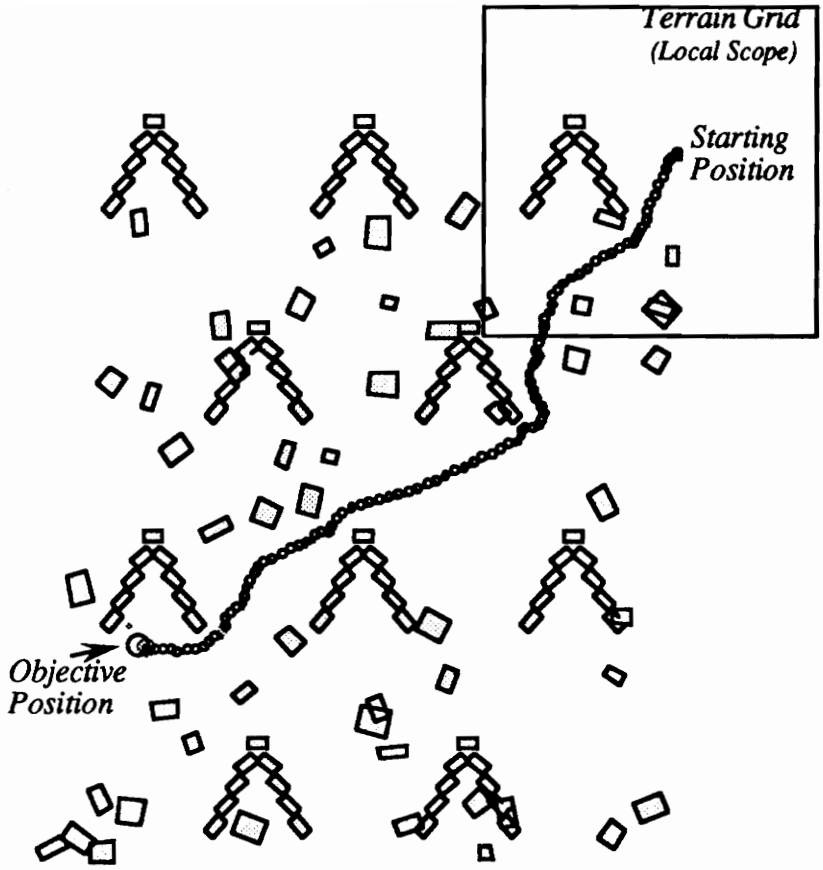
LNav directing a robot to an objective position

Figure 5.14



Moving in an objective direction in the presence of significant sensing and acting uncertainty

Figure 5.15



Moving in an objective position in the presence of significant sensing and acting uncertainty

Figure 5.16

5.5. Final Words

LNav is implemented in Lisp on a Macintosh II. The LNav Coordinator and the boulder modeling system presented above are highly specialized pieces of code. The majority of the code implements NaTs, the combining function, and the spatial modeling framework described in Chapter 4. There are approximately 10,000 lines of code; of that only about 500 were required to implement routines specific to controlling a robot in Boulder World. The implemented system operates in near "real time"; requiring 3 real

seconds for every 1 simulation second to produce the results shown in Figures 5.13, 5.14, 5.15, and 5.16 (this time includes operation of a sensing system simulation). The details involved in the production of the simulation trace in Figure 5.13 are presented in Appendix A. All of the figures in this dissertation containing either a gradient or a simulation trace (except for Figure 2.1) were produced by either running the entire LNav system or by creating a NaT-based navigation plan and running the combining function at the positions implied by the figure.

Chapter Six

6. Final Words

6.1. Summary

In order to move around in the world, the robot requires a mechanism which determines the way that the robot should proceed at any given moment in time. Such mechanisms have commonly addressed this need in one of two ways: through the creation of detailed plans intended to accomplish some specific task (e.g., to reach the water fountain proceed: "forward 1.22 meters, right 12 degrees, then forward 0.87 meters"), or without creating any plan, but rather continually transforming the current sensor state into actions based on some set of *fixed* internal goals. The first approach has the advantage of allowing the robot to act in the service of explicit and changing navigation tasks (e.g., "go to Joe's office"). However, techniques that have been developed to generate such plans typically eliminate any connection between the resulting plan and the environmental constraints which shaped the plan's formation. Furthermore, due to the world's uncertainties, any such detailed plan will become invalid shortly after execution of the plan begins. This is a result of the fact that the plan is not connected to the environmental constrains which drove its creation; thus, there is no way to modify one portion of the plan without recreating the entire plan from scratch. The second approach makes no assumptions about the world; instead, systems adopting this approach directly transform sensor readings into actuator commands. Such an approach is, by its very definition, highly connected to the robot's perception of the world. As a result, the approach is robust in the face of many of the world's uncertainties. However, because the approach provides

no mechanism for the creation and/or manipulation of plans, this approach by itself is unable to accommodate changing high level goals.

Quite clearly the answer lies somewhere between these two approaches. Navigation plans are needed to direct the robot's actions in service of changing high level goals. However, any such plan must be highly flexible so that it can be modified to remain consistent with the robot's perception of the world. With such a plan, a robot can both respond to the world's uncertainties as well as accomplish high level navigation tasks. One of the earliest approaches to robot navigation comes very close to this ideal. Potential fields can be used to create flexible navigation plans by associating an attractive force to the goal position and repulsive forces to the obstacles in the environment. The plan is used to provide the robot's low level control loop with a preferred direction of travel from any given position. The flexibility of this approach results from two of its characteristics: the preferred direction of travel at any position is determined independent of all other positions, and the structures composing the plan are independent of one another. These two characteristics allow a potential field plan to be easily modified in order to remain consistent with the robot's perception of the world. While potential field-based navigation plans are well connected to the robot's perception of the world, there is no explicit mechanism for controlling the creation of the gradient other than by changing the goal position. As a result, potential fields and similar approaches have met with limited success when employed to control the actions of a robot operating in service of specific navigation tasks.

Recent work in planning and reacting has shown that a robot's ability to accommodate the world's uncertainties can be greatly enhanced if the robot is given the ability to characterize the relationship between the known environmental state and the current task (i.e., the current situation). When the robot is able to characterize the situation, known techniques can be employed to further progress on the current task. Consider a situation in which the task is to travel up a hallway containing a "static" object located along the right wall. Presented with such a situation, the robot might reasonably accomplish the task by simply passing the obstacle to its left and continuing up the hallway. Such an analysis can be made with only the most rudimentary of metric information, and no information about the obstacle's identity. It is enough to know that the obstacle is "static", that it is located to the right side of the hallway, and that there is adequate space to the left of the obstacle. The reason that the solution to the situation seems so obvious is that it is a commonly-encountered situation for which a solution is known. The belief that most situations

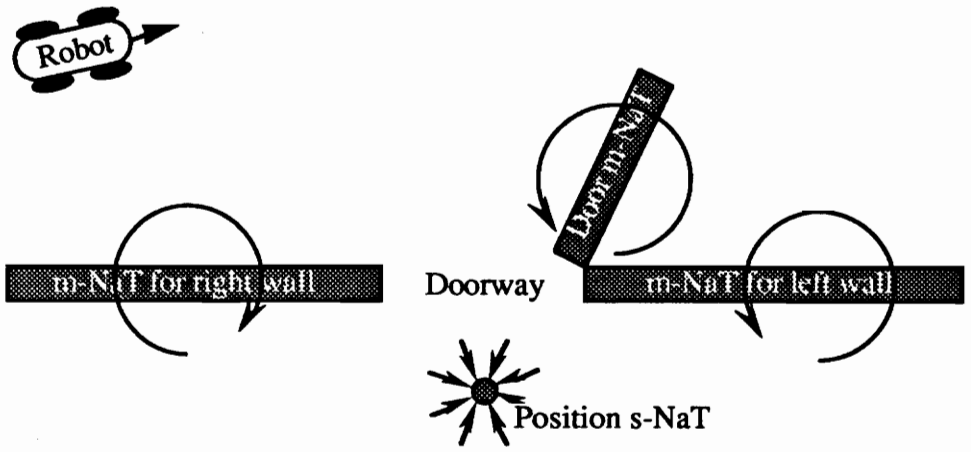
encountered can be handled by applying known solutions is at the heart of this view towards artificial intelligence.

The ideas presented here were inspired by a robot's need for navigation plans, the simplicity of the potential field and like approaches for controlling a robot's actions, and the use of known solutions to handle commonly encountered situations. To support this, a technique for constructing navigation plans that allows the relationship between the navigation task and the environmental constraints to be made explicit has been developed. Central to the technique is the observation that when one is trying to go somewhere and an object is in the way, there are two ways to accommodate the constraint: by passing the obstacle to its left or by passing the obstacle to its right. Furthermore, the critical aspect of navigation plans for mobile robots is not so much the actual space-time trajectory the robot takes, but rather the intended relationship between the robot's actions and the obstacles in the environment. For example, given the task of entering a doorway, the navigation plan might be as simple as "pass between the right and left door jambs". With respect to the navigation plan, the details of the way the plan gets transformed into actuator commands are unimportant as long as the process can respond quickly to changes in the navigation plan.

Navigation Templates (or NaTs) are introduced as building blocks for constructing navigation plans. There are two types of NaTs: those used to characterize the navigation task, referred to as substrate NaTs (or s-NaTs); and those used to model obstacles, referred to as modifier NaTs (or m-NaTs). S-NaTs support accomplishment of a particular navigation task by providing a gradient field which at every position defines the local direction of travel most directly in service of the navigation task. M-NaTs are used to modify the basic scheme encoded in the s-NaT's gradient to accommodate the presence of obstacles in the environment. As such, each m-NaT represents the physical extent of the obstacle and has a spin (either clockwise or c-clockwise) indicating the intended relationship between the robot's actions and the obstacle modeled by the m-NaT. Together, an s-NaT and a number of m-NaTs represent the elements from which a particular navigation plan is built.

Consider the structure of a NaT-based navigation plan for passing through a doorway. To form the foundation of the navigation plan, a position s-NaT is placed on the far side of the doorway indicating that the navigation task is to move over to that position. In addition

to the s-NaT, three m-NaTs are needed: one for each of the walls forming the doorway and one to represent the door (see Figure 6.1). The necessary relationship between the walls and the accomplishing of the navigation task is easy enough to identify in that, when passing forward through any doorway, the motion will be clockwise with respect to the right wall and c-clockwise with respect to the left wall. Thus, each of the m-NaTs used to construct this navigation plan is annotated with the appropriate spin direction. Because the door is to the robot's left at the doorway, it is treated in a similar fashion as the left wall and given a c-clockwise spin. The resulting NaT-based navigation plan is shown in Figure 6.2.

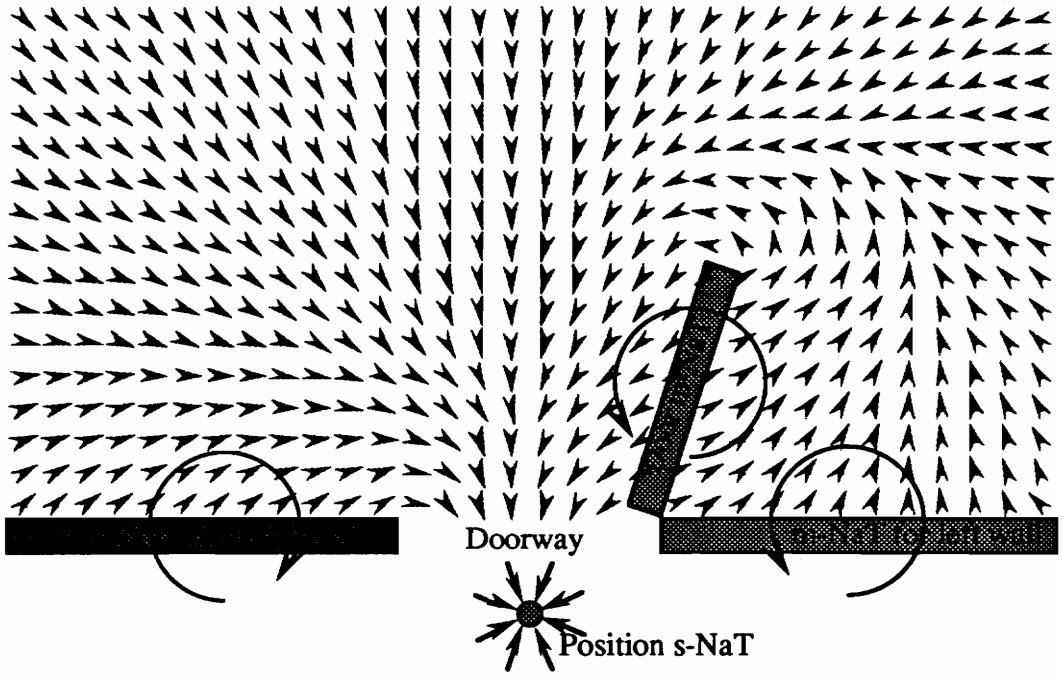


NaT-based navigation plan for entering a doorway

Figure 6.1

Because a NaT-based navigation plan provides only a rough plan for accomplishing a navigation task (e.g., "proceed clockwise around the box and c-clockwise around the fountain, ..."), it cannot be used directly to guide the robot's low level control loops which generate the robot's motions in the world. To provide the robot with immediate and continuous guidance, a combining function is developed. For any given position, the combining function examines the relative influences of all the NaTs in the current navigation plan to identify a preferred direction of travel. Figure 6.2 shows the result of running the combining function at a number of positions, using the above navigation plan for moving through a doorway. This approach has the advantages of the potential field and similar approaches, in that the computation needed to determine the gradient's direction at

any particular position is independent of all other positions, making it flexible in the face of the world's uncertainties. However, because the NaTs provide an explicit mechanism for characterizing the way the robot should proceed past obstacles (i.e., an m-NaT's spin), most of the problems associated with the potential field approaches are avoided.



Gradient produced by the combining function

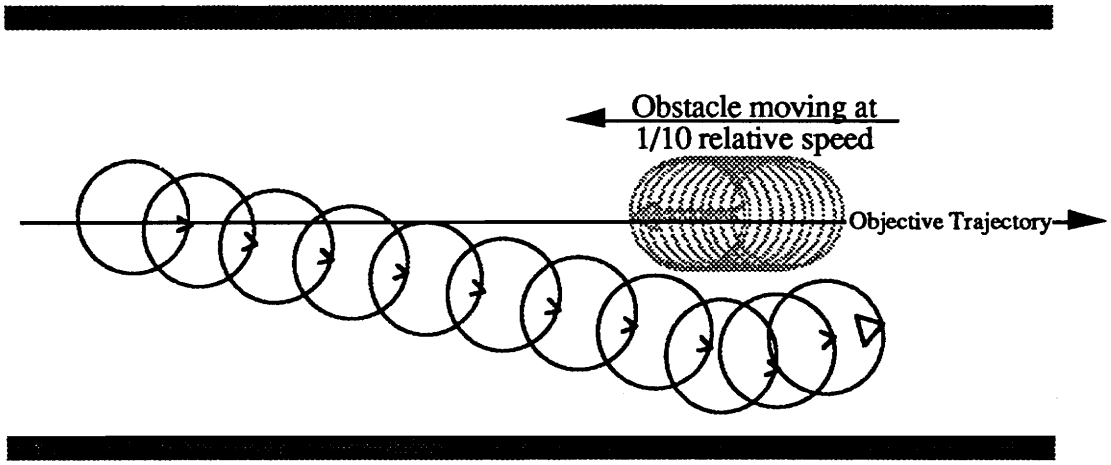
Figure 6.2

6.2. Dynamic Worlds

With respect to navigation, the idea that the objects in the world can be categorized as either static or dynamic is questionable at best. The term static, as applied to an object in the environment, has typically implied that the object will not move. The problem lies not in the definition, but rather in the assumptions made by many navigation systems based on the definition. The most typical of these assumptions is that because something is static it will not change its position. While this may seem tautological, it is not, because an object's

position is part of the robot's perception of the world and unless the robot is omniscient the perceived position of the object is subject to the uncertainties associated with sensing and acting. The result is that, with respect to the robot's perception of the world, there are no "static" objects. Objects which may be known to be incapable of motion will change their perceived position as the robot moves through the world or senses its surroundings. Thus, the design of a navigation system that is to be part of a robot whose perception of the world will never be exact must, to a certain degree, be founded on the idea that all objects in the world are dynamic.

The conception of NaTs is based on the fact that the robot's perception of the world will be continually changing and, as a result, a navigation plan must be flexible so that it can be kept valid. As an example of the flexibility of a NaT-based navigation plan, consider a situation in which the robot must proceed up a hallway which contains a single centrally located obstacle. Furthermore, assume that the navigation plan indicates the obstacle should be passed in a c-clockwise fashion. Now consider the impact on the situation if, as the robot is moving up the hallway, the m-NaT associated with the obstacle must be shifted towards the robot in order for the plan to remain consistent with the robot's perception of the world (whether the motion is real or the result of sensor imprecisions is irrelevant). Figure 6.3 shows a simulation trace of a NaT-based control loop presented with the aforementioned situation. Even though the environment changes after the navigation plan's construction, because it is connected to the robot's perception of the world, it is not invalidated by the obstacle's apparent motion.



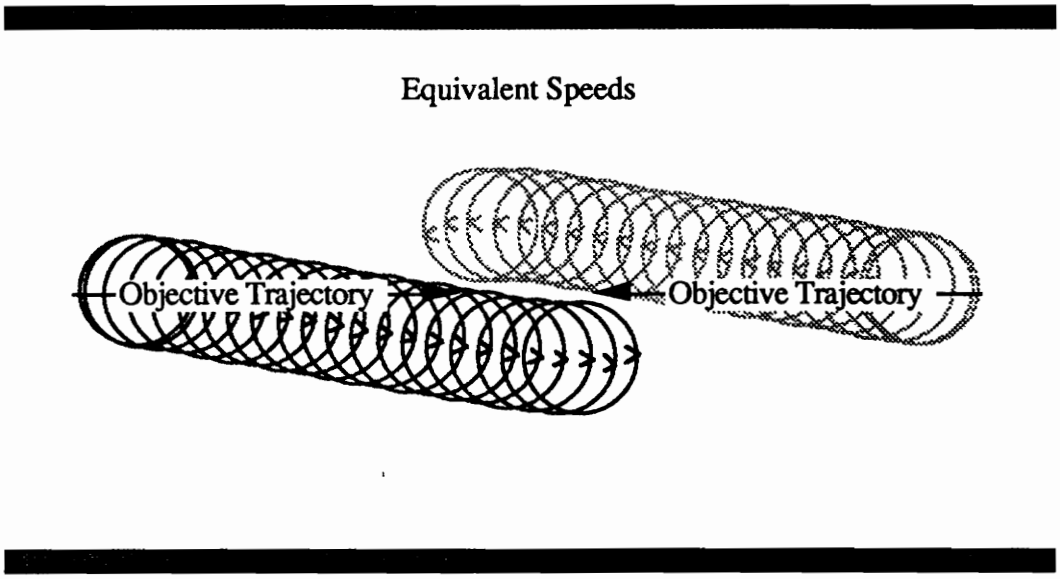
Avoiding a slowly moving "static" object

Figure 6.3

A fundamental aspect of the world in which we live is that it is populated with moving objects: other people, cars, bikes, boats, etc. A robot able to function in a dynamic world must be endowed with a navigation system that is capable of controlling the actions of the robot in the presence of such dynamic objects. Work in robot navigation for dynamic worlds has generally fallen into a trap. This trap is formed when the navigation system makes assumptions about the actions of other agents that must remain true for extended periods of time in order for the navigation plan is to work properly {Kant84, Reif85, Erdmann86, Fujimura86, Lamadrid86, Slack87b, Fujimura88, Fujimura88b, Parker88, Slack88}. Such assumptions are typical of approaches which create detailed plans for the way in which the robot is to proceed through the changing world. Such detailed plans are generally computationally expensive to construct and must be based on assumptions about the actions of other agents in order for the plan to guide the robot's actions for some period of time into the future. However, other agents are not under the control of the robot and are free to act in service of their changing goals; thus, any assumption made about their actions cannot be too specific as uncertainties in both the robot's actions and the other agent's actions will quickly invalidate such assumptions. Hence, approaches that construct detailed plans to guide a robot's actions are doomed when operating in environments in which other agents are in pursuit of unknown goals. While there is clearly a need to make some assumptions about the actions of the other agents, it is important that plans based on such assumptions be highly flexible so that as the assumptions change the plan can be easily modified to keep pace.

NaTs support operation in dynamic worlds by allowing the navigation relationship between a dynamic obstacle and the robot to be characterized. Once again, consider a situation in which a robot has been assigned the task of proceeding up a hallway. However, this time there is another robot proceeding down the hallway towards the first robot. A robot presented with such a situation might reason, based on social convention, that if it passes the other robot to the right the other robot will attempt to do likewise, requiring no further analysis of the situation other than tracking the position of the other robot to ensure that the assumption remains true. Figure 6.4 shows the simulation traces of two robots in such a situation. Each robot is given a navigation plan constructed from an s-NaT directing the robot in the appropriate direction along the hallway, as well as m-NaTs for the right and left walls. In addition, each of the robots appears in the other's navigation plan as an m-NaT which should be passed in a c-clockwise fashion. The maximum

velocity of robots in the depicted situation is equivalent. Each frame of the simulation trace indicates a position where the combining function updates the robot's preferred direction of travel. The result shows the way the robots pass each other in the hallway without reasoning explicitly about the other's actions; rather, they make the assumption that they will pass each other in a c-clockwise fashion and allow the details to unfold at execution time.

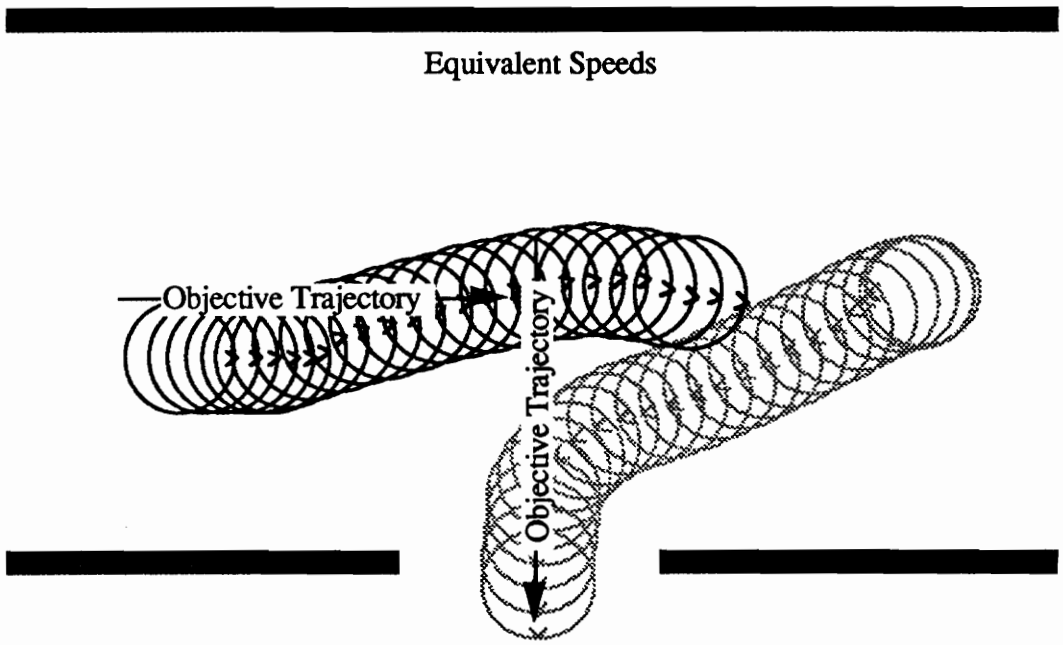


Modeling dynamic objects with NaTs

Figure 6.4

As a further example of the way situational analysis provides a basis for powerful heuristics and the way NaTs can be used to control the actions of a robot operating in a dynamic world, consider the situation depicted in Figure 6.5. One robot is proceeding up the hallway (i.e., from left to right in the figure) and another robot is proceeding in the opposite direction down the hallway. However, rather than hedging to its right, the other robot begins to fade to its left, towards the door. An analysis of the situation yields the assumption that the robot intends to pass through the doorway, and therefore should be

passed in a clockwise fashion[1]. In this situation, the robot moving towards the doorway assumes that it can reach the doorway without impeding the progress of the other robot; therefore, it should be passed in a clockwise fashion. The simulation traces of the figure show the actions of the two robots as guided by such NaT-based navigation plans. These examples indicate that operation in dynamic worlds can be accomplished without making detailed assumptions about the actions of other agents in the world.



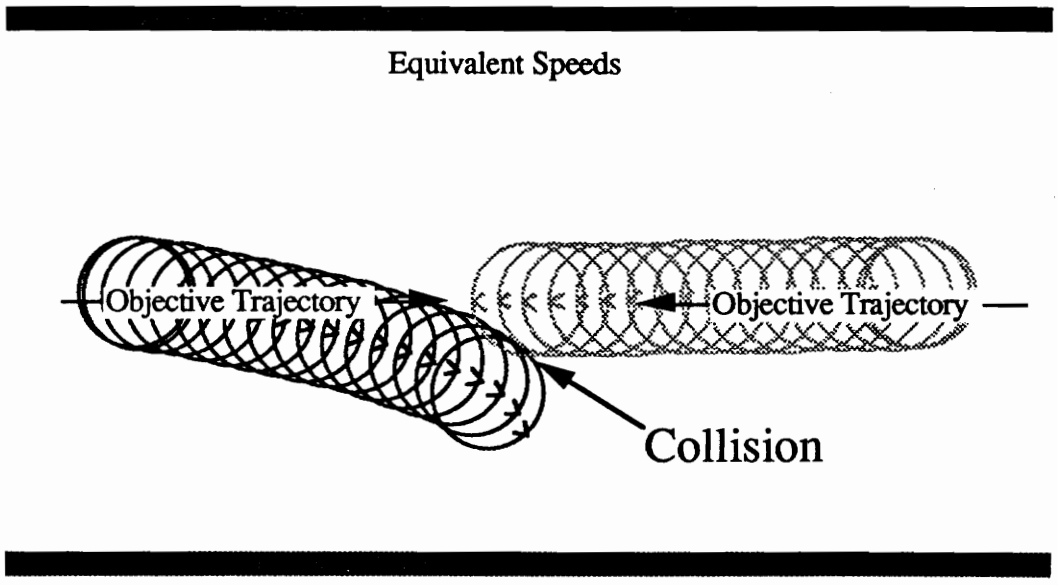
Making assumptions about the intentions of another agent

Figure 6.5

NaT-based navigation plans that are properly connected to the robot's perception of the world support the robot's actions in a dynamic world. However, there are still a number of situations in which the direct approach of simply associating an m-NaT directly with the position of a dynamic object fails to produce the desired results. Consider the situation depicted in Figure 6.6, in which the robot is once again proceeding up the hall. This time the other robot is unaware of the first robot's presence and therefore does not modify its

[1]Such an analysis is beyond the scope of this work and represents a difficult problem; however, NaTs allow such an analysis to remain at a symbolic level.

actions to accommodate that robot's presence; the first robot is unable to compensate adequately, and a collision results. Thus, while it is possible for a robot using NaTs to avoid the need for explicit temporal projection in many situations, there are still many situations in which such projection is necessary. When the speed of the robot is slow in comparison to the speed of other objects, the need for temporal projection becomes even more apparent. One method supporting this need would locate the m-NaT of a dynamic obstacle at the position projected to be the closest approach between the obstacle and the robot. Thus, rather than updating the position of the m-NaT to remain synchronized with the current position of the obstacle, the system maintaining the navigation plan would update the position of the m-NaT to be at the projected position of closest approach between the robot and the obstacle. Ideally, the position of the obstacle and the associated m-NaT would be identical at the position of the obstacle's closest approach. This would allow the robot to anticipate the actions of another agent and to position itself to accommodate such behavior. While not necessarily an easy proposition, it would greatly increase the level of goal-directed behavior manifested by the robot.



Collision between the robot and an oblivious agent

Figure 6.6

Although NaTs and the combining function provide some support for navigating through dynamic spaces, the real control of the situation must reside at the level of the

system creating the navigation plan. Consider turning onto a busy street. In such a situation it is likely that the solution would require the robot to remain stopped until there is a sufficient gap in the flow of traffic to allow it to proceed. Such a scheme is clearly beyond the abilities of a NaT-based navigation plan, as NaTs are used for forming plans for motion and being stopped until some condition becomes true is simply not supported. Operation in truly dynamic worlds is a vastly complex problem and will require the efforts of many different techniques working in a coordinated fashion.

6.3. Limitations and Extensions

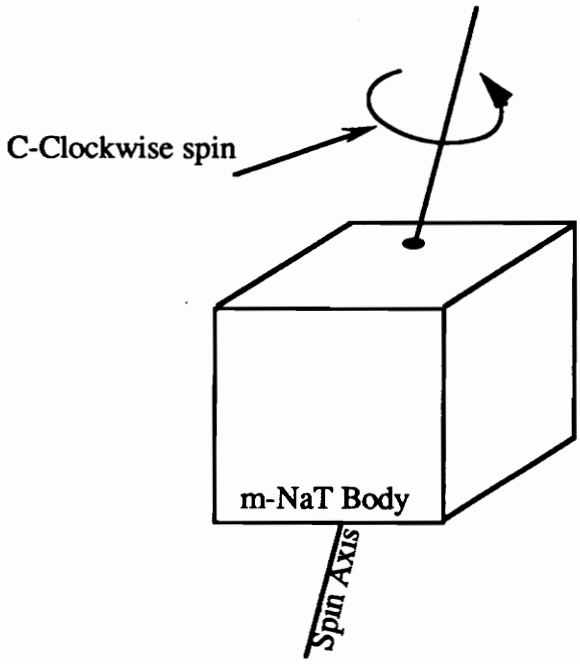
Much has been said about the usefulness of the NaTs and the combining function; however, there are some extensions and limitations to NaTs and the combining function that should be mentioned.

6.3.1. Extending to Three Dimensions

In many situations, a robot navigation system can control a robot's actions through the use of two-dimensional models and navigation plans. In other situations, the robot's operating space is three-dimensional and cannot be reduced to a simpler two-dimensional problem without severely limiting its ability to function in the environment. Outer Space or underwater applications in which a robot might be expected to operate in a free-flying fashion are examples of such domains. Thus, extending NaTs to allow modeling of three dimensional environments represents an obvious extension to this work that merits investigation.

A number of issues must be addressed if such an extension is to be made. One involves the creation of three dimensional s-NaTs. This is simple enough, as the idea of a gradient field which flows through two dimensions is easily generalized to three dimensions. The next step is a bit more involved, as m-NaTs must be generalized to three-dimensions. First is the obvious extension of an m-NaT's body from two to three-dimensions. Second, the concept of an m-NaT's spin must be generalized to some three-dimensional analog. In two-dimensions, spin indicates in which direction the robot should move around an obstacle modeled by the associated m-NaT. A three-dimensional

equivalent of this concept is less clear, since rather than just two ways around an obstacle, there is an infinite number of ways around an obstacle. Intuitively, however, when one thinks about moving around an obstacle in three dimensions, it is common to talk about moving around a particular side of the obstacle. Obviously not all three dimensional objects have discernable sides; however, it is possible to associate an axis with an obstacle in order to provide a frame of reference (see Figure 6.7). This axis, referred to as the m-NaT's axis of spin, serves as the basis for defining the spin of the m-NaT, thus providing the navigation system with a mechanism for manipulating navigation plans. Just as in the two-dimensional case, the navigation plans characterize the relationship between an obstacle and the navigation task and constrain the robot's actions to moving relatively clockwise or c-clockwise with respect to the defined axis of spin. While the details with respect to the combining function must still be worked out, it does appear that the fundamentals of Navigation Templates can be generalized to three dimensions.

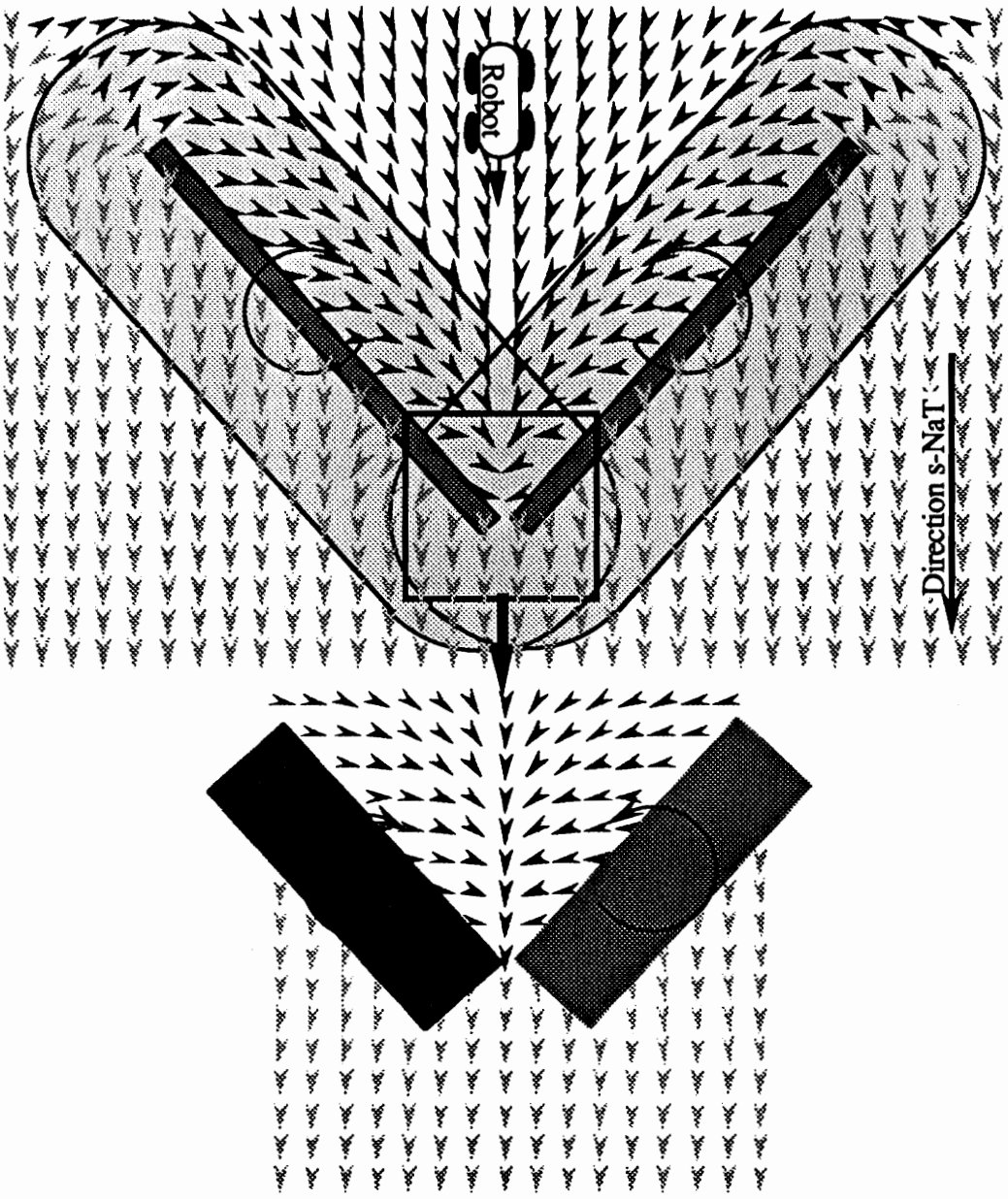


A 3-dimensional m-NaT and its axis of spin

Figure 6.7

6.3.2. A Tight Squeeze

Consider the situation depicted in Figure 6.8, in which a direction s-NaT and two m-NaTs characterize a navigation plan. The spins of the m-NaTs direct the robot through an opening between the two m-NaTs that is too narrow for the robot to pass. Presented with such a plan, the combining function provides a gradient that flows through the opening anyway. The reason is that the combining function, as defined, takes no account of the robot's physical dimensions and makes no attempt to determine if the gradient characterizes a viable path. While this may seem to be a serious flaw it is important to remember that a NaT-based navigation plan is a plan for motion and assumes that puzzles involving the robot's interactions with the obstacles in the world have already been dealt with. It is possible to modify the combining function in order to account for the robot's physical dimensions and generate no gradient when the navigation plan dictates that the robot move through an opening that is too small. This would be a mistake, however, as there are many situations in which such a plan is required. For example, objects in the environment may react to the robot's presence and create the need for a plan that forces the robot into a tight position so that the environment will change due to the robot's actions. Such a plan might be required even when there are no dynamic objects in world, as the navigation task might simply require that the robot move into a corner between two obstacles. Thus, the fact that the combining function will create gradients that flow into tight spaces is a feature rather than a limitation; the combining function has no knowledge of the bigger picture. However, the problem of detecting when the navigation plan is simply wrong is still an issue which must be addressed. To support the detection of deficient navigation plans, the combining function might be modified to return additional information. For example, the combining function might return the immediate navigation objective identified by the combining function. This information could be used by the system constructing the navigation plan to verify that the identified objective is reasonable.



Gradient flowing through a "local minimum"

Figure 6.8[2]

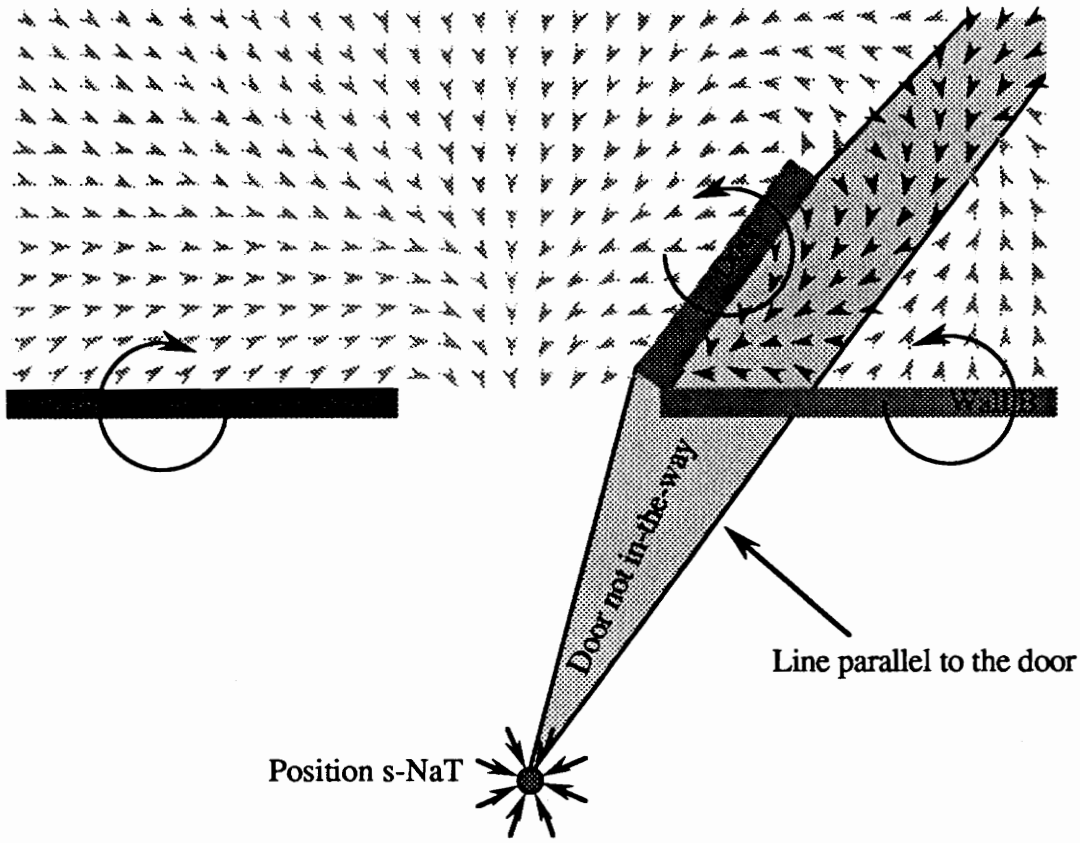
[2]Note that the lower figure is simply a more detailed view of the boxed portion of the upper figure.

6.3.3. M-NaT Independence

As mentioned earlier, NaTs form plans for motion. For this reason the system has the built-in assumption that once an obstacle is not in-the-way, the gradient will flow away from that obstacle. As a result, it is possible to construct situations in which the gradient will flow into a corner when it appears that the navigation plan should not produce such a result. Consider the situation depicted in Figure 6.9, in which a navigation plan to move the robot through a doorway has been constructed. The navigation plan is composed of a position s-NaT and three m-NaTs. The arrows in the shaded region show the gradient to be flowing into the corner behind the door, even through the navigation plan indicates that the gradient should flow around the door. This is a result of the fact that, in the shaded region, the door m-NaT is not in-the-way and, as a result, the identified immediate navigation objective in the region indicates that the gradient should flow between the door and wall B. This problem is eliminated easily enough by locating the objective position of the s-NaT in such a way that it forces the door to be in-the-way when the robot is operating behind the door. However, this is not an intuitive approach to addressing the problem and places significant burden on the system maintaining the navigation plan.

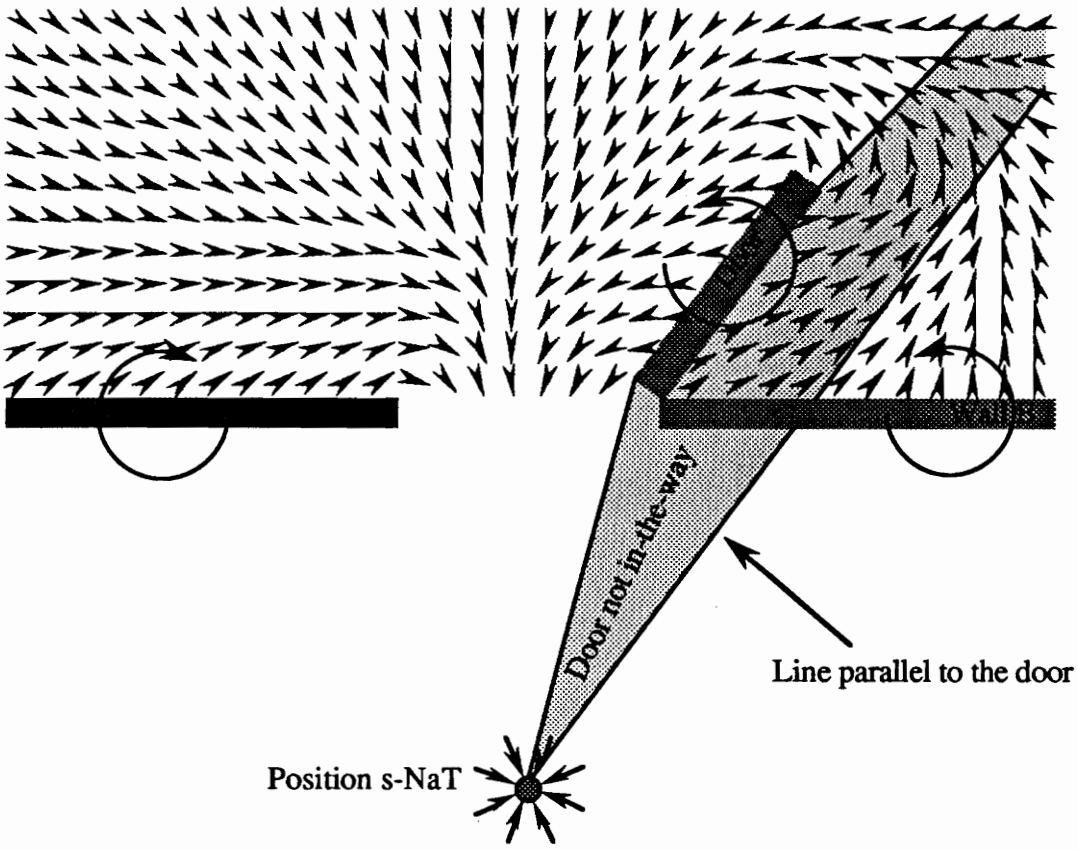
The problem is rooted in the fact that, with respect to accomplishing the navigation task, the door and wall B form a single obstacle. One way to solve this problem is to make the wall and the door a single object; however, due to some assumptions made by the combining function, all m-NaTs must be composed of convex geometric objects. A second way to handle this problem would be to allow the system maintaining the navigation plan to determine when an m-NaT is in-the-way or not in-the-way. Such a change would require only trivial changes to the combining function, but would significantly increase the responsibilities of the system constructing the navigation plan; thus, it should be avoided. A better technique to address this situation is to introduce the concept of m-NaT groups, and to modify the combining function to provide a mechanism that forces m-NaTs that might not otherwise be considered to be in-the-way to be so considered as such whenever a single member of an m-NaT group is in-the-way. Therefore, by grouping together the door and wall B, in all of the shaded region wall B is in-the-way, thus the door is also in-the-way, resulting in the elimination of the problem. To verify the feasibility of this approach a primitive ability to form m-NaT groups has been implemented. Figure 6.10 shows a gradient formed by the combining function when the door and wall B are grouped

together. Unlike the gradient in Figure 6.9, the gradient in Figure 6.10 flows up and around the door. While there are still a number of details to be worked out, grouping m-NaTs allows the system constructing the navigation plan to simply be concerned with the formation of logical groupings of m-NaTs leaving the details to combining function.



Existence of an unwanted local minimum

Figure 6.9



Logically grouping the Wall B and the Door

Figure 6.10

6.4. Conclusions

A robot operating in the world will inevitably be confronted with the problem of keeping its plans up-to-date as its perception of the world changes. For plans that are executed over long time scales (e.g., where to go to college or what to have for breakfast the next morning) it may be appropriate to simply discard the current plan and rethink a problem every time new information becomes available. However, when the time scales become very much shorter, there may not be time to rethink the entire problem every time a new fact is uncovered or information about a known fact is refined. In such situations, it is

more appropriate to attempt to make the current plan work by changing selected aspects in order to avoid having to operate for a period of time without a plan. If a plan can be constructed so that it is flexible and accommodating to new or refined information, then it is possible to avoid the shortcomings of functioning in the world without a plan.

Previous approaches addressing the local navigation problem have typically either ignored the need for flexible navigation plans or the need for navigation plans altogether. Inflexible local navigation plans suffer from the problem that every time the robot's perception of the world changes, the plan becomes invalid. In any realistic situation in which the environment has not been engineered for the robot, the robot's perception of the world changes continually and without warning. Therefore, unless such navigation plans are computationally trivial to construct, such approaches are inappropriate for guiding the robot's actions as it moves through its local environment. The other common approach to local navigation is to omit the navigation plan altogether. Such an approach typically directs the robot's actions by operating directly on the robot's current perception of the world, and thus always remains in synchronization with the robot's perception of the world. However, such approaches do not provide enough focus to successfully accomplish local navigation tasks in any sort of a directed fashion. The best alternative is to build navigation plans that are tightly bound to the robot's perception of the world, thereby ensuring that as the robot's perception of the world changes, so does its current navigation plan.

Navigation plans built with NaTs are flexible, making them robust in the face of the world's uncertainties. For the system maintaining a navigation plan built with NaTs, there are five basic events that may require the navigation plan to be modified:

- Changes in the perceived position of an obstacle
- Changes in the navigation plan as it relates to a particular obstacle
- Incorporation of new obstacles into the navigation plan
- Removal of old obstacles from the navigation plan
- Change in the navigation task

Navigation Templates and the combining function support each of these modifications. The first involves changes in the perceived position of an obstacle. This is easily handled,

as the combining function makes no assumptions as to the position of the m-NaTs between updates. Hence, when the perceived position of an obstacle changes, it is a simple matter to update the navigation plan by changing the position and/or orientation of the m-NaT in the navigation plan representing the obstacle. Sometimes the navigation plan itself requires modification because some aspect(s) of the plan are in error. For example, consider a situation in which the robot is proceeding up a hallway with a known obstacle in the middle. The original navigation plan indicated that the obstacle should be passed in a c-clockwise fashion. As the robot approaches the obstacle, the perceived position of the obstacle places it much further to the right than the original estimates. At this point it is possible that the system maintaining the navigation plan may need to reconsider its decision to pass the obstacle in a c-clockwise fashion. A NaT-based navigation plan can easily accommodate this change by simply reversing the spin on the m-NaT associated with the obstacle. Because no state is carried between runs of the combining function, modifying the navigation plan causes no side-effects other than changing the flow of the resulting gradient. The addition and elimination of constraints from NaT-based navigation plans is as easy as inserting or deleting an m-NaT from the current navigation plan. In the case of elimination, the combining function can be used to aid in determining when an m-NaT has become obsolete by announcing whenever an m-NaT is both not in-the-way and the robot is outside of its danger zone. The final issue involves changing the s-NaT of the current navigation plan. This is a much more subtle issue, since simply changing from one s-NaT to another might require that the spins of the m-NaTs be reexamined in light of the change. This basically involves the creation of an entirely new navigation plan; however, the m-NaTs that must be incorporated in the new plan are already known. The problem of changing from one s-NaT to another must be handled carefully or the robot may spend some time without any guidance. One way to do this would be to construct the new plan before the old one becomes useless. By double buffering the navigation plans the transition between navigation tasks can be made smoothly.

Action without plans is undirected. Plans created without regard for action in the real-world are useless. Only by creating plans for action can a robot act in a directed fashion.

Bibliography

{Adelson84} Adelson, E., H., Anderson, C., H., Bergen, J., R., Burt, P., J., Ogden, J., M, Pyramid Methods in Image Processing, in the *RCA Engineer Magazine*, Nov/Dec 1984.

{Agre87} Agre, P., E., Chapman, D., Pengi: An Implementation of a Theory of Activity, in the *Proceedings of AAAI 87*, AAAI, Seattle Washington, pp. 268-272, July 1987.

{Agre88a} Agre, P., E., Chapman, D., *What Are Plans For?*, A.I. Memo 1050, MIT, Artificial Intelligence Laboratory, September, 1988.

{Agre88b} Agre, P., E., *The Dynamic Structure of Everyday Life*, Ph.D. Thesis, MIT, 1988.

{Anderson85} Anderson, C., H., Burt, P., J., van der Wal, G., S., Change Detection and Tracking Using Pyramid Transform Techniques, in the *Proceedings of the SPIE Conference on Intelligent Robots and Computer Vision*, 1985.

{Andress88} Andress, K., M., Kak, A., C., Evidence Accumulation & Flow of Control in a Hierarchical Spatial Reasoning System, in the *AI Magazine*, Volume 9, Issue 2, Summer, 1988, or Technical Report 88-9, School of Electrical Engineering, Purdue University, 1988.

{Andrews83} Andrews, R. J., *Impedance Control and a Framework for Implementing Obstacle Avoidance in a Manipulator*, Masters thesis, Massachusetts Institute of Technology, 1983.

{Arkin87} Arkin, R., C., Motor Schema Based Navigation for a Mobile Robot: An Approach to Programming by Behavior, in the *Proceedings IEEE International Conference on Robotics and Automation*, Raleigh, North Carolina, pp. 264-271, 1987.

{Arkin89} Arkin, R., C., Three-Dimensional Motor Schema Based Navigation, in the *Proceedings of the NASA Conference on Space Telerobotics*, Pasadena, California, Jan 31 - Feb 2, 1989.

{Avnaim88} Avnaim, F., Boissonnet, J., D., Faverjon, B., A Practical Exact Motion Planning Algorithm for Polygonal Objects Amidst Polygonal Obstacles, in the *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 1656-1661, 1988.

{Ayala85} Ayala, D., Brumet, P., Navazo, I., Object Representation by Means of Non-Minimal Division Quadtrees and Octrees, in the *ACM Transactions on Graphics*, Volume 4, Number 1, pp. 41-59, January 1985.

{Brooks82} Brooks, R. A., Solving the Find Path Problem by a Good Representation of Free Space, in the *Proceedings of AAAI 82*, AAAI, pp. 381-386, 1982.

{Brooks83} Brooks, R., A., Lozano-Perez, T., A Subdivision Algorithm in Configuration Space for Findpath with Rotation, in the *Proceedings of the International Joint Conference on Artificial Intelligence*, Karisruhe West Germany, August 1983.

{Brooks86a} Brooks, R., A., A Robust Layered Control System for a Mobile Robot, in the *IEEE Journal of Robotics and Automation*, Volume RA-2, Number 1, pp. 14-23, March 1986.

{Brooks86b} Brooks, R., A., Connell, J., H., Asynchronous Distributed Control System for a Mobile Robot, in the *Proceedings of the SPIE Conference on Mobile Robotics*, pp. 77 - 84, 1986.

{Brooks88} Brooks, R., A., *A Robot that Walks: Emergent Behaviors from a Carefully Evolved Network*, MIT Technical Report, September, 1988.

{Burt88} Burt, P., J., Algorithms and Architectures for Smart Sensing, in the *Proceedings of the Image Understanding Workshop*, DARPA, 1988.

{Charniak86} Charniak, E., A Neat Theory of Marker Passing, in the *Proceedings of AAAI 86*, AAAI, pp. 584-588, 1986.

{Chatila85} Chatila, R., Laumond, J., Position referencing and Consistent World Modeling for Mobile Robots, in the Proceedings of the International Conference on Robotics and Automation, IEEE, pp. 138-145, 1985.

{Connell87} Connell, J., H., Creature Design with the Subsumption Architecture, in the *Proceedings of the International Joint Conference on Artificial Intelligence*, Milano, Italy, 1987.

{Dijkstra59} Dijkstra, E., W., A Note on Two Problems in Connexion with Graphs, *Numerische Mathematik*, Volume 1, pp. 269-271, 1959.

{Dorst88} Dorst, L., Trovato, K., Optimal Path Planning by Cost Wave Propagation in Metric Configuration Space, in the *Proceedings of the SPIE Conference*, Volume 1007, Cambridge, Mass, pp. 186-197, November, 1988.

{Doshi88} Doshi, R., S., Lam, R., White, J., E., Region Based Route Planning: Multi-Abstraction route planning based on intermediate level vision processing, in the *Proceedings of the 7th SPIE Conference*, Cambridge Mass, Nov 1988.

{Elfes87} Elfes, A., Sonar-based Real-World Mapping and Navigation, in the *IEEE Journal of Robotics and Automation*, Volume RA-3, Number 3, pp. 249-265, June 1987.

{Elfes89} Elfes, A., A Tessellated Probabilistic Representation for Spatial Robot Perception and Navigation, in the *Proceedings of the NASA Conference on Space Telerobotics*, Pasadena, California, January 31 - February 2, 1989.

{Erdmann86} Erdmann, M., Lozano-Perez, T., On Multiple Moving Objects, in the *IEEE Proceedings of the International Conference on Robotics and Automation*, pp. 1419-1424, 1986.

{Faverjon87} Faverjon, B., Tournassoud, P., The Mixed Approach for Motion Planning: Learning Global Strategies from a Local Planner, in the *International Joint Conference on Artificial Intelligence*, Milano Italy, pp. 1131-1137, August 1987.

{Feng89} Feng, D., *Satisficing Feedback Strategies for Local Navigation of Autonomous Mobile Robots*, Ph.D. Dissertation, Department of Electrical and Computer Engineering, Carnegie Mellon University, May, 1989.

{Fikes71} Fikes, R., E., Nilsson, N., J., STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving, in *Artificial Intelligence*, Volume 2, pp. 189-208, 1971.

{Firby87} Firby, R., J., An Investigation into Reactive Planning in Complex Domains, in the *Proceedings of AAAI 87*, Seattle, Washington, pp. 202-206, July 1987.

{Firby89a} Firby, R., J., *Adaptive Execution in Complex Dynamic Worlds*, Technical Report 672, Yale University, Department of Computer Science, Ph. D.. Thesis, January, 1989.

{Firby89b} Firby, R., J., Task Directed Sensing, in the *Proceedings of SPIE's conference on Sensor Fusion*, Philadelphia, Pennsylvania, 1989.

{Fujimura86} Fujimura, K., Samet, H., *A Hierarchical Strategy for Path Planning Among Moving Obstacles*, Technical Report CS-TR-1736, Center for Automation Research University of Maryland, November 1986.

{Fujimura88} Fujimura, K., Samet, H., *Time-Minimal Paths Among Moving Obstacles*, Tech Report CS-TR-2124, Computer Science Department, University of Maryland, October, 1988.

{Fujimura88b} Fujimura, K., Samet, H., Path Planning Among Moving Obstacles using Spatial Indexing, in the *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 1662-1667, 1988.

{Gat89} Gat, E., Firby, R., J., Miller, D., P., Planning for Execution Monitoring on a Planetary Rover, in the *Proceedings of the NASA Conference on Spacecraft Operations Automation and Robotics*, Houston Texas, 1989.

{Gat90} Gat, E., *Towards an Integrated Control Architecture for an Autonomous Mobile Robot*, paper in preparation.

{Gat90b} Gat, E., Slack, M., G., Firby, R., J., Miller, D., P., Path Planning and Execution Monitoring for a Planetary Rover, in the *Proceedings of the 1990 IEEE Conference on Robotics and Automation*, 1990.

{Georgeff87} Georgeff, M., P., Lansky, A., L., Reactive Reasoning and Planning, in the *Proceedings of AAAI 87*, Seattle Washington, pp. 677-682, July 1987.

{Kaelbling87} Kaelbling, L., P., Rex: A symbolic Language for the Design of Parallel Implementation of Embedded Systems, in the *Proceedings of the AIAA Conference on Computers in Aerospace*, Wakefield, Massachusetts, 1987.

{Kambhampati86} Kambhampati, S., Davis, L., S., Multi-Resolution Path Planning for Mobile Robots, in the *IEEE Journal of Robotics and Automation*, Volume RA-2, No. 3, pp. 135-145, September 1986.

{Kanayama85} Kanayama, Y., Miyake, N., Trajectory Generation for Mobile Robots, in the *Proceedings of the Third International Symposium on Robotics Research*, ISRR-3, Paris, October, 1985.

{Kant84} Kant, K., Zucker, S., W., Planning Collision-Free Trajectories in Time-Varying Environments: A Two-Level Hierarchy, in the *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 1644-1649, 1988.

{Keisey88} Keirsey, D., M., Krozel, J., Payton, D., W., Scale-Space Representations for Flexible Automated Terrain Reasoning, in the *Proceedings of the US Army Symposium on Artificial Intelligence Research in Military Battlefield Environments*, El Paso, Texas, November, 1988.

{Khatib85} Katib, O., Real-Time Obstacle Avoidance for Manipulators and Mobile Robots, in the *Proceedings of the International Conference on Robotics and Automation*, pp. 500-505, 1985, and in the *International Journal of Robotics Research*, Volume 5, Number 1, pp. 90-98, 1986.

{Kheradpir88} Kheradpir, S., Thorp, J., S., Real-Time Control of Robot Manipulators in the Presence of Obstacles, in the *IEEE Journal of Robotics and Automation*, Volume 4, Issue 6, pp. 687-698, 1988.

{Khosla88} Khosla, P., Volpe, R., Superquadratic Artificial Potentials for Obstacle Avoidance and Approach, in the *Proceedings of the International Conference on Robotics and Automation*, IEEE, pp. 1778-1784, 1988.

{Kirkpatrick79} Kirkpatrick, D., *Efficient Computation of Continuous Skeletons*, IEEE Publications, 1979.

{Kriegman88} Kriegman, D., J., Binford, T., O., Generic Models for Robot Navigation, in the *Proceedings of the 1988 International Conference of Robotics and Automation*, pp. 746-751, 1988.

{Kriegman89} Kriegman, D., J., Triendl, E., Binford, T., O., Sterio Vision and Navigation in Buildings for Mobile Robots, in the *IEEE Journal of Robotics and Automation*, Volume 5, Issue 6, pp. 792-803, 1989.

{Krogh84} Krogh, B., H., A Generalized Potential Field Approach to Obstacle Avoidance Control, in the *Proceedings of the Robotics International Robotics Research Conference*, Bethlehem PA, August 1984.

{Krogh86} Krogh, B., H., Thorpe, C., E., Integrated Path Planning and Dynamic Steering Control for Autonomous Vehicles, in the *Proceedings of the International Conference on Robotics and Automation*, San Francisco California, pp. 1664-1669, April 1986.

{Lamadrid86} de Lamadrid, J., G., Avoidance System for Moving Obstacles, in the *Proceedings of the SPIE Conference on Advances in Intelligent Robotics Systems*, 1986.

{Lamadrid87} de Lamadrid, J., G., *Obstacle Avoidance Heuristic for Three Dimensional Moving Objects*, technical report 87-16, Computer Science Department, Institute of Technology University of Minnesota, February 1987.

{Laumond83} Laumond, J. P., Model Structuring and Concept Recognition: Two Aspects of Learning for a Mobile Robot, in the *Proceedings of the International Joint Conference on Artificial Intelligence*, Karisruhe West Germany, pp. 839-841, August 1983.

{Laumond87} Laumond, J., Finding Collision-Free Smooth Trajectories for A Non-Holonomic Mobile Robot, in the *International Joint Conference on Artificial Intelligence*, Milano Italy, August 1987, pp. 1120-1123.

{Levitt87} Levitt, T., S., Lawton, D., T., Chelberg, D., M., Nelson, P., C., in the *Proceedings of the AAAI Conference 1987*, pp. 689-694, 1987.

{Lozano-Perez79} Lozano-Perez, T., Wesley, M., A., An Algorithm for Planning Collision Free Paths among Polyhedral Obstacles, in the *Communications of the ACM*, Volume 22, Number 10, pp. 560-570, October 1979.

{Lozano-Perez83} Lozano-Perez, T., Spatial Planning: A Configuration Space Approach, in the *IEEE Transactions on Computers*, Vol C-32, Number 2, pp. 108-120, February 1983.

{Lozano-Perez86} Lozano-Perez, T., A Simple Motion Planning Algorithm for General Robot Manipulators, in the *Proceedings of the 5th AAAI Conference*, Philadelphia, pp. 626-631, 1986.

{Marra88} Marra, M., Dunlay, R., T., Mathis, D., Terrain Classification Using Texture for the ALV, in the *Proceedings of SPIE's 7th Conference on Mobile Robots III*, Volume 1007, 1987.

{Mazer87} Mazer, E., Jones, J., Lanusse, A., Lozano-Perez, T., O'Donnell, P., Tournassoud, P., Using Automatic Robot Programming for Space Telerobotics, in the *Proceedings of the JPL Workshop on Space Telerobotics*, Vol III, pp. 139-150, Pasadena, California, January, 1987.

{McDermott84} McDermott, D. V., Davis, E., Planning and Execution Routes Through Uncertain Territory, *Artificial intelligence*, Volume 22, pp. 107-156, 1984.

{Miller85} Miller, D., P., *Planning by Search Through Simulation*, Doctoral Thesis, Yale University, October 1985.

{Miller87} Miller, D., P., Slack, M., G., Efficient Navigation Through Dynamic Domains, in the Proceedings of Workshop on Spatial Representation and Multi-Sensor Fusion, St. Charles Illinois, pp. 230-239, October 1987.

{Miller88} Miller, D. P., A Task and Resource Scheduling System for Automated Planning, in the *Annals of Operations Research*, Volume 12, pp. 169-198, 1988.

{MingWang88} MingWang, C., Location Estimation and Uncertainty Analysis for Mobile Robots, in the *Proceedings of the 1988 IEEE International Conference on Robotics and Automation*, Philadelphia, PA, pp. 1230-1235, April 24-29, 1988.

{Moravec83} Moravec, H., P., The Stanford Cart and the CMU Rover, in the Proceedings of the IEEE, Volume 71, Issue 7, July, 1983.

{Moravec85} Moravec, H., P., Elfes, A., High Resolution Maps from Wide Angle Sonar, in the *Proceedings of the IEEE international Conference on Robotics and Automation*, St. Louis Missouri, pp. 116-121, March 1985.

{Moravec87} Moravec, H. P., Certainty Grids for Mobile Robots, in the *Proceedings of the Workshop on Space Tele-Robotics*, JPL Publication 87-13, Volume I, pp. 307-312, Pasadena, California, January 1987.

{Nguyen84} Nguyen, V., *The Find-Path Problem in the Plane*, MIT AI Technical Report 760, 1984.

{Nilsson71}.Nilsson, N. J.,*Problem-Solving Methods in Artificial Intelligence*, New York: McGraw-Hill, 1971.

{O'Rourke84} O'Rourke, J., Convex Hulls, Voronoi Diagrams, and Terrain Navigation, in the *Proceedings of the Ninth William T. Pecora Memorial Remote Sensing Symposium*, IEEE, USGS, NASA, ASP, Sioux Falls, South Dakota, pp 358-360, 1984.

{Oommen86} Oommen, B., J., Iyengar, S., S., Rao, N., S., V., Kashyap, R., L., *Robot Navigation in Unknown Terrains Using Learned Visibility Graphs: Part 1, The Disjoint Convex Obstacle Graph*, School of Computer Science Carleton University, Ottawa Canada, Technical Report 86, February 1986. Note to appear in the IEEE Journal of Robotics and Automation.

{Oskard90} Oskard, D., N., Hong, T., -H., Shaffer., C., A., Real-Time Algorithms and Data Structures for Underwater Mapping, to appear in the *IEEE Transactions on Systems, Man, and Cybernetics*, 1990.

{Payton88} Payton, D., W., Internalized Plans: A Representation for Action Resources, in the *Proceedings of the Workshop on Representation and Learning in an Autonomous Agent*, Lagos, Portugal, November, 1988.

{Parker88} Parker, L., E., *A Robot Navigation Algorithm for Moving Obstacles*, Masters Thesis, University of Tennessee, Knoxville, June 1988.

{Puri87} Puri, S., Davis, L., S., Two Dimensional Path Planning with Obstacles and Shadows, Technical Report CS-TR-1760, Center for Automation Research University of Maryland, January 1987.

{Rao86} Rao, N., S., V., Iyengar, S., S., Jorgensen, C., C., Weisbin, C., R., Concurrent Algorithms for Autonomous Robot Navigation in an Unexplored Terrain, in the *Proceedings of the International Conference on Robotics and Automation*, San Francisco California, pp. 1137-1144, April 1986.

{Reif85} Reif, J., H., Sharir, M., *Motion Planning in the Presence of Moving Obstacles*, Technical Report 39/85, The Eskenasy Institute of Computer Science, Tel-Aviv University, 1985.

{Rosen70} Rosen, C., A., *An Experimental Mobile Automaton*, Technical Report 39, Stanford Research Institute, July, 1970.

{Rueb87} Rueb, K., D., Wong, A., K., C., Structuring Free Space as a Hypergraph for Roving Robot Path Planning and Navigation,in the *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9 #2, pp. 263-273, March 1987.

{Sacerdoti77}.Sacerdoti, E., D., *A Structure for Plans and Behavior*, American Elsevier Publishing Company, Inc., 1977.

{Samet89} Samet, H., *The Design and Analysis of Spatial Data Structures*, Addison-Wesley Publishing Company, Reading Mass, 1989.

{Samet90} Samet, H., *Applications of Spatial Data Structures*, Addison-Wesley Publishing Company, Reading Mass, 1990.

{Schoppers87} Schoppers, M., J., Universal Plans for Reactive Robots in Unpredictably Environments, in the *Proceedings of the International Joint Conference on Artificial Intelligence*, 1987.

{Shamos75} Shamos, M., I., *Computational Geometry*, Ph.D. Thesis, Yale University, 1975.

{Slack87a} Slack, M., G., *Spatial and Temporal Path Planning*, Masters thesis, Virginia Polytechnic Institute and State University, Spring 1987. Technical Report 87-19, 1987.

{Slack87b} Slack, M. G., Miller D. P., Path Planning Through Time and Space in Dynamic Domains, in the Proceedings of the 10th IJCAI, Milano Italy, pp. 1067-1070, 1987. Also see Technical Report 87-5, Department of Computer Science, Virginia Tech.

{Slack88} Slack, M. G., Planning Paths Through a Spatial Hierarchy: Eliminating Stair-Stepping Effects, in the *Proceedings of SPIE's 7th Conference on Sensor Fusion*, Cambridge Massachusetts, printing pending, 1988.

{Smith} Smith, T., R., Parker, R., E., An Analysis of the Efficacy and Efficiency of Hierarchical Procedures for Computing Trajectories over Complex Surfaces.

{Sussman73} Sussman, G., J., *A Computational Model of Skill Acquisition*, Phd Thesis, Artificial Intelligence Laboratory MIT, Cambridge Massachusetts, August 1973.

{Thorpe84} Thorpe, C. E., Path Relaxation: Path Planning for a Mobile Robot, in *Proceedings of AAAI 84*, AAAI, pp. 318-321, 1984.

{Tournassoud88} Tournassoud, P., Jehl, O., Motion Planning for a Mobile Robot with a Kinematic Constraint, in the *Proceedings of the IEEE International Conference on Robotics and Automation*, pp.1785-1790, 1988.

{Trovato89} Trovato, K., I., Differential A*: An Adaptive Search Method Illustrated with Robot Path Planning for Moving Obstacles & Goals, and an Uncertain Environment, in the *Proceedings of the IEEE Conference on Robotics and Automation*, May, 1989.

{Udupa77} Udupa, S., Collision Detection and Avoidance in Computer Manipulators, in the *Proceedings of the International Joint Conference on Artificial Intelligence*, Cambridge, Mass, pp. 737-748, 1977.

{Wilcox87} Wilcox, W., H., Gennery, D., B., Mishkin, A., H., Cooper, B., C., Lawton, T., B., Lay, N., K., Katzmann, S., P., A Vision System for Mars Rover, in the *Proceedings of SPIE's 7th Conference on Mobile Robots II*, Volume 852, 1987.

Appendix A

An LNav Simulation Run

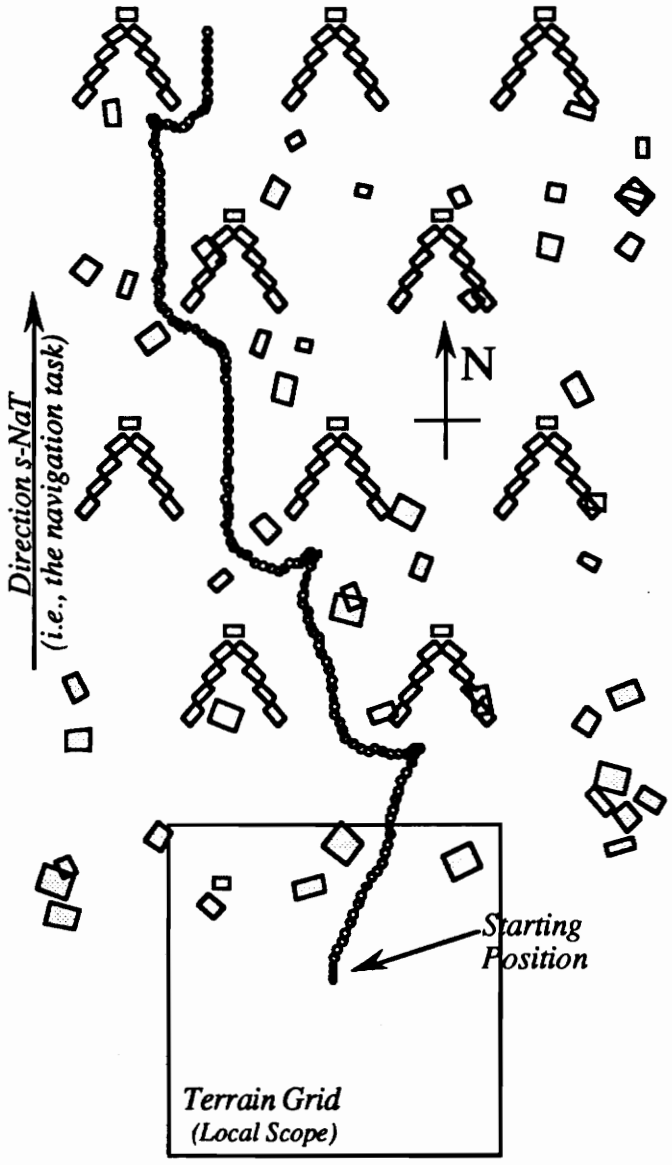
The following details an LNav simulation run. In this example LNav is controlling the actions of a robot as it moves through a boulder field containing local traps. Figure A.1 shows the operating environment and a trace of the robot's position as it moves through the boulder field. The region depicted is approximately 60 meters east to west and 90 meters north to south.

The robot controlled by LNav is a circular synchro-drive robot that is one meter in diameter. The servo loop used to control the robot is identical to the one presented in Chapter 3, where the linear and angular velocities are controlled to be inversely proportional to each other. The maximum linear velocity in this example is 0.5 meters per second and the maximum angular velocity is 0.4 radians per second. Four times a second the combining function is run to update the linear and angular accelerations to be applied to the robot for the intervening 0.25 seconds.

The sensing scheme employs a rectangular terrain port 20 meters wide and 1 meter deep. The sweeping scheme steps the terrain port in one meter increments out to a maximum range in front of the robot of 10 meters. The simulation assumes that the sensing system can update the terrain grid associated with the terrain port in under a second. Thus, the port is advanced to a new position once a second. For this example the sensing system maintained a terrain grid which is 32 meters square with 1.0 meter cells.

Initially, the robot is at rest. LNav is given the navigation task of moving in a northerly direction. This results in the Coordinator creating a new direction s-NaT and passing it to the NaT Manager. The robot's action system then begins to move the robot in response to the guidance that has become available. Because LNav initially has no model of its local surroundings the navigation plan consists simply of the s-NaT. Thus, until

LNav becomes aware of something in its environment that must be avoided the robot simply moves north.

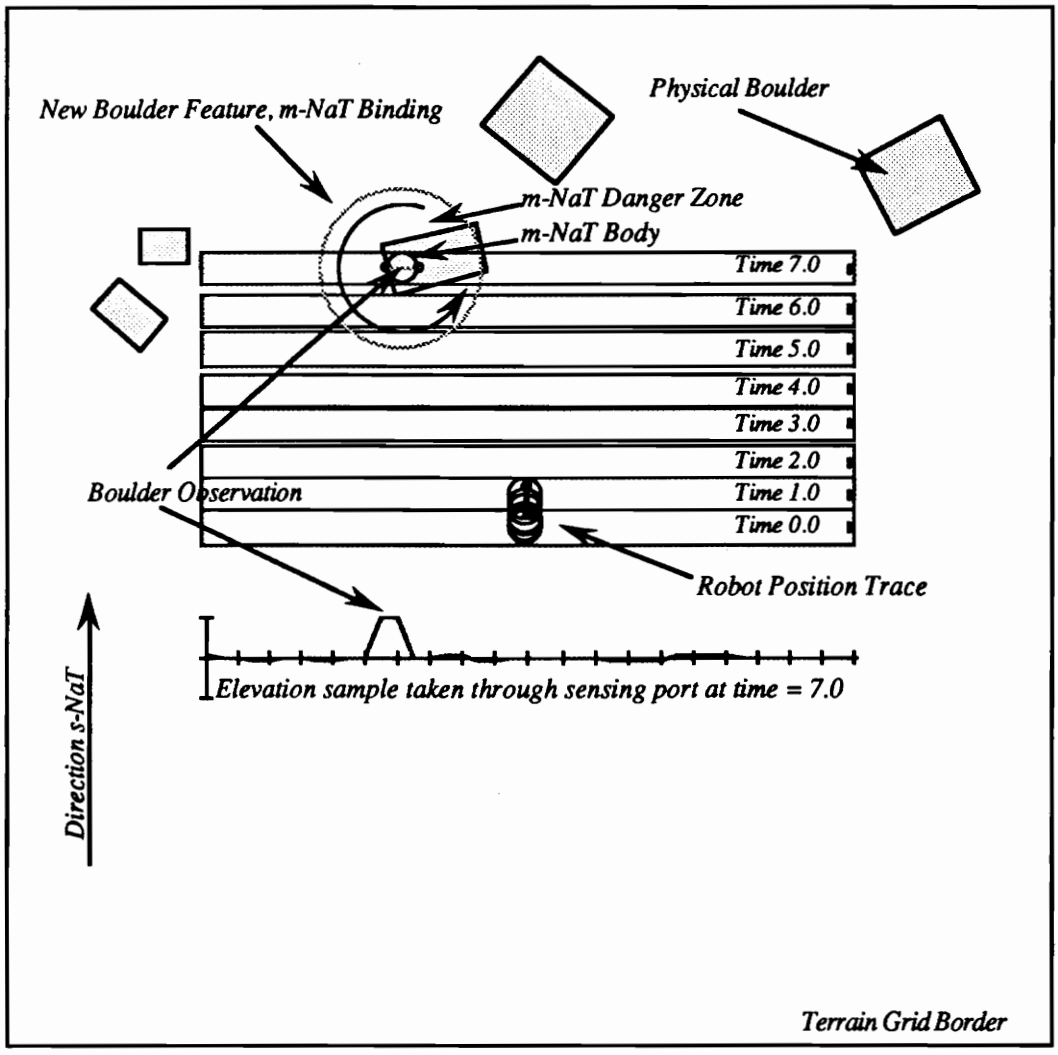


LNav simulation trace

Figure A.1

The first seven seconds of the simulation are shown in Figure A.2. At each step the terrain port's position is advanced one meter further in front of the robot. Initially, the positions of the terrain port are adjacent to one another; however, as the robot's velocity increases, the distance between the ports increases by the distance traveled by the robot in the intervening second. At time step 7.0 a boulder observation is made (indicated in the

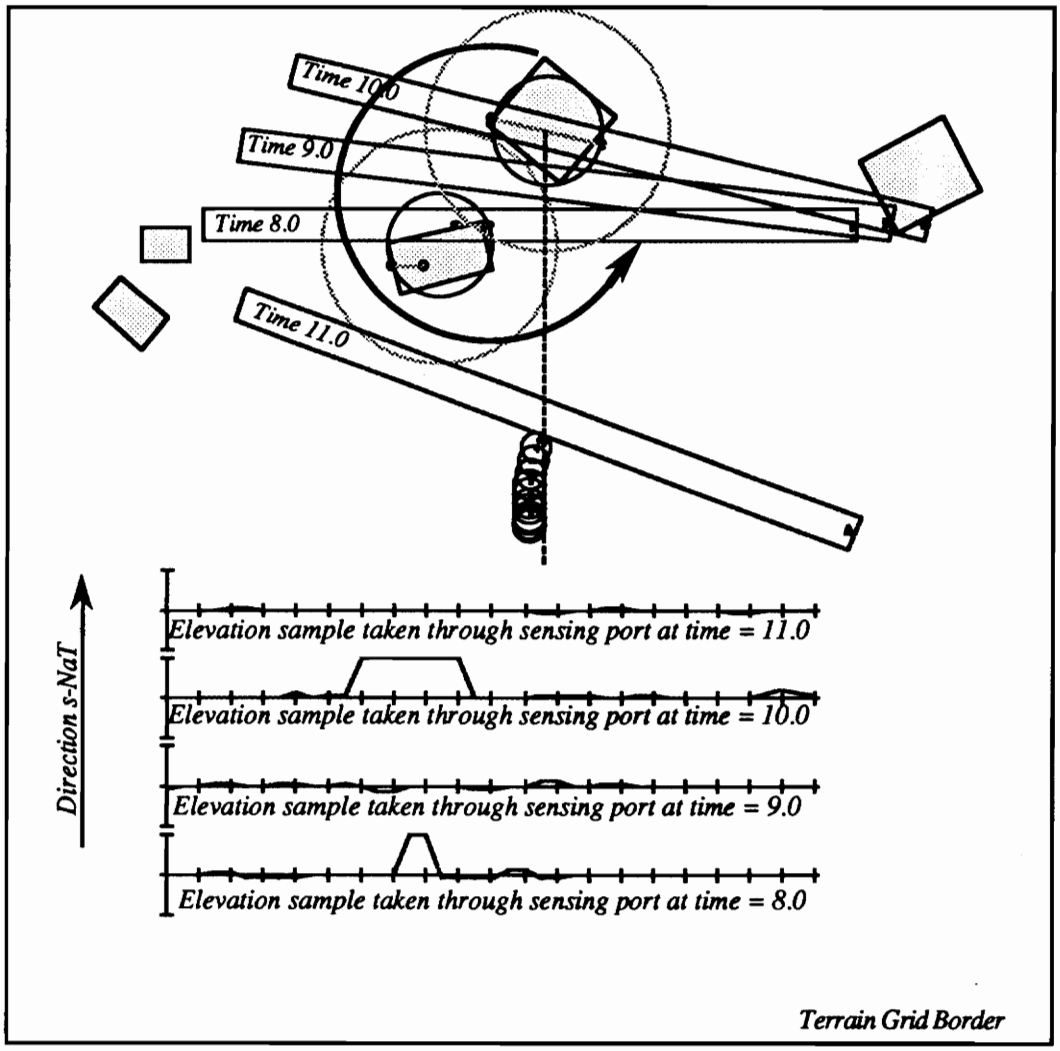
figures by the two small circles connected by a line segment). The observation is given an expiration date of 22.0 (i.e. observations persist for 15 seconds). The new observation causes the Feature Manager to create a new boulder feature in which to stick the observation. The feature manager in turn notifies the Coordinator that a boulder has been located. The Coordinator responds to this activity by creating a new circular m-NaT to associate with the boulder feature. The safe distance for the robot moving through the boulder field is two meters, thus the danger zones on all of the m-NaTs are represented by a circle two meters greater in radius than the radius of the m-NaT. As the boulder represents the first known feature in the world it is placed into its own group. The m-NaT is assigned a type of unspun and passed to the NaT Manager, thus becoming part of the current navigation plan.



Start of the simulation and the first boulder observation

Figure A.2

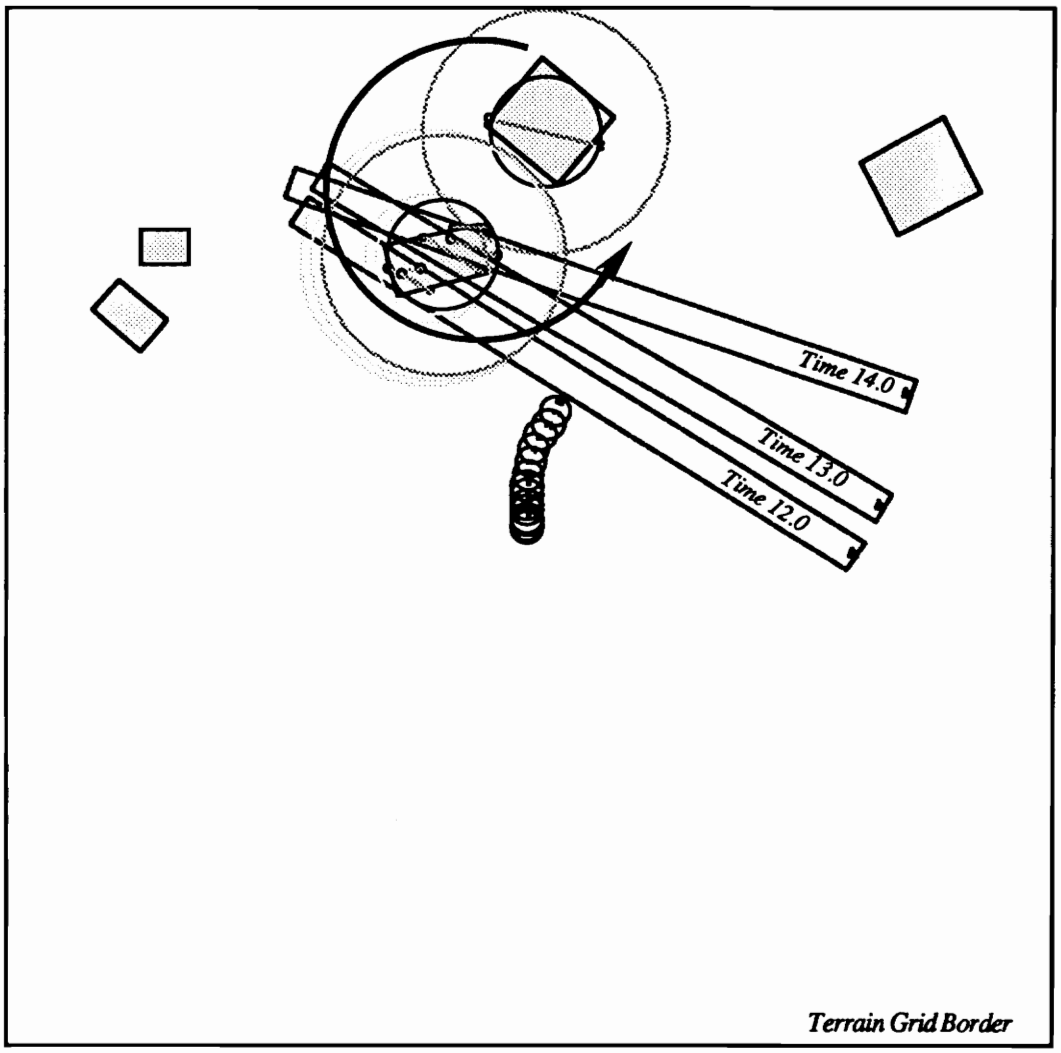
Figure A.3 shows the state of the both the boulder observations as well as the state of the navigation plan at simulation time 11.0. The m-NaT introduced into the navigation plan at time 7.0 takes on a c-clockwise spin as that is the default value assigned to it by the combining function. The addition of the m-NaT into the navigation plan results in the combining function directing the robot toward the c-clockwise nominal bound of the m-NaT. At time step 8.0 a new boulder observation is obtained resulting in the insertion of a boulder observation to the first boulder feature (notice the plots of the elevation sample employed by the filtering function). The Coordinator is notified of the update and the radius and position of the associated m-NaT are adjusted accordingly. At time step 9.0 the filtering function returns no new observations. However, at time step 10.0 the filtering function does locate a new observation and the Feature Manager is notified. The distance between the observation and the nearest boulder indicates that there is enough distance for the robot to pass between the existing boulder and the boulder observation (1.5 meters for this simulation). Hence, the Feature Manager creates a new boulder and reports its existence to the Coordinator. The Coordinator creates a new m-NaT for the feature, then checks the boulder feature's position with respect to the existing boulder group. The result of the check indicates that the distance between the boulders is too small (1.5 meters for this simulation). This prompts the Coordinator to insert the boulder into the existing group, causing its spin to adopt the c-clockwise spin of the group. Once the grouping has been established the new m-NaT is passed to the NaT Manager becoming part of the existing navigation plan. The subsequent time step is 11.0, initiating the beginning of another sensor sweep and the relocation of the terrain port immediately in front of the robot.



Insertion of a boulder into an existing group

Figure A.3

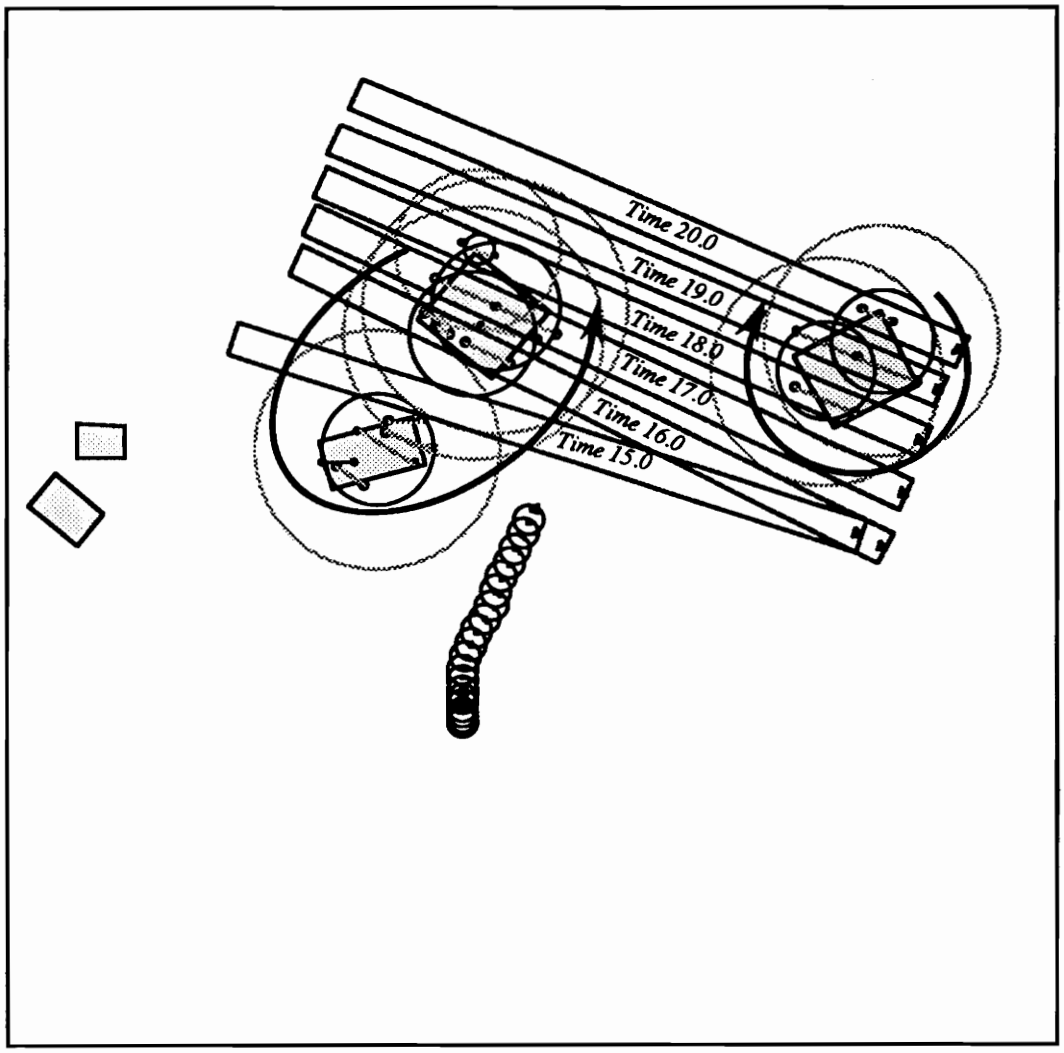
Figure A.4 shows the state of both boulder observations as well as the state of the navigation plan at simulation time 14.0. The next three time steps provide additional observations to further isolate the location of the original boulder feature, indicated by the lightly shaded circles around the m-NaT. The light circles represent the location and radius of the m-NaT at time steps 12.0 and 13.0.



Adding observations to an existing boulder feature

Figure A.4

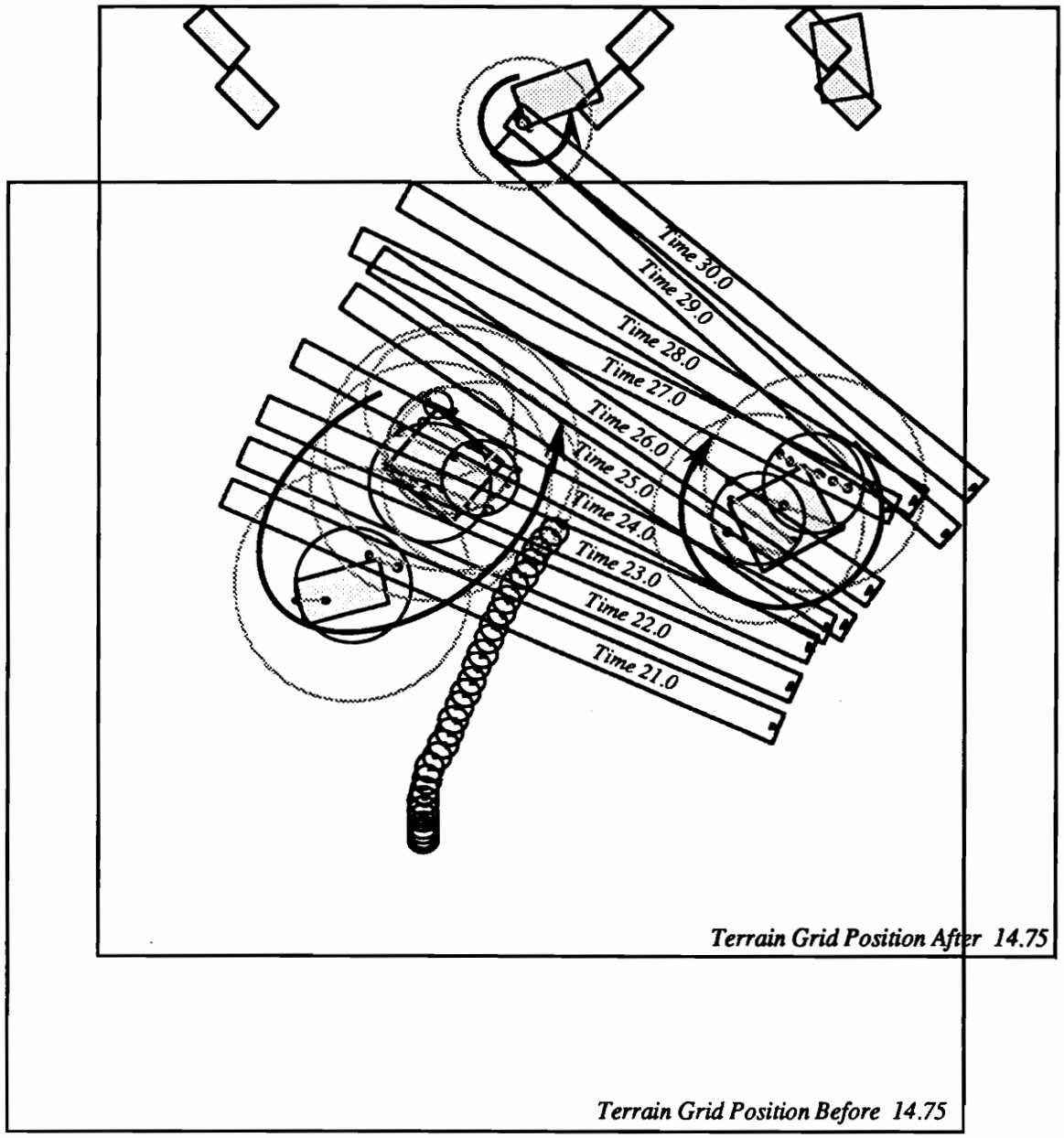
Figure A.5 shows LNav's internal state after continuing the simulation to time 20.0. Notice that a new boulder grouping has been formed and assigned the appropriate c-clockwise default spin.



Advancing the simulation state to time 20.0

Figure A.5

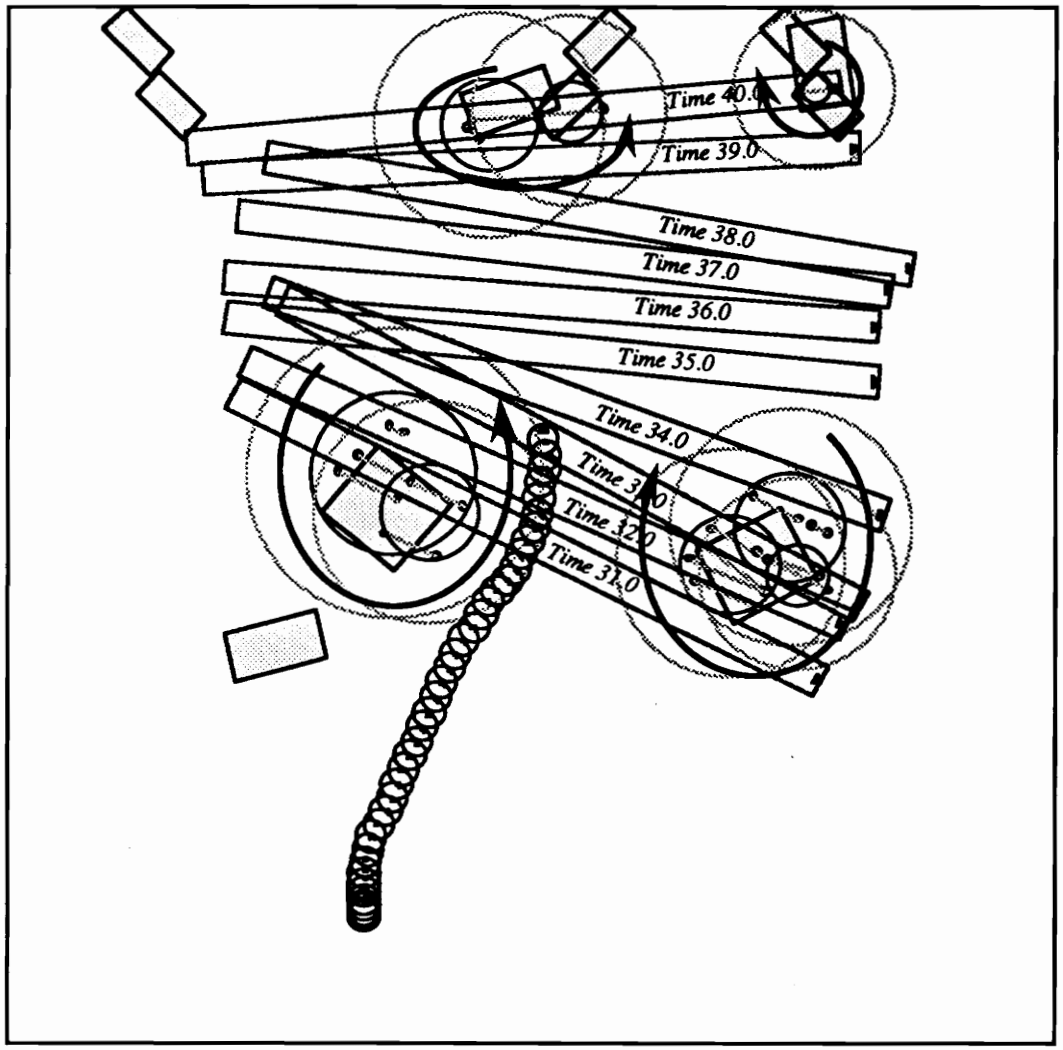
As the simulation advances to time step 30.0 two events of note occur (see Figure A.6). The first happens at 14.75. At this time the robot's position is updated, as it always is every 0.25 seconds; however, the terrain grid's stored up translations exceed its threshold of 6.0 meters resulting in the terrain grid scrolling in order to remain more or less centered with respect to the robot. The other event worth noting happens at time 30.0, when the sensing port is at its maximum range. In particular, the left entrance to the as yet unknown trap is identified. The robot's location with respect to the new boulder feature causes its spin to be set to c-clockwise, directing the robot towards the trap's entrance.



Edge of the trap sighted

Figure A.6

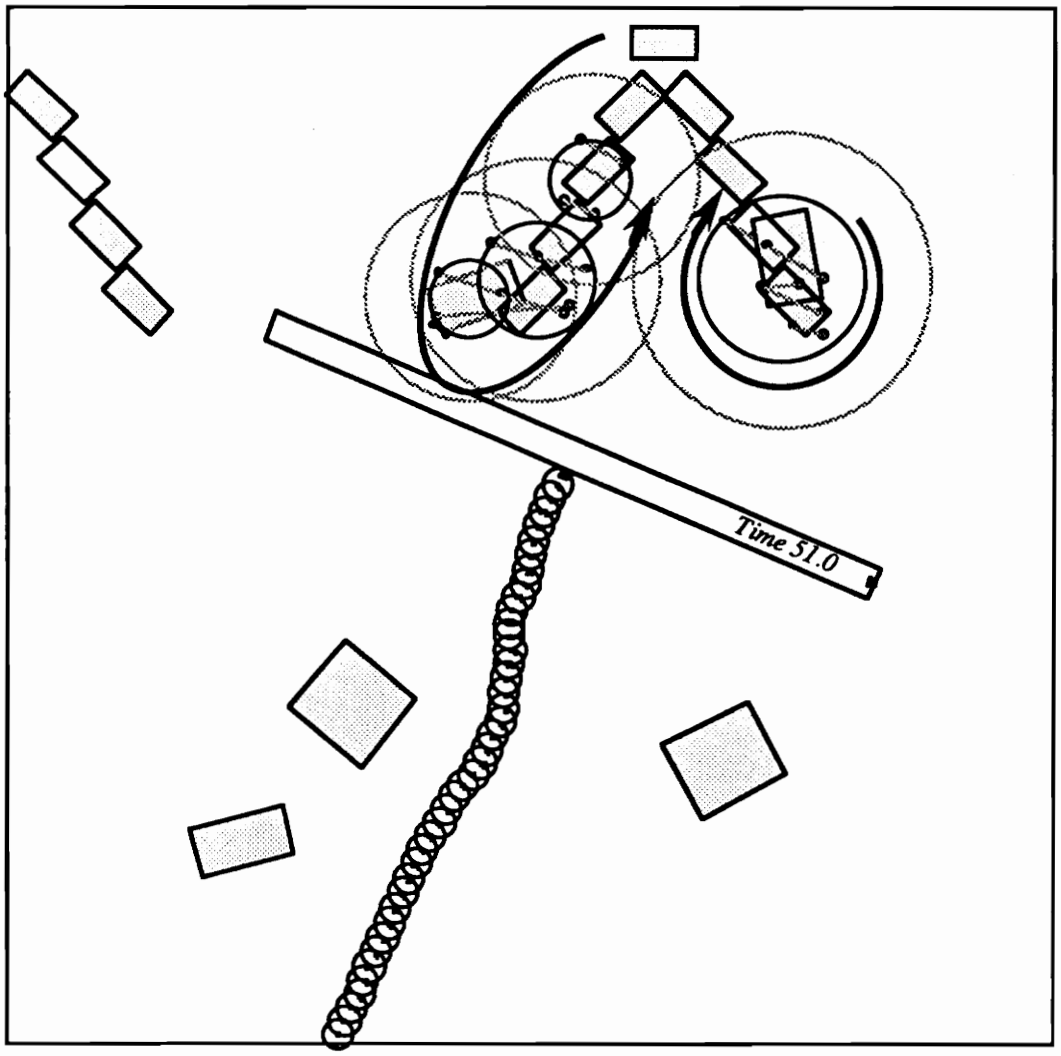
At the end of the next sweep of the sensing port, the right entrance to the trap is located and assigned a clockwise spin. At this point the navigation plan directs the robot between the entrances to the trap (see Figure A.7). Since the sensing port is at its maximum range it is not until later that more of the trap is illuminated.



Setting the trap

Figure A.7

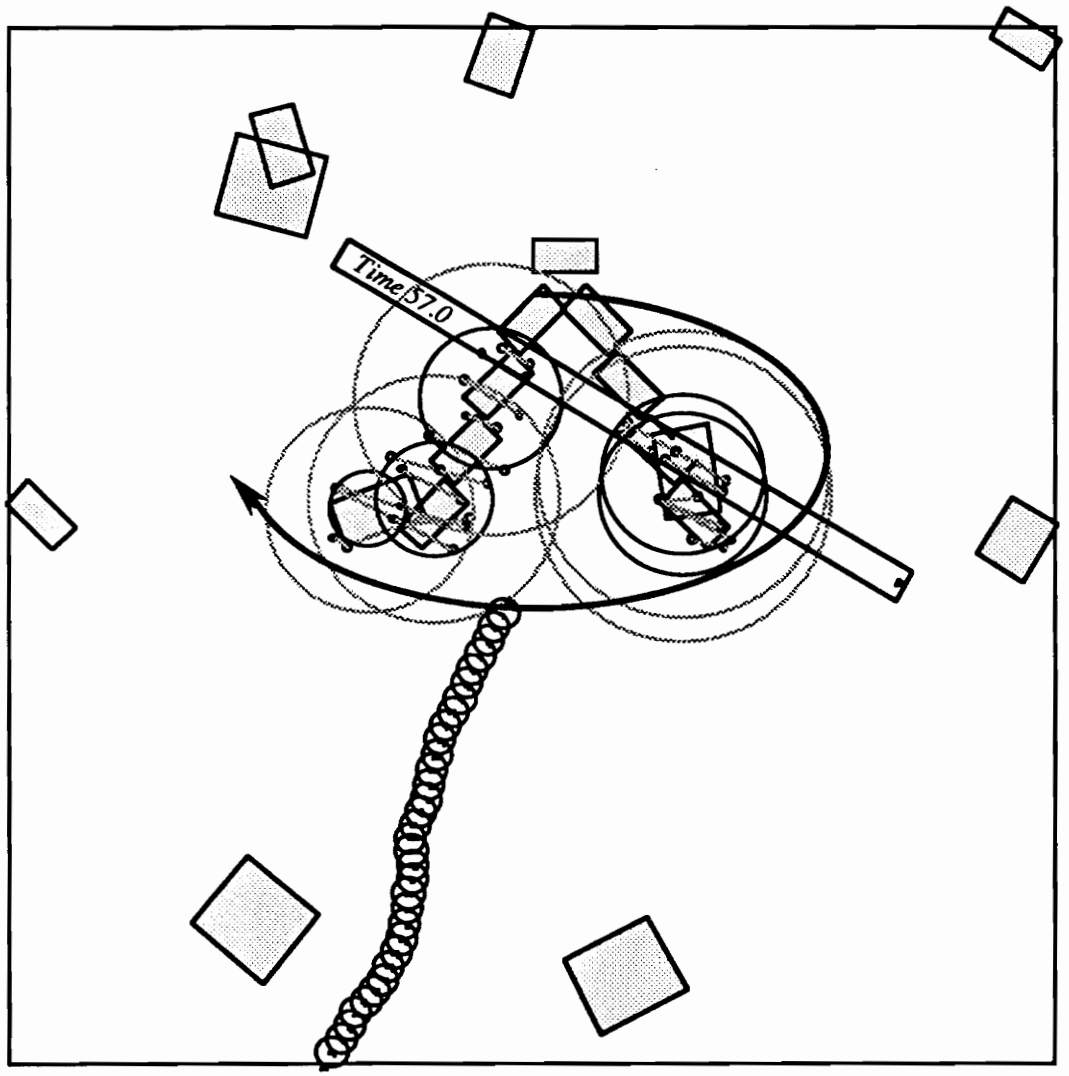
At time 51.0 the sensing scheme has not yet examined the back of the trap; thus, the navigation plan is still directing the robot into the trap (see Figure A.8).



Still operating under the assumption that there is a viable path ahead

Figure A.8

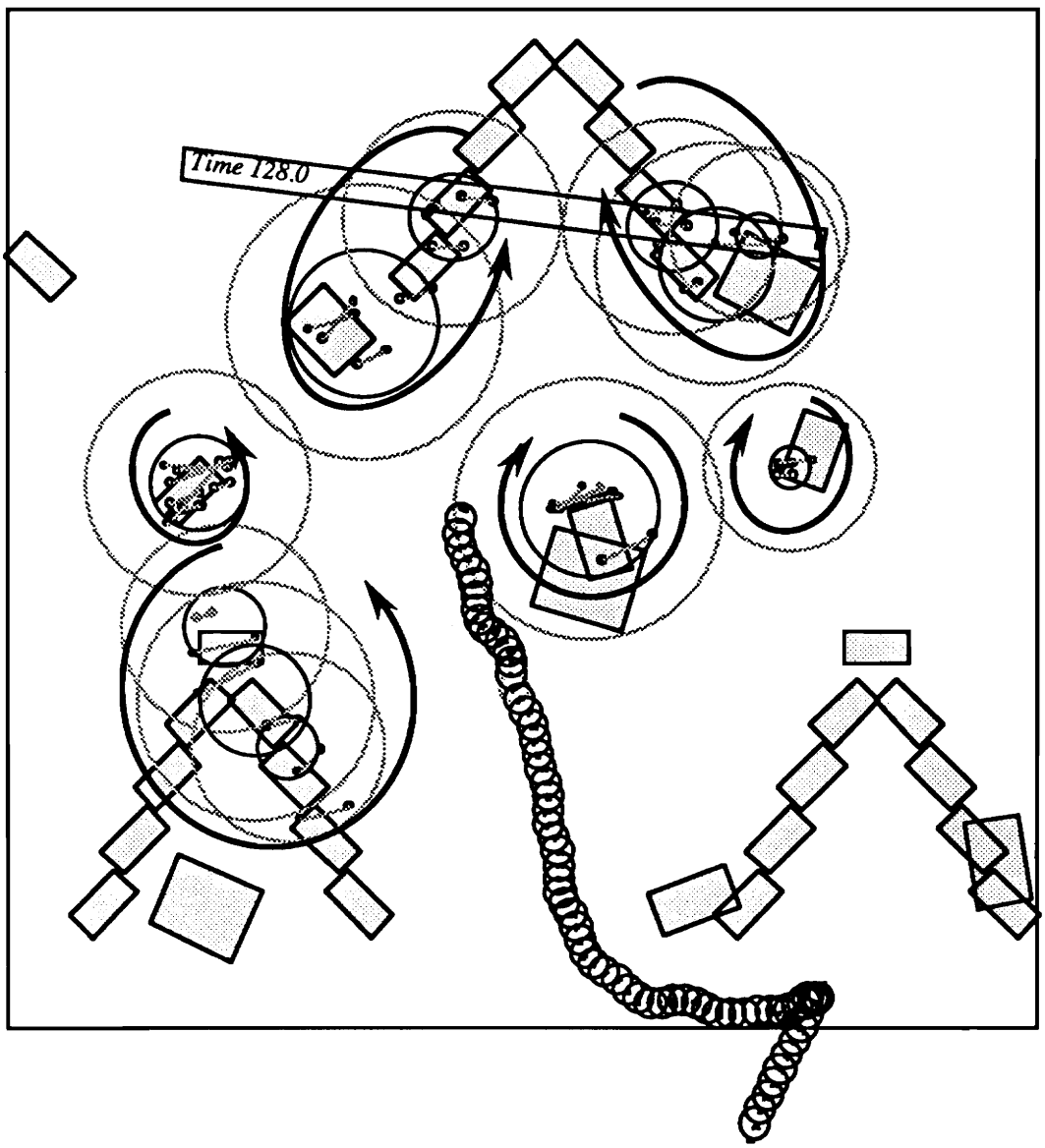
At time 57.0 LNav's activity results in the two boulder groups forming the trap to become sufficiently close to one another that they are combined (see Figure A.9). The combined grouping takes on a clockwise spin because the total area of the features to the right exceeds the total area of the features to the left (not accounting for overlay between members of a single boulder group).



Locating the trap

Figure A.9

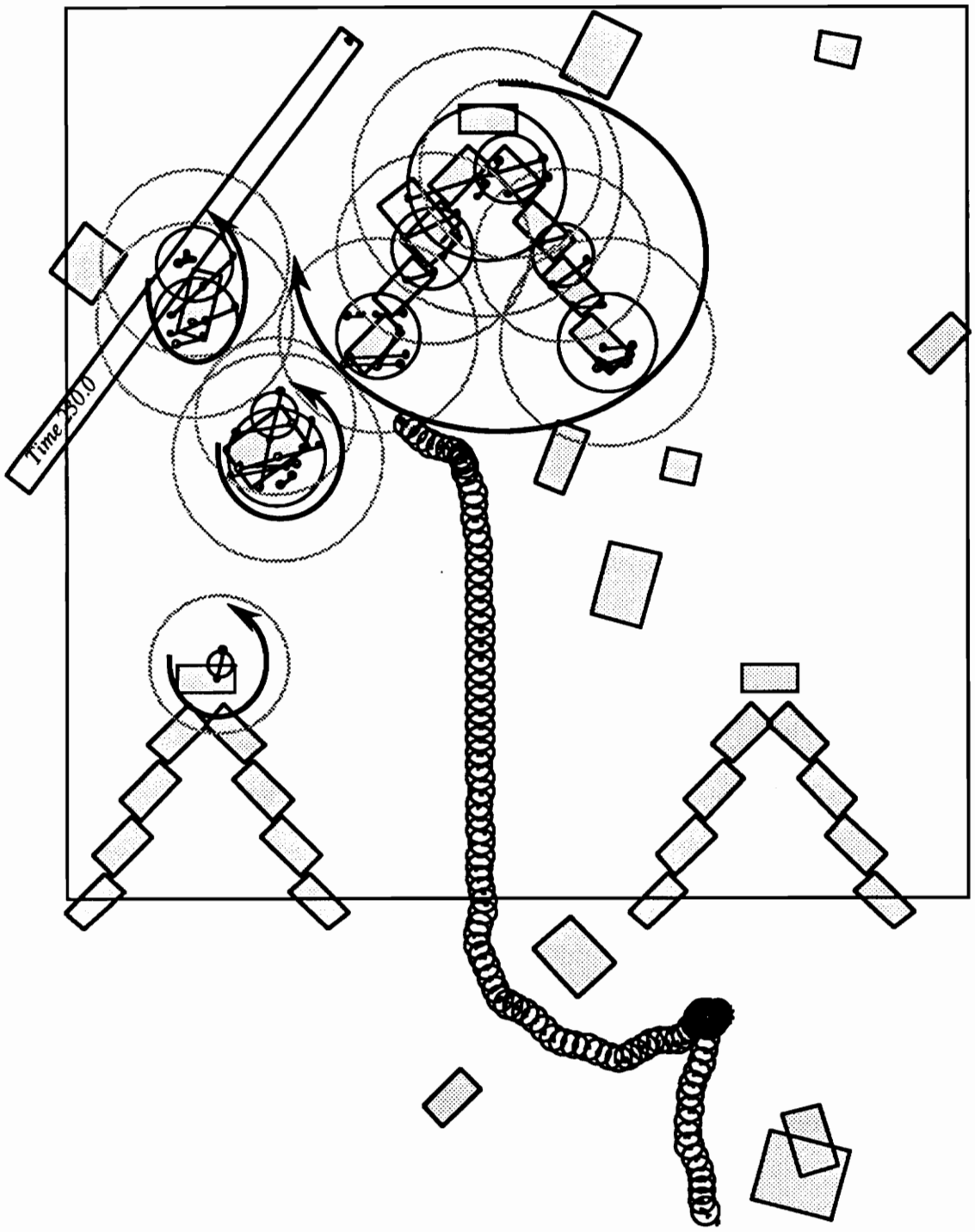
Having identified the first trap LNav is on the way again and is headed for another trap (see Figure A.10).



Heading into another trap

Figure A.10

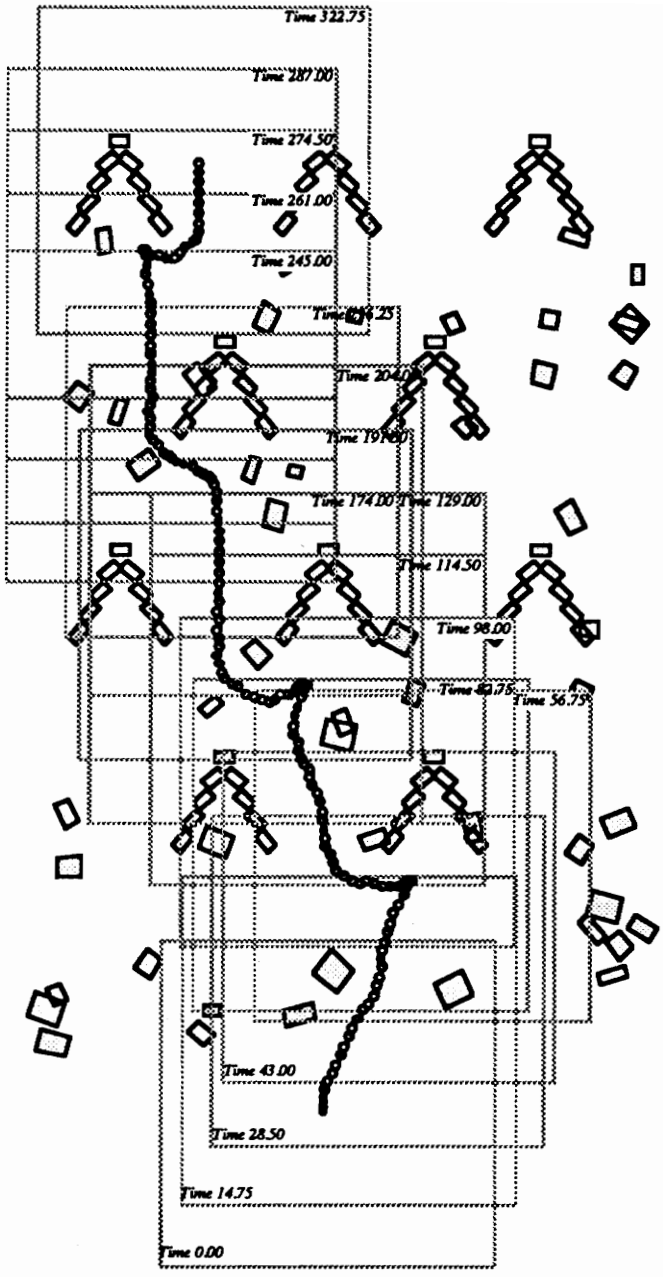
LNav continues to detect and avoid traps while at the same time making progress on the navigation task (see Figure A.11).



Detection and avoidance of a trap

Figure **A.11**

LNav is able to detect and avoid the final trap as well (see Figure A.12). A full trace of the simulation is shown in Figure A.12. The rectangles in the figure represent the positions that the terrain grid assumed during the robots' motion through the boulder field. The total "real-time" simulated is 330 seconds. Running on a Macintosh II in Lisp the simulator takes about 1200 seconds, about 200 of which is spent in garbage collection. These timings include the operation of sensor simulation which transforms the rectangular boulders into the unstructured representation made available to LNav.



Path taken through the boulder field.

Figure A.12

Vitae

Education:

1984: B.A. (math minor) Computer Science, Virginia Polytechnic Institute, Blacksburg, Virginia.

1987: M.S. Computer Science, Virginia Polytechnic Institute, Blacksburg, Virginia.

1990:Ph.D. Computer Science, Virginia Polytechnic Institute, Blacksburg, Virginia. Dissertation title: "Situationally Driven Local Navigation for Mobile Robots"

Professional Experience:

June 88 - Present: Member of technical staff, Robotic Intelligence Group, Jet Propulsion Laboratory, Pasadena, California. Primary responsibility: The design, development, and testing of navigation algorithms for control of mobile planetary exploration robots.

January 85 - June 88: Graduate Teaching Assistant,Virginia Polytechnic Institute, Blacksburg, Virginia.

June 86 - December 87: Graduate Research Assistant, Virginia Polytechnic Institute, Blacksburg, Virginia.

June 87 - September 87: Summer Employee, Artificial Intelligence Group, Jet Propulsion Laboratory, Pasadena, California.

December 84 - October 85: Programmer, Agronomy Department,Virginia Polytechnic Institute, Blacksburg, Virginia.

September 84 - December 84: Microprocessor Consultant, Electrical Engineering Department, Virginia Polytechnic Institute, Blacksburg, Virginia.

June 82 - September 82: Customer Support Consultant, Planning Research Corporation, McLean, Virginia.

Publications:

Gat, E., Slack, M., G., Miller, D., P., Firby, R., J., Path Planning and Execution Monitoring for a Planetary Rover, to appear in the *Proceedings of the 1990 IEEE Conference on Robotics and Automation*, 1990.

Slack, M. G., A Space-Time Model for Using Environmental Constraints in a Mobile Robot System, in the *Proceedings of SPIE's 8th Conference on Sensor Fusion*, 1989.

Atkinson, D., J., Cameron, J., Cooper, B., Doshi, R., Gennery, D., Katzmann, S., Lam, R., Lawton, T., Lay, K., Miller, D., P., Mishkin, A., Ruoff, C., Slack, M., White, J., Wilcox, B., Semi-Autonomous Mars Rover Navigation, in the *Videotape Program of the International Joint Conference on Artificial Intelligence*, Detroit, Michigan, 1989.

Slack, M. G., Planning Paths Through a Spatial Hierarchy: Eliminating Stair-Stepping Effects, in the *Proceedings of SPIE's 7th Conference on Sensor Fusion*, Cambridge Massachusetts, pp. 350-357, 1988.

Miller, D.P., Slack, M., Efficient Navigation Through Dynamic Domains, in the *Proceedings of the Workshop on Spatial Reasoning and Multi-Sensor Fusion*, pp. 230-239, AAAI, St.Charles, Ill, October 1987.

Slack, M., Miller, D.P., Path Planning Through Time and Space in Dynamic Domains, in the *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, pp. 1067-1070, AAAI, IJCAI, Milano Italy, August 1987.

Slack, M., Miller, D.P., Route Planning in a Four Dimensional Environment, in the *Proceedings of the 1987 Workshop on Tele-Robotics*, pp 41-47, JPL, Pasadena, CA, January, 1987.

Technical Reports:

Slack, M., Miller, D.P., *Path Planning Through Time and Space in Dynamic Domains*, Masters Thesis, Virginia Tech Department of Computer Science Research Report #TR-87-5, 1987.

Workshops:

Temporal-Spatial Route Planning, at the Planning systems for Autonomous Mobile Robots, part of AAAI 1987, Seattle Washington, July 1987.

Space-Time Path Planning, at the First Annual Conference of the Artificial Intelligence Society of the Mid-Atlantic States (AISMAS), Blacksburg, Virginia, April, 1987.

Invited participant in the AAAI Spring Symposium, Stanford University, March 1989.

Professional Affiliations:

American Association for Artificial Intelligence

Artificial Intelligence Society of the Mid-Atlantic States

Association for Computing Machinery - SIGART

The Planetary Society

SPIE