CHAPTER IV

NAVIGATIONAL PATH PLANNING

Obtaining intelligent travel has long been a concern for AI and robotics researchers. Many different issues are involved in the production of such travel. These include spatial reasoning, heuristic search, motor control, representation of uncertainty and environmental sensing of various types, particularly vision.

This chapter is concerned primarily with path construction and navigation in a partially modeled environment. The Autonomous Robot Architecture (AuRA) incorporates a hierarchical planner consisting of a pilot, navigator, mission planner and motor schema manager (the execution arm of the pilot). This chapter addresses specifically the role and operation of the navigator and its associated world model representations upon which the navigator bases its decisions.

In the introduction, relevant work will be cited followed by a description of the UMASS environment. Section 2 will describe the representation used by the navigator and the cartographic software that builds this map. The operation of the navigator will be described in Section 3. The extension of the representation to include diverse terrain types will be related in Section 4. A summary and conclusions will complete this chapter.

§1. Introduction

Early in the days of artificial intelligence, Amarel showed that a good representation is essential for the solution of a problem [1]. AuRA's representation was chosen based on an analysis of the existing representational strategies used in this domain (Chap. 2). A hybrid vertex-graph free-space representation was chosen for global path planning.

The UMASS VISIONS [53,102,140] system encompasses a multi-level representation scheme (Fig. 11). Previously, VISIONS has not maintained representations specifically

addressing navigational path planning. While provision is made for 3D representations of objects and their 2D projections in the vertical plane, no express representation of horizontal projections to the ground plane has been available. AuRA extends and complements the VISIONS system by adding representation levels that specifically deal with the issues involved in navigational path planning.

Navigation path planners come in many forms, simple [130], hierarchical [63,66,90,100, 108] and distributed [51,120]. Unfortunately, in developing hierarchical planners, such as the one used here (Fig. 4), no clear guidelines demarcate what should be delegated to each of the different components (pilot, navigator, etc.). In many cases functionality in one component for one given system is significantly greater than in another (e.g. Nitao and Parodi's reflexive pilot [98,103] incorporates much of what would be considered a navigator in other systems). The navigator in AuRA serves a role analogous to the navigator in a road rally: to provide a piecewise linear path to the vehicle pilot (driver) for execution. Instructions might be: proceed 1.2 km on this road then turn right 90 degrees at the traffic light. The navigator operates from a relatively static map and is not concerned with unrepresented obstacles unless the pilot expressly requests an alternate route.

The pilot is considerably more short-sighted. It is concerned only with satisfying one subgoal from the navigator at a time (although future subgoals may affect its decisions). The pilot additionally accepts constraints from the navigator such as criteria for failure to attain a subgoal. If any of those criteria are met, the navigator is informed and navigational replanning initiated. Local alterations in the route can be made without reinvocation of the navigator as long as these alterations fall within acceptable limits. The pilot need not utilize the same representation the navigator does, as it assumes that the navigator has correctly produced a path that avoids any modeled obstacle. Consequently the pilot is concerned solely with avoiding unmodeled obstacles (subject to certain constraints). Other work [63,66,98] describes the use of similar hierarchical planning systems.

§1.1 *UMASS environment*

The two arenas in which the robot is operated include both indoor and outdoor environs. The first is within the confines of our building, the Lederle Graduate Research

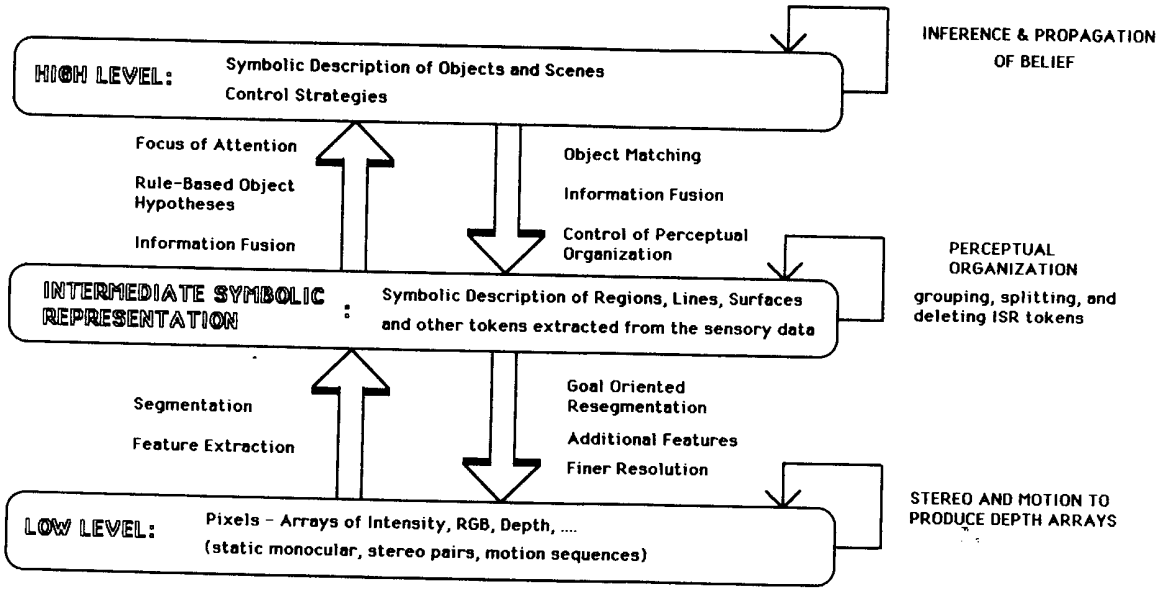


Figure 11: Representations and Processing in VISIONS

Representations ranging from schemas to low-level visual data are organized in a multi-level system. The reader is referred to [55] for additional information. (Reprinted from [55])

Center (GRC). The navigator assumes significant but incomplete *a priori* knowledge of the world. Blueprints for the building (Fig. 12) constitute the basis on which the initial indoor representation is built. A digitizer is used to enter the relevant map features.

The second arena is the grounds surrounding the GRC. This model is derived largely from a map made via aerial photography (Fig. 13). Figure 14 shows two photographs of the same area. Multiple terrain types are present including concrete sidewalks, grassy regions and a gravel path, all of which are available for navigation by the robot.

It should be noted that although the ground plane assumption is made (i.e. the free space is flat) as a simplification for these early phases of the research, there is nothing inherent in the representation that precludes the use of surface models (e.g. planar patches) to represent topographic features within the free space regions.

§2. Representation

To address the issues of path planning, a static representation and a dynamic representation have been developed (the reader is referred to [140] for visual interpretation representations). The static representation, or long-term memory (LTM), is where all *a priori* knowledge is embedded. It is static only in the sense that learning has not yet been incorporated into the system. Although a variety of sensor interpretation strategies also access data stored in LTM, the navigator is the prime consumer of this representation within the hierarchical planner.

The dynamic representation, short-term memory (STM), is a layered representation consisting of the robot's current perception of the world based upon a long-term memory context. Of the planner components, the pilot (and motor schema manager, which is the execution arm of the pilot) is the principal user of the data stored here. Portions of LTM are instantiated in STM based upon the robot's current position and the navigator's instructions. As the robot traverses this path, sensor data (visual and ultrasonic) are incorporated by the cartographer to build up a dynamic model of the perceived world. This is then used to direct the pilot to appropriate action when the path is blocked or a short-cut makes itself apparent. Additionally, vehicle localization (increasing positional certainty) can be guided by available landmarks found in these regions of visibility. A discussion of the details of short-term memory for navigation appears in Section 5.

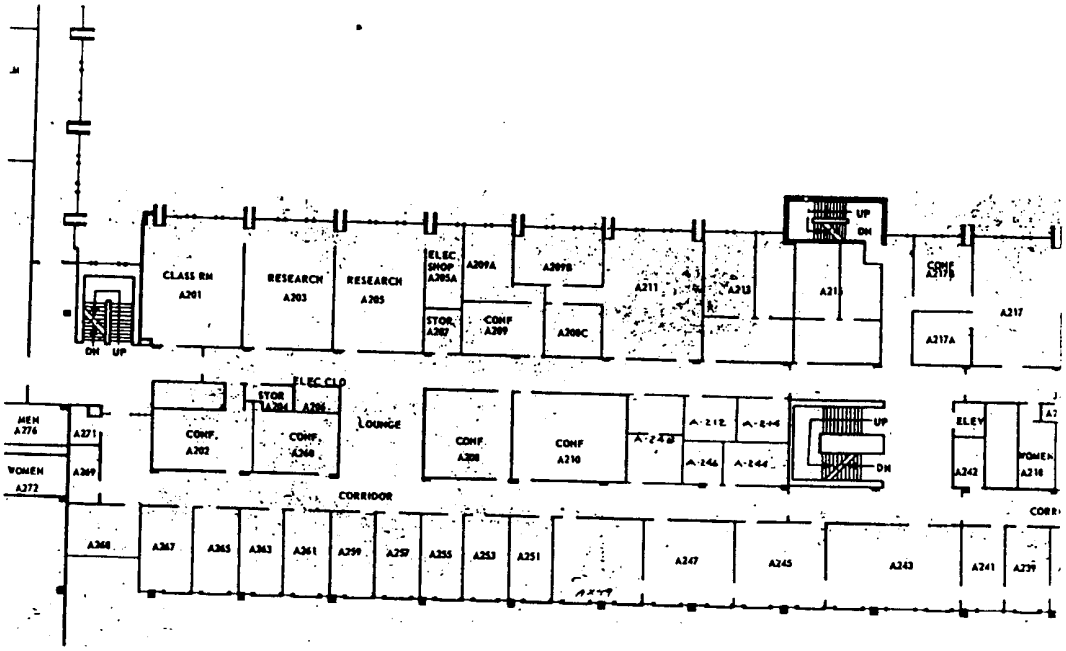


Figure 12: Indoor environmental model

This diagram, a partial blueprint for the GRC, serves as the basis for building the long-term memory model of the robot's world (indoor case).

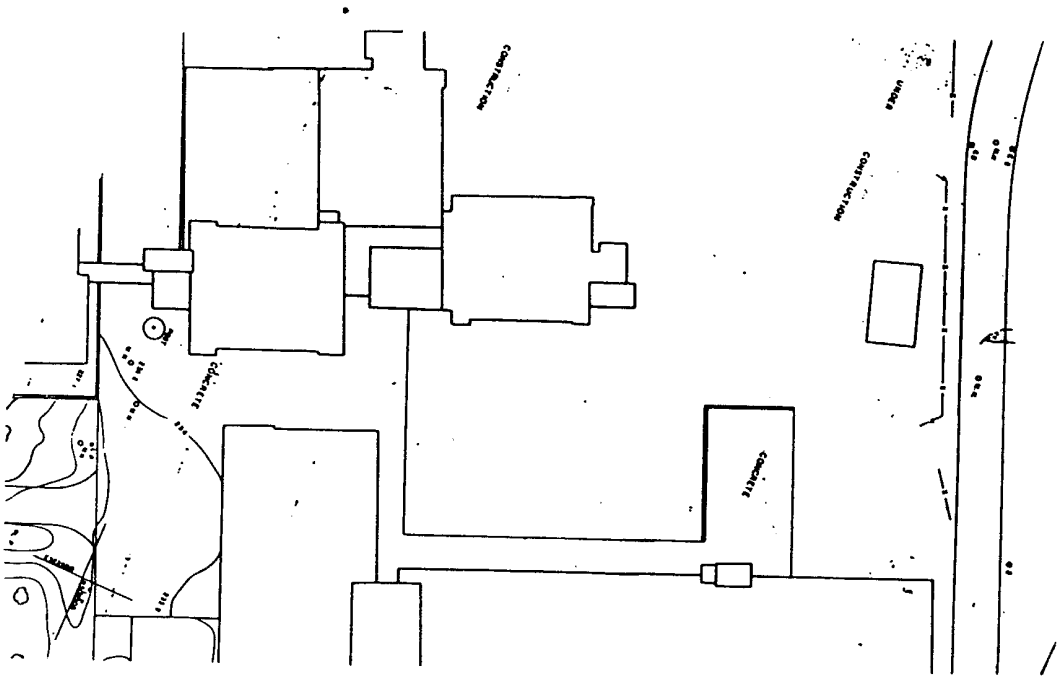


Figure 13: Outdoor environmental model

For the outdoor scenario, the long-term memory representation is built starting with an aerial map of the environment surrounding the GRC.

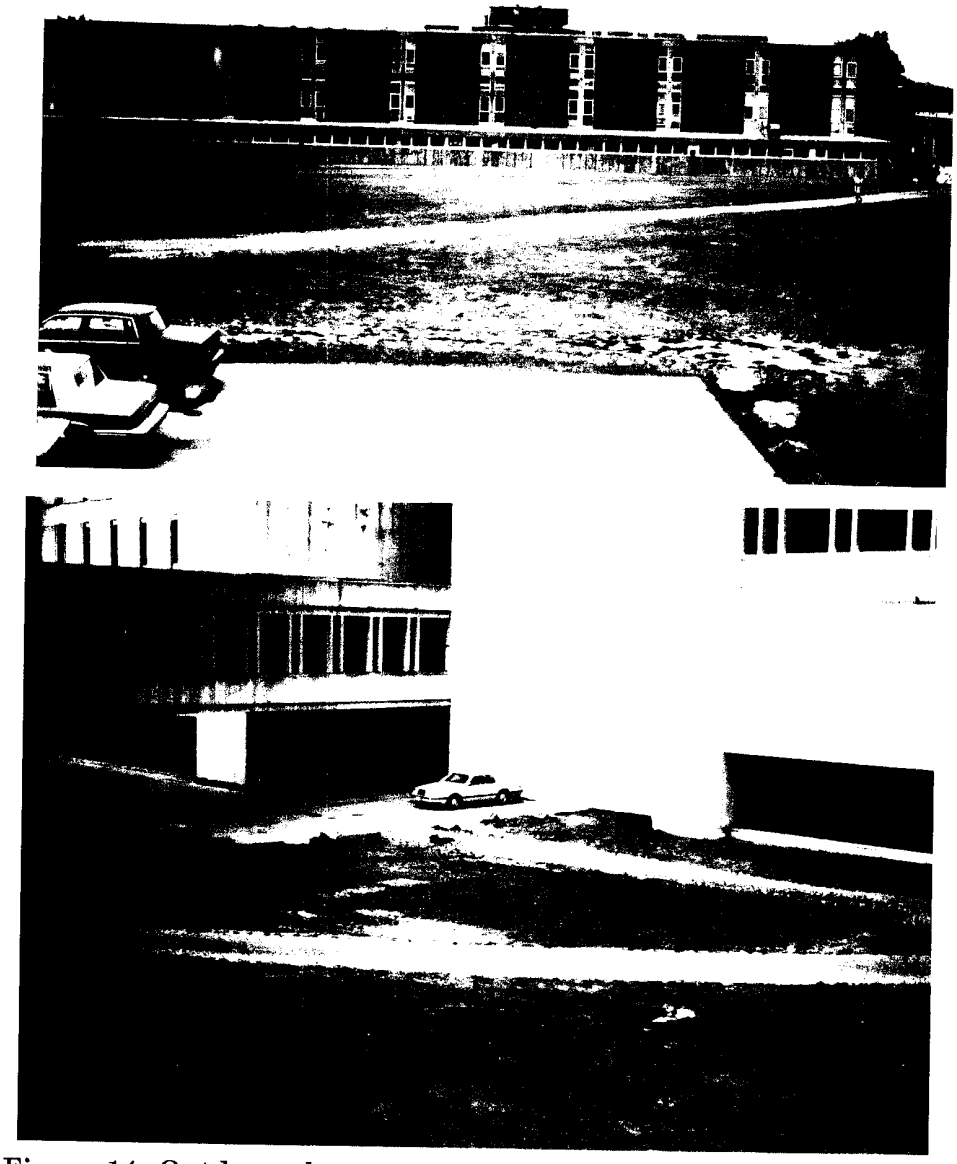


Figure 14: Outdoor photos

Two outdoor photos of the area depicted in Fig. 13.

§2.1 *Long Term Memory - Meadow Map*

The principal representation used by the navigator is a "meadow map" representation (a hybrid vertex-graph free-space model) based on previous work by Crowley [36] and Chatila and Laumond [33]. It models free space as a collection of convex polygons. Diagrams depicting an indoor scene and an outdoor scene appear in Figures 12 and 13 respectively. The rationale for using convex regions is that a line between any point within one convex region to any other point within that same region is guaranteed to be free of collisions with all known obstacles. Thus, the global path planning problem simplifies to finding an appropriate sequence of convex region traversals. (Actually finding a "good" path is more difficult - see the path improvement strategies described in Sections 3.2 and 4.2).

What distinguishes this representation from the efforts that preceded it is the ability to embed both terrain characteristics and data for establishing sensor expectations and its extension to allow navigation over diverse terrain types (see Section 4). Convex regions were chosen over a regular grid approach due to their ability to avoid digitization bias, a smaller search space, and a significant reduction in memory requirements. Voronoi diagrams (a set of polygonal regions, each representing an enclosed area in which all contained points are closer to one particular point in a given point set than to any other point in the set) were avoided due to their inability to relate landmark and terrain data readily and their perceived limitations on flexibility of path construction when compared to the strategy used in AuRA.

In this section, the map building algorithm used to construct the basic representation (not multi-terrain) is described. This is followed by a brief presentation on the role of the feature editor and its importance for guiding environmental sensing.

Meadow Map Construction

The algorithm for the construction of the LTM meadow map is described in Figure 15 and consists of the following phases: initialization, main map building and clean-up.

Initialization

In the initialization phase, a series of vertices in global coordinates describing the maximum reaches of robot navigation are accepted. In the case of the interior of a

FREE SPACE MAP BUILDING ALGORITHM

Initialization

Accept and shrink bounding region (*a lá*) configuration space)
Accept and grow modeled obstacles (configuration space)
Merge collided grown obstacles and border together
Attach obstacles to border (1 region results)

Main map building algorithm

IF region is convex (no concave angles present)
    done
ELSE
    Find (most,least,first) concave angle
    Connect it to (most opposite,leftmost,rightmost) clear vertex
    Apply main map-building algorithm recursively
        to the two resulting regions
ENDIF

Clean-up

Merge any regions together that will yield a convex region
Output list of connected convex regions

Figure 15: Free space map-building algorithm

building, this would be the bounding walls. In more open terrain, it might be boundaries of limiting paths or an imaginary polygon bounding the traversable region. There are no restrictions on the shape of the bounding region (and obstacles) other than that they be represented by a series of straight line segments. Curving surfaces must be converted to piecewise-linear segment approximations. The raw data is obtained from a map or blueprint of the region and the use of a bitpad digitizer.

After the actual coordinates of the bounding region are accepted, the region is shrunk by a distance equal to the radius of the robot plus a safety margin (Fig. 16) in the configuration space (C-space) manner as developed by Lozano-Perez [76]. This enables the robot to be treated as a point thereafter for path-planning purposes. Ties are maintained via pointers from the newly created C-space vertices to the original bounding vertex.

During the shrinking (and obstacle growing process described below) a deviation from standard C-space techniques was required in the case of concave vertices. Normally the circular robot would produce a curved C-space for a concave angle (Fig. 17a). Two alternatives are available to produce the required linear segments. First the resulting side line segments could be extended until they meet (Fig. 17b). This could result in significant and unnecessary loss of free space for very sharp angles, even resulting in the blockage of free space corridors (Fig. 17c). The second alternative is extending the line segments and then chopping the resulting C-space region, when a line normal to the bisector of the grown angle is intersected (at a distance from the vertex equal to the robot diameter) with the segments produced in the first method (Fig. 17c-d). Although this approach still wastes some free space, it is considerably better than the first case. The principal drawback is the formation of two grown vertices from the original ungrown one, which results ultimately in more regions being formed and thus more processing time during the later path planning phase. The decision at what degree of convexity to switch from straight extension to chopping mode is controlled by setting a program parameter. Highly cluttered areas would favor chopping for most of the concave angles to prevent passage occlusion, whereas relatively clear areas would prefer the straightforward extension method.

Known obstacles that are present in the environment (pillars, telephone poles, etc., – any static impediment to motion) are then added. These also are digitized in the manner above. These obstacles are then grown in the C-space style for the same reason that the

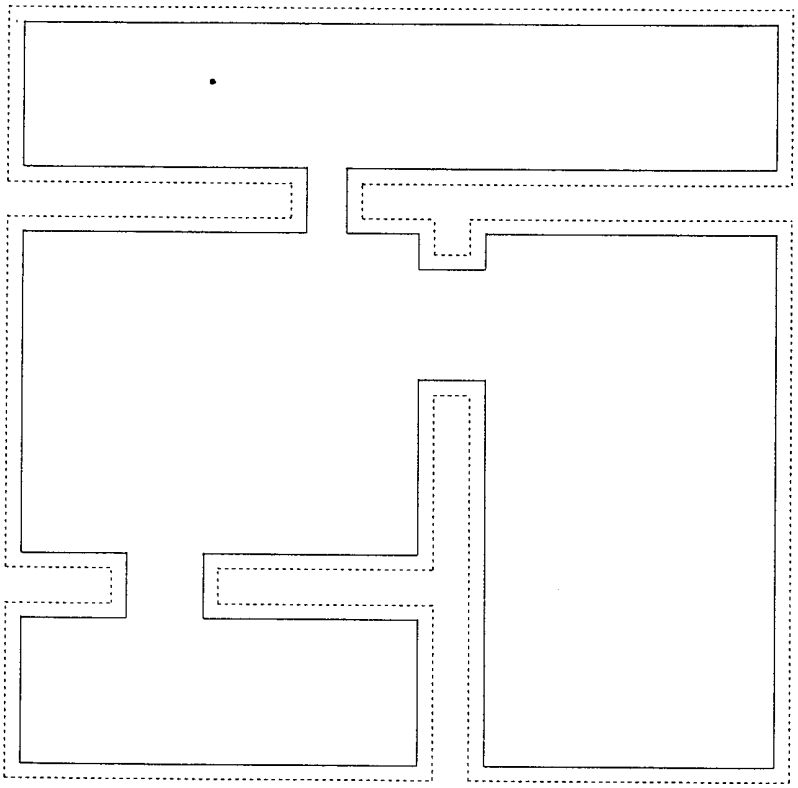


Figure 16: Grown border

A test indoor case, based on an example from [51], shows the original ungrown border, (dotted line), and the resulting shrunken region (solid line). (The shrinking is exaggerated in this figure for the sake of clarity)

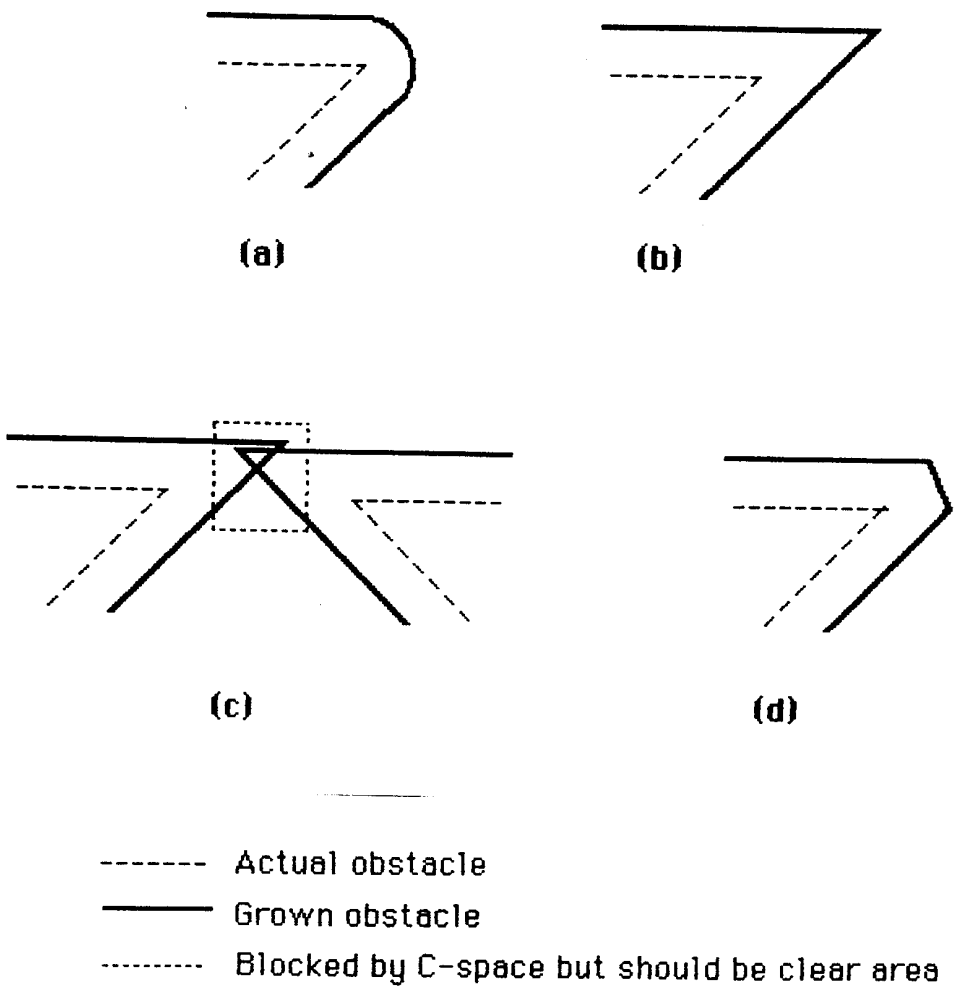


Figure 17: Chopping concave angles

(a) Traditional configuration space growing for a circular robot.

(b) Simple intersection method results in a single vertex.

(c) Simple method can occlude passages that should be clear.

(d) Chopping the angle reduces waste but results in 2 vertices.

bounding region was shrunk (Fig. 18). Any obstacles whose growth results in a collision with the bounding region, are merged into the border as they no longer can be completely circumnavigated.

Finally, the remaining obstacles are attached to the bounding region as follows. The obstacle vertex that is closest to a bounding obstacle is attached to the bounding region by two passable links; one going out to the obstacle and the other returning. This is repeated until all the obstacles are connected. In essence, a single region, consisting of the border and the perimeters of all the grown obstacles, is produced (Fig. 19). To the mathematical purist, this would not be a region as the two passable lines connecting the obstacle to the border are in identical positions. Nonetheless, this simplifies the recursive decomposition algorithm which appears in the step following.

Main Map Building Algorithm

This portion of the algorithm decomposes the region produced in the initialization phase by recursively splitting the area until all resulting regions are convex. Upon receipt of the initial region it is checked for convexity. If the region is convex, this portion of the procedure terminates. If it isn't, which usually is the case, a concave angle is selected from those available in the region. There is guaranteed to be at least one concave angle or the region would be convex. Three options were considered for selection of the vertex: the least concave, the most concave, or the first concave angle found can be chosen (Fig. 20). Intuition as well as empirical results indicate that choosing the most concave angle results in the fewest regions to be remerged during the cleanup phase below. Choosing the first concave angle would be more computationally efficient during this phase, but may require additional compute time in the clean-up phase, offsetting any gains here.

After an appropriate concave vertex is selected, the second (victim) vertex for splitting the region in two must be chosen. Again we have defined three choices (Fig. 21): the leftmost clear vertex, the rightmost clear vertex, or the most nearly opposite vertex (right of center). A connecting edge, labeled as passable, is completed between the concave angle and its victim and the initial region is split in two. The algorithm is then applied recursively to each of the resulting two newly formed regions. Pointers within the newly produced edges are maintained indicating the adjacent passable region (Fig. 21a). Thus a graph of convex regions and their traversability is produced, facilitating search during

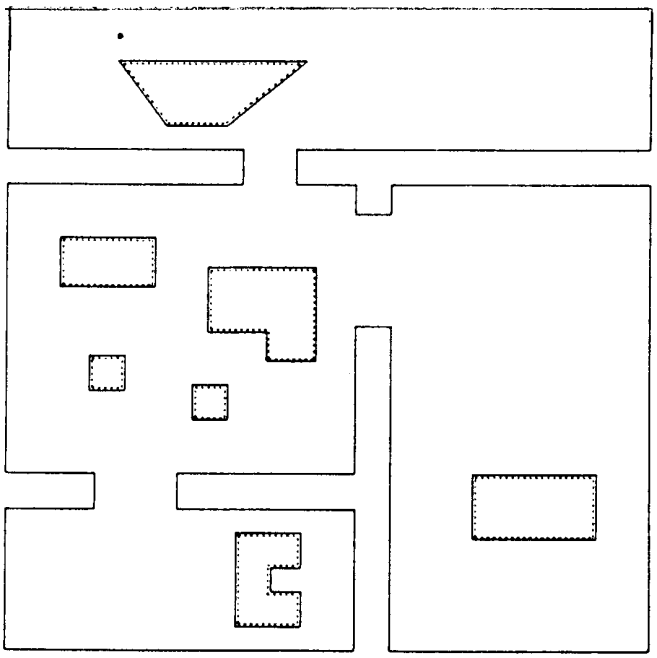


Figure 18: Obstacles

The obstacles have been added to the case shown in 16. The dotted line represents their original position and the solid line their grown area. Only the shrunken border region is shown (solid line), not the original border.

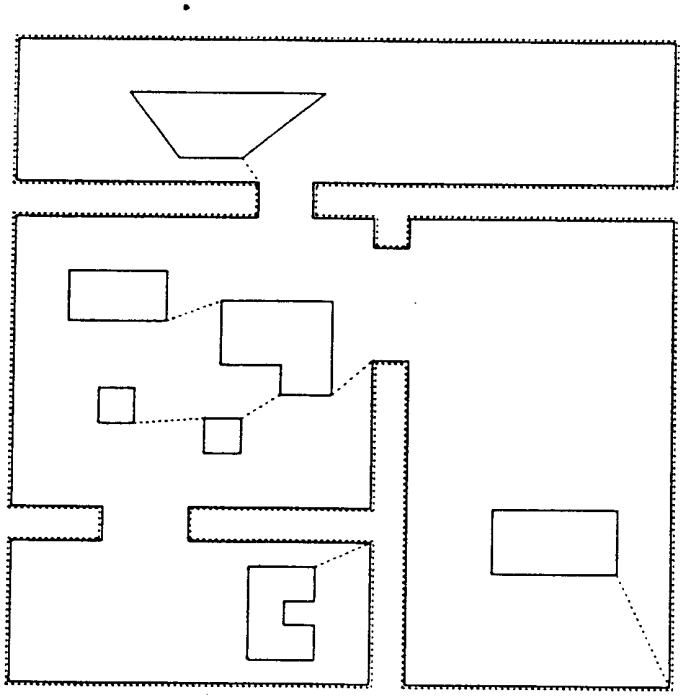


Figure 19: Attached obstacles

The grown obstacles have been attached to the surrounding shrunken border. The resulting single region is now ready to undergo convex decomposition.

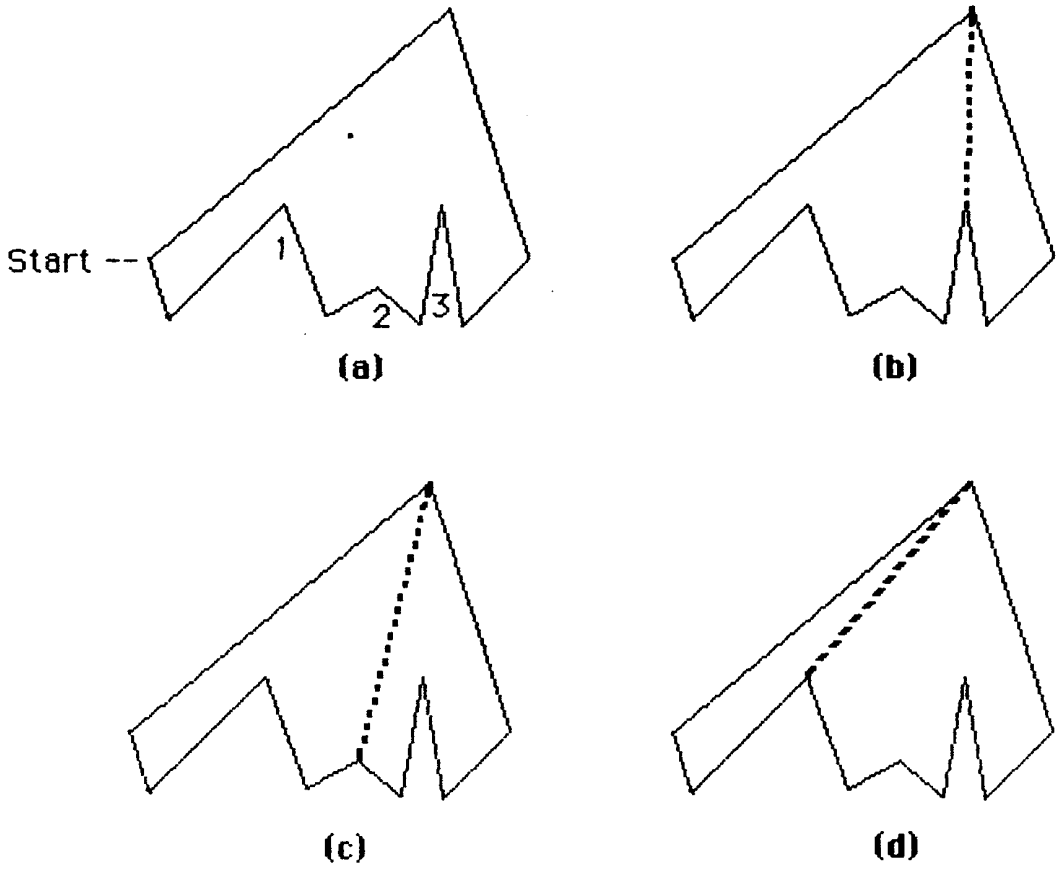


Figure 20: Effect of vertex selection on decomposition

a) A region to be decomposed that contains 3 concave vertices (numbered 1,2,3). The list representing the region begins at start and proceeds counter-clockwise. Figures b-d show how the vertex can affect the decomposition. For each case below, the most opposite vertex is selected as the victim.

b) Most concave vertex selected.

c) Least concave vertex selected.

d) First concave vertex selected.

the path finding process. This decomposition continues until all of the regions produced by this algorithm are convex.

The efficiency of each mode and their impact on the path planning computation, and data regarding map-building times appears in Appendix A. Considerable experimentation was carried out, trying to determine which of the concave selection modes and victim selection modes (of the 9 possibilities) yields the "best" results. Figure 22 shows 5 different decompositions on the same region. Just how to define what constitutes the best result is nebulous. Shortest Euclidean distance as a path length metric (which might appear to be the most obvious choice) may result in significant problems with the clipping of modeled obstacles during travel due to the inherent positional uncertainty found in the mobile robotics domain. Fewest overall legs in a particular path is another possible choice. In one instance [36] an algorithm producing the maximally large convex region is used. This might actually work against the path optimization strategies described below, although conceivably improving overall search time for the coarse "raw" path.

When the path search was restricted to the midpoints of the bounding regions, (A*-1, see Section 3.1), the experimentation indicated, even based on the shortest distance metric, that the results obtained were more strongly influenced by the shape of the initial bounding region and the choice of start and goal points of a particular path than by any predetermined choice of vertex selection modes for decomposition. That is not to say the choice of decomposition method did not produce significantly different paths in certain circumstances for the midpoint search; rather, information that is dependent on a particular initial region and the most likely paths to be taken within that region should appropriately influence the vertex selection process. An expanded search (A*-3) through 3 points on each passable meadow boundary (the midpoint and one point near each endpoint) largely decouples the dependency of the path cost on the decomposition method. Consequently, it becomes less significant which map-building strategy is chosen if this more costly search methodology is used.

When not guided by other factors, this author would choose the most concave angle and most nearly opposite angle as selection modes, as it generally results in the fewest merges in the clean-up phase while yielding a more aesthetically pleasing result (aesthetics are not forwarded as a metric however). It also is intuitively appealing as a "natural" decomposition strategy.

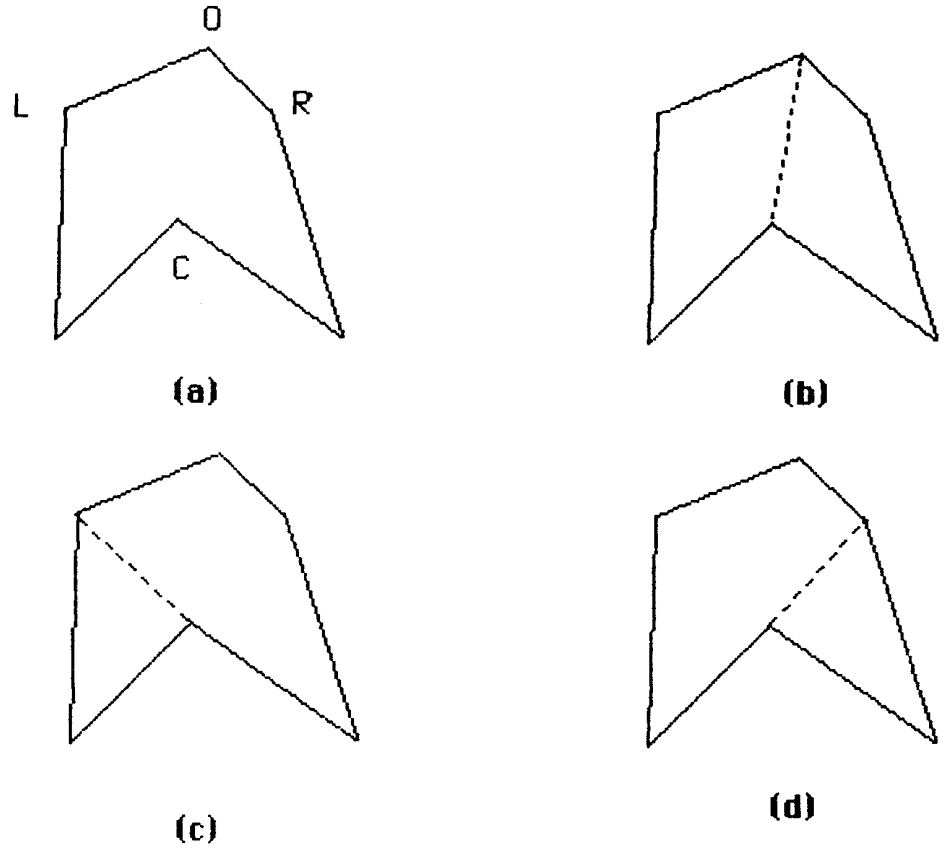


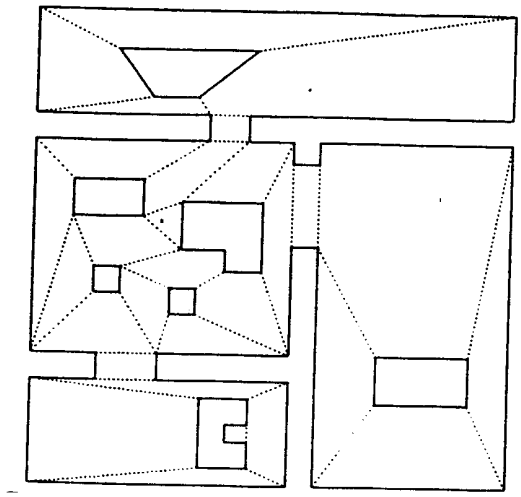
Figure 21: Effect of victim selection on decomposition

a) This figure contains one concave vertex (C) with three possible victims. O is the most opposite victim, L is the leftmost and R is the rightmost.

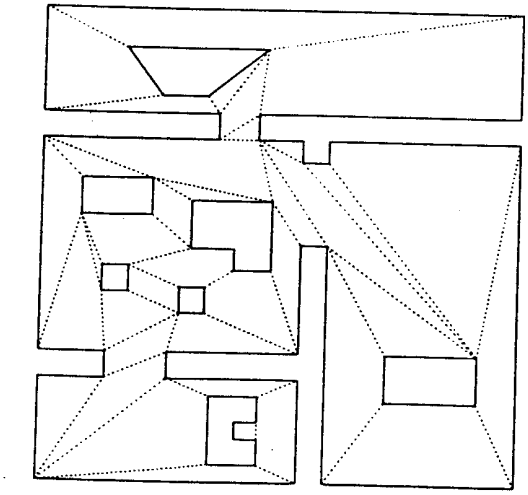
b) The decomposition resulting from the most opposite victim selection mode.

c) The decomposition resulting from the leftmost victim selection mode.

d) The decomposition resulting from the rightmost victim selection mode.



(a)



(b)

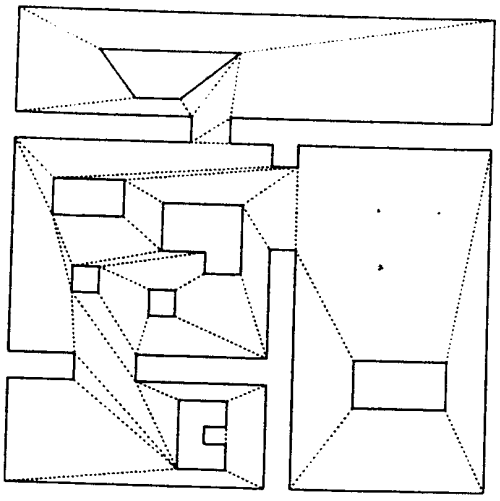
Figure 22: Different convex decompositions

Figure 22 shows 5 of the nine different decompositions available for the region shown in Fig. 19. Solid lines represent impassable obstacles and borders, dotted lines - passable boundaries between meadows.

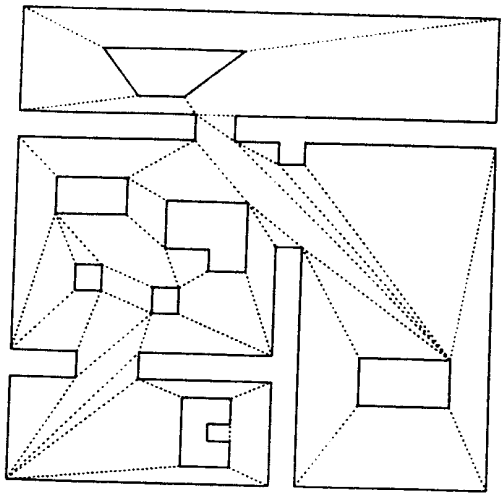
a) Selection modes: most concave angle, most opposite victim.

b) Selection modes: most concave angle, leftmost victim.

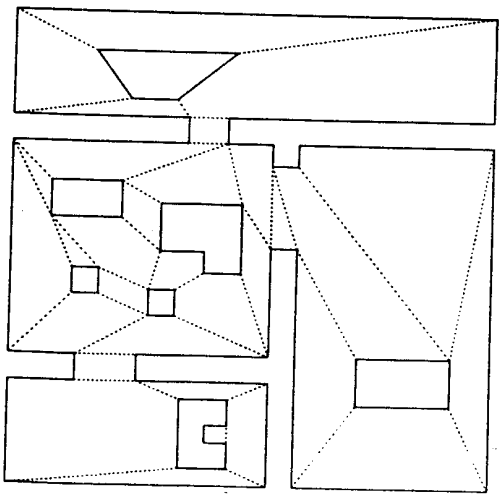
(Figure continued on following page).



(c)



(d)



(e)

Figure 22 continued.

c) Selection modes: most concave angle, rightmost victim.

d) Selection modes: least concave angle, most opposite victim.

e) Selection modes: first concave angle, most opposite victim.

One other note: although the algorithm is recursive, for efficiency in implementation, instead of using the system stack and system-provided activation frames found with standard recursive calls, a push down stack was managed by the map-builder process itself, avoiding the significant system overhead that would be required for the decomposition of large and complex free space areas. The net result in any case is a list of the resulting convex regions along with an embedded connectivity graph maintained by pointer links in the passable edges connecting a region to its adjacent passable regions.

Clean-up

The resulting number of convex regions produced by the main decomposition algorithm is not always minimal. In other words there may be some regions which can be merged together that still result in a convex region. This is a consequence of the local nature of the decomposition technique; no checks are run to determine the global consequences of a region splitting. Although this could be built into the algorithm, the backtracking that would be required is believed to be considerably more expensive than the simple merging step. In some instances (e.g. most concave - most opposite vertex selection modes) merging is relatively rare, while in others it is relatively common. During this phase, a pass is made on the convex region list that merges together any regions that would result in a single convex region. On merge completion, the map-builder process then invokes the feature editor.

It should be recalled that this is the algorithm for the simplest case, involving only one terrain type. Multiple terrain types require additional processing which will be described in Section 4.

§2.2 *Feature editor*

The feature editor was created to allow data pertinent to sensor guided navigation and uncertainty management to be added to the meadow map. It serves as an interface to assist in the knowledge acquisition and representation processes.

A major advantage of the meadow map representation is the ease with which representations of objects, landmarks, terrain features, etc., can be expanded. In an experimental system this is very important. The level of granularity at the point of attachment can vary. Obstacles or walls can be represented with full 3D models of the entire object;

2D planar representations of surfaces (pop-up views) can be associated with meadow sides (edges); or simplistic line models for individual corners, projected up into the image plane, can be attached to meadow vertices. For terrain, entire region characteristics (traversability data, statistical error data, data for guiding visual region segmentation algorithms, etc.) can be tied to the free space regions. Individual meadows can have topographical models (for non-planar surfaces) which represent contours in any way the designer of such a representation chooses. This flexibility for adding and modifying world representations is one of the prime factors in the choice of the meadow map scheme over other alternatives such as the regular grid or Voronoi diagram.

The mechanism for adding these representations is through the use of the feature editor. The concept is simple: a particular free space region, obstacle, obstacle edge, or vertex is chosen through the editor; data for the new representation is accepted by the editor; storage is allocated for it and a link is made between the new representation and the old. This is repeated until no more data is to be added. The data within the allocated representations can also be modified interactively if required.

The data stored in the representations can be acquired through sensors as well. For example in the case of visual data for region segmentation, by pointing the robot camera in the direction of a known region type (e.g. grass), and then acquiring the appropriate statistics through interactive use of the video digitizer and a histogram process, the robot can store the statistical features for a particular terrain type on a per run basis. This avoids the inflexibility that would be present if the statistics had to computed once for all weather and seasonal conditions. The result is more robust visual segmentation. In essence the robot can be trained quickly and efficiently to recognize certain terrain features. It would not be difficult to extend this to include data for landmark recognition and other necessities once appropriate representational strategies were chosen. Static representations can be input from data files, reducing the map-building time. It should be recognized, however, that changing lighting conditions and other environmental factors can render these statistics obsolete. Consequently, adaptive (feedforward) mechanisms are required by the perceptual processes to make the system more reliable (Chap. 6).

Some of the initial features included in the system are:

- Terrain data
  - Traversability factor (ease of passage)
  - Terrain-specific translational and rotational error data - (to guide the uncertainty map manager in handling positional uncertainty)
  - Data to guide terrain region labeling using visual region segmentation
  - Unmodeled obstacle density
- Obstacle data
  - 3D models of landmarks
  - Partial wire frame models of buildings
  - Vertical edge data for particular vertices (sides of buildings, doorways, etc.)

Currently this data is user-supplied. In the last stage of the map-builder, after the user exits the feature editor, the pointers for long-term memory are installed making LTM available to other processes. The map-builder process then terminates.

§3. Navigation (Global Path Planning)

After the map-builder process terminates, the planner process is initiated. The planner is hierarchical in design; consisting of a mission planner, navigator and pilot. The mission planner is delegated the responsibility for interpreting high level commands, determining the nature of the mission, setting criteria for mission, navigator and pilot failure, and setting appropriate navigator and pilot parameters. For example, if the mission is reconnaissance oriented, (e.g. searching for lost keys), the pilot mode of operation would be set to path seeking. On the other hand if it was target oriented, (e.g. delivering a pizza), the mode would be set to goal seeking. The mission planner, although part of the overall design, is not yet fully implemented, and has a relatively low priority. Section 6.1 describes the rudimentary mission planner used in the first-pass implementation of AuRA.

The navigator accepts a start and a goal point from the mission planner and, using the global map built by the map-builder, determines the "best" path to attain that goal. The

definition of optimality is determined by the mission planner. It might be the shortest, or the safest, or the fastest, or the least energy consuming path. In essence, the mission planner determines which cost functions and heuristics that the navigator will use in carrying out its role.

The remainder of this section deals with the search strategies used by the navigator, the path improvement strategies that convert a coarse, raw path into a refined one, and a presentation of results. The modifications necessary for multi-terrain navigation are presented in the section following.

§3.1 *Search*

The navigator's task is to search through the meadow map produced by the map-builder and derive a good path available for a specified start and goal. As stated earlier, "best" is difficult to define. Many different criteria can be used to affect the quality of a path. Parodi [104] used a weighted cost function and dynamic programming techniques to search through the solution space of a regular grid and arrive at the best path.

The A* search algorithm [56,96] is used in AuRA with heuristics that guarantee optimality. The A* cost function is defined as:

$$f = g + \hat{h}$$

where:

g = the measured cost of the path up the current point

$\hat{h}$ = the heuristic cost from the current point to the goal

(to be admissible, $\hat{h}$ must not overestimate the actual cost to the goal).

Two different search spaces are available for the search algorithm. The simplest and most efficient, A*-1, is built from the midpoints of the bordering passable regions (a concept derived from Crowley's adits [36]). The larger space, A*-3, is derived from a triad of points on the bordering regions; the midpoint and two points near each end of the passable edge (separated from the end by a specified safety margin). Although computationally more expensive (the space is larger), the advantage of A*-3 over A*-1 lies in a significant decoupling of the path planning from the map-building free space decomposition method (due to the expanded search space). The A*-3 method explores more alternatives, possibly resulting in a lower cost path than would be available with

the A*-1 search. Additionally, since it tests points in close proximity to obstacle vertices, the location of the boundaries of the adjacent meadows themselves become less important (especially for short paths). In either case (A*-1 or A*-3), the search space is smaller (and consequently faster) than that of a regular grid or pure vertex graph representation. Finding the initial coarse path is a fairly rapid operation (see Appendix A). It is guaranteed to be the best path available (subject to the cost function chosen) within the specified search space. This space, however, is not strictly analogous to the physical world.

The choice of A*-1 or A*-3 is made by the mission planner, differentiating between the two on the basis of whether it is more important to compute a path rapidly (A*-1: faster) or more important to traverse the path rapidly (A*-3: can yield a lower cost path). In many cases the paths resulting from both A*-1 and A*-3 are identical.

In order to ensure admissibility, the heuristic function $\hat{h}$ of A* must never overestimate the cost remaining to the goal. The easiest $\hat{h}$ heuristic function to guarantee an underestimate (or the exact cost) is the straight-line Euclidean distance on the plane from the current position to the goal assuming the best terrain. Since the search space is relatively small, no effort has gone into finding better heuristics. The computation time required to produce the path (in A*-1) is somewhat dwarfed by the time required to convert this initial raw path into a refined and reasonable path (see Appendix A).

The cost function used takes into account the traversability factor of a given terrain type, the actual distance traversed, and can readily incorporate other factors such as threat measurement, topographical grades, etc. This cost function is used in the g component of the A* algorithm. Other factors might include unmodeled obstacle density (perhaps a function of time of day - e.g. high obstacle density between classes on a sidewalk, low otherwise), and ease of localization (based on numbers of readily discernible landmarks within a given region).

At this point in our research, it is impossible to say just what constitutes a "best" path. If a path is very short but the robot gets lost due to inadequate landmarks for localization, or it gets mired in poor terrain and its dead reckoning sensors become grossly misleading, little has been achieved. The only effective metric is the robot's ability, under the pilot's control, to successfully complete the path. If it can, only then can we take into account additional yardsticks such as time, distance, etc.

Consequently, the ultimate goal of the navigator is to arrive at a "reasonable" path rather than a claimed "best" path. By reasonable we mean a path that appears plausible from a human's perspective - i.e. it is conceivable that a person would take a similar path. In any case even if an optimal path (by whatever definition) was attainable by the navigator it could only be based on partial information (i.e. the modeled world). Since the robot's environment is subject to unmodeled and even moving obstacles, there is no *a priori* guarantee that any path produced by any navigator is the best path, given only incomplete world knowledge (although the path is optimized relative to the current world model). Reasonableness seems an acceptable criteria.

In a dynamically changing world, which can quickly invalidate preformulated plans, how do we gauge "reasonableness"? Only successful completion of the robot's experiments can be the judge. This is especially the case for multi-terrain navigation. When would you, as a human, take a short-cut over the grass in lieu of the sidewalk? Robot's have different locomotion systems so this example is not fully extendible, but the navigator can still be judged in this light. One more point in defense of the premise of reasonableness: do people really choose an "optimal path" when traveling from one point to another? I think not, except under rare circumstances. The time required to compute the path, (referring to a map etc.), might take longer than the completion time of the path itself. Therefore, no claims are made for the optimality of the paths produced by the navigator, only that the resultant paths are reasonable under all observed conditions.

In summary, the navigator algorithm (shown in Fig. 23) accepts two points from the mission planner. It then searches, using the A* algorithm with a cost function based on terrain factors and traversability, the space of midpoints (A*-1) or triads (A*-3) of connecting adjacent passable meadows, outputting a coarse path consisting of a series of piecewise linear segments connecting the start, the edges of bordering meadows and the goal. This approach generally expands fewer nodes than would a comparable pure vertex graph of the obstacle edges (and obviously much fewer than a regular grid) as the number of passable meadow boundaries is less than the number of vertices. The pure vertex graph (although guaranteed to produce the shortest path) also suffers from an inability to readily produce safe paths (paths that minimize the danger of the robot clipping an obstacle), since the free space is not directly represented. A Voronoi diagram can readily produce safe paths, but also lacks the flexibility afforded by this representation

to change its strategies (safe to short to fast) when deemed appropriate by the mission planner. The Voronoi diagram discards information regarding the obstacles themselves too soon, making it necessary to reconstruct that data when needed for alternate path finding strategies.

The key, however, lies in the path improvement techniques described below. Without these techniques the raw path produced in many cases would appear to be haphazard and unreasonable even to the casual observer (especially for A*-1).

§3.2 *Path Improvement Strategy*

Path improvement techniques are relatively common in the use of regular grids. Although the representation used for AuRA's long-term memory is a meadow map and not a regular grid, the precedent of refining a coarse path into a better one exists. Thorpe uses a relaxation based approach on a coarse grid [126] while Mitchell and Keirsey use a compensation technique [88] to minimize inefficiency due to digitization bias.

The algorithm for path improvement for the simple single terrain case is presented in Figure 24. A graphic illustration of the process appears in Fig. 25. The "raw" path is first received from the navigator. Beginning at the start path node and proceeding to the end node, each node on a passable meadow border is tested at three locations; slid all the way to the left (leaving a specified safety margin clearance), slid all the way to the right (minus safety margin), and unchanged at the middle. The lowest cost solution is chosen and the path modified accordingly. This can be visualized as pulling on the ends of the path thus tightening the path around the obstacles and walls. This is considerably less costly than a relaxation algorithm requiring multiple iterations over the entire path. (A limited relaxation algorithm involving only the transition zones is required for the multiple terrain case - see Section 4). The A*-3 search method can bypass this initial tautness processing as its search strategy has already effectively accomplished it.

For eliminating unnecessary turns, if deemed appropriate by the mission planner, a straightening algorithm is utilized. Any unnecessary turns in the path are removed. Beginning with the start path node, all further path nodes are checked against the current path node to see if a path exists that does not intersect with any of the known environmental obstacles. If such a path exists, all intervening path nodes between the two connectable nodes are deleted from the path. This process is repeated for all nodes in

PATH FINDING ALGORITHM

Accept start and goal from mission planner.

Check for validity (located in free space).

Search

Apply A* search algorithm through convex region connectors.

(A* - 1: midpoint only)

(A* - 3: midpoint + two points near endpoints of connectors)

Output Raw Path.

Path Improvement Techniques

If specified

A. Tighten path by sliding towards side vertex
by some given amount.

B. Straighten path by removing any turns that are not essential
for a clear traversal.

(details for both parts A and B appear in Fig. 24)

Return "reasonable" piecewise linear path through world model.

Figure 23: Path finding algorithm

PATH IMPROVEMENT ALGORITHM

(single terrain type)

Accept the coarse path from the search component of the navigator

Tautness part

Accept safety margin (clearance from side)
Get first border midpoint of coarse path
DO WHILE Not at end of path
    compute length of path for three cases
        a. Midpoint unchanged
        b. Midpoint slid to right (maintaining safety margin)
        c. Midpoint slid to left (maintaining safety margin)
    choose lowest cost path from a, b or c.
    modify path if necessary and mark path node as moved
    Get next path node
ENDDO

Straightness Part

Get start of path
DO WHILE not at end
    IF clear path is available to any path node ahead of current node
        delete all intervening nodes
    ENDIF
    Get next path node
ENDDO

Clean up

Slide towards edges again as in tightening part above if path was
  straightened (only for unmoved path nodes still at midpoint)
Output refined path

Figure 24: Path Improvement Algorithm

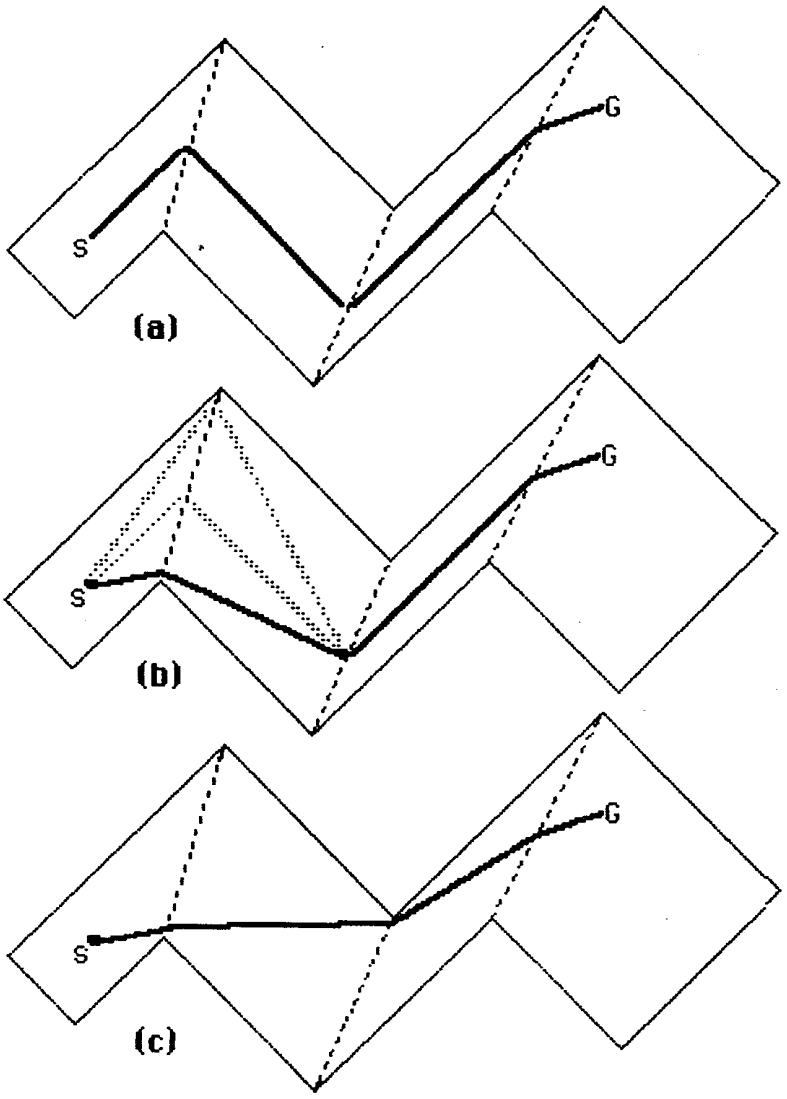


Figure 25: Path improvement - tautness component

a) The initial raw path passed from the A* search strategy to the tautness component. S denotes the start and G the goal.

b) The first passable boundary is tested at three locations, the midpoint and the two endpoints (minus a safety margin). The lower cost result is shown as the dark line.

c) The process is repeated at the other two passable boundaries. This results in a lower cost but as yet unstraightened path. (The kink from the final boundary to the goal will be removed by the straightening component).

the path.

If the path is straightened, a better path may now be obtained by sliding some of the previously unmoved path nodes. Before exiting, the algorithm checks all these unmoved nodes, if there are any, to see if a lower cost path can be obtained by sliding them along their meadow boundaries (basically the same procedure as in the tautness part above but checking only a subset of the remaining path nodes). The resulting refined path is output from the navigator and stored in short-term memory for use by the pilot.

Reasonable paths have been observed in extensive testing of all cases presented. Unnecessary detours around obstacles are removed by the straightening component of the path improvement strategies, while overall cost minimization is ensured by the tightening approach.

§3.3 *Results*

The results are presented in Figures 26-32. The straightening component of the algorithm can be observed to remove unnecessary detours around obstacles, while the tightening component reduces the overall path cost. The cost function used is the same cost function used in the search algorithm (for these figures, Euclidean distance is the cost).

A significant advantage in deferring the path improvement strategy until after the search (rather than being an integral portion of the search algorithm) lies in the ability to alter the path, if necessary, without re-searching. Additionally, embedding the straightening portion would be awkward at best within the search algorithm.

For A*-1, (Figures 26-29), the actual paths produced from the navigator are a function not only of the start-end points and improvement strategies used, but are also dependent on the modes used during the map-building. Considerable experimentation was conducted trying to determine which if any of the nine modes available to the map-builder resulted in consistently better paths. No clear connection could be made between the cost of the path, the start and end points of the path, and the nature of the convex region decomposition. In some decompositions, for a given start and end point a better path (A*-1) could be obtained using one decomposition approach over another (Fig. 28 and 29). For another set of start-goal points, however, the same approach that performed poorly in the first case did better than the one that previously performed well. For all

Figure 26: Single terrain path planning example - (A^*-1)

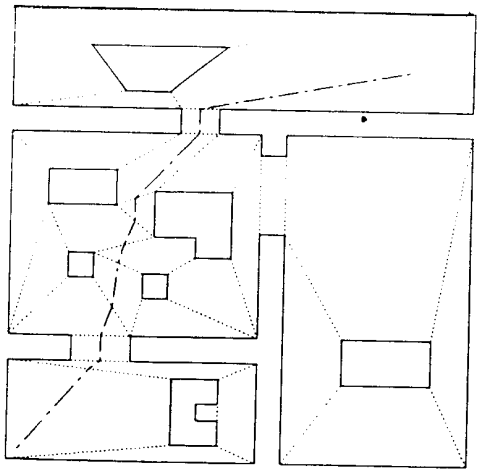
This sequence illustrates the path finding process of the navigator for a single terrain type. The convex decomposition of Fig. 22a is used for Figs. 26-28. The initial start in this case is in the lower left corner, while the goal is in the upper right. Solid lines represent grown obstacles and borders, dotted lines represent passable meadow boundaries and the dot-dash line is the path. The safety margin was specified as 1 foot (on an overall scale of approximately 400' by 400').

a) The initial path produced by the A^*-1 search algorithm through the midpoints of the passable bordering meadows.

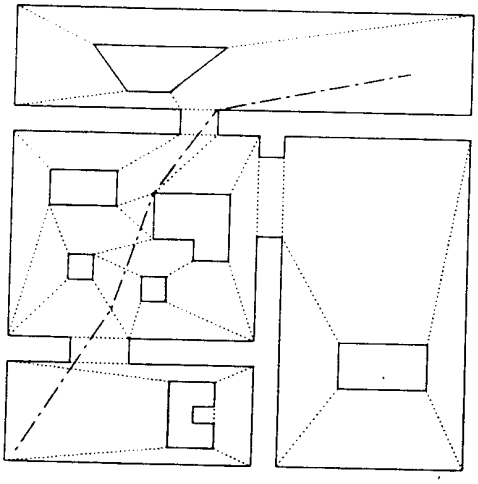
b) The path after undergoing path improvement strategies.

c) The same path as in (b), shown without passable boundaries for clarity.

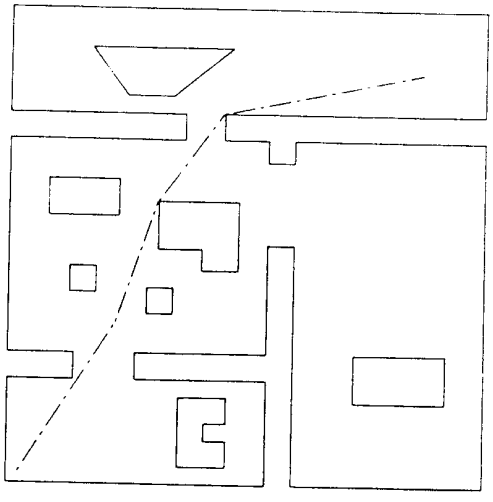
d) A safer path (safety margin 10 ft). Note that a safety margin of 10 feet does not guarantee 10 foot clearance of all obstacle vertices. It serves only to limit the tightening (sliding) along the passable border to within 10 feet of the vertex. It is NOT a measure of path distance from the vertex as one should note in the last corner in the upper right portion of the path.



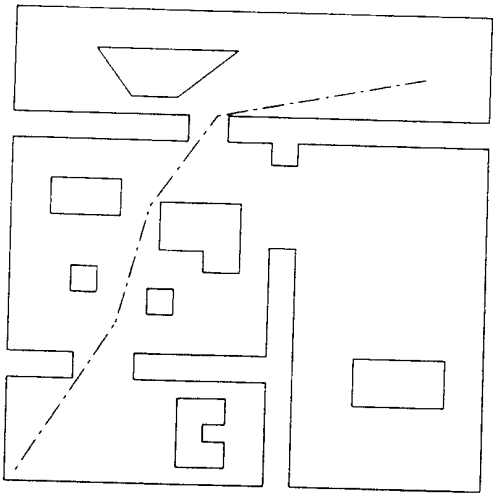
(a)



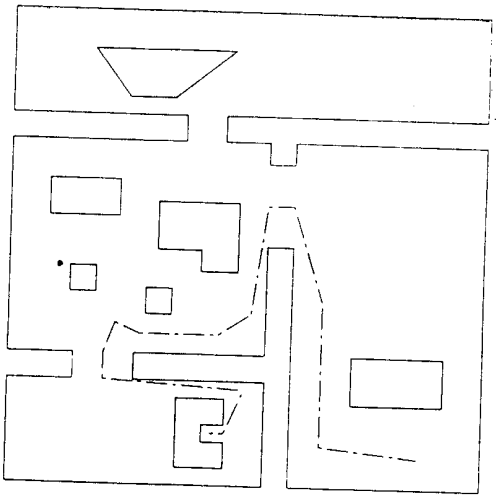
(b)



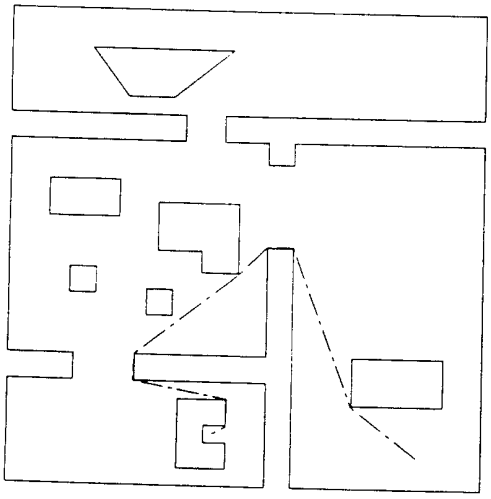
(c)



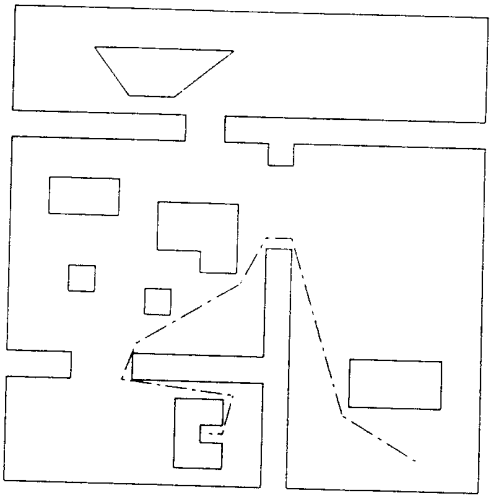
(d)



(a)



(b)



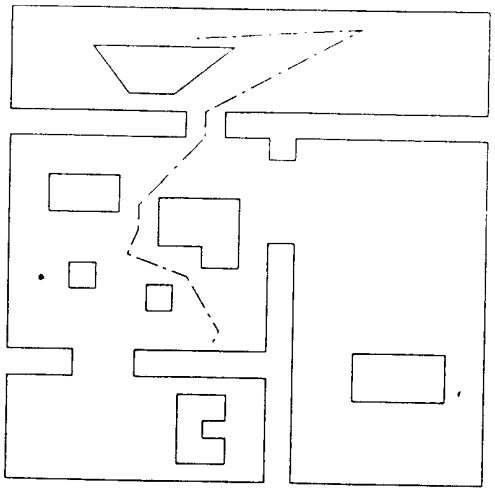
(c)

Figure 27: Another single terrain planning example (A*-1)

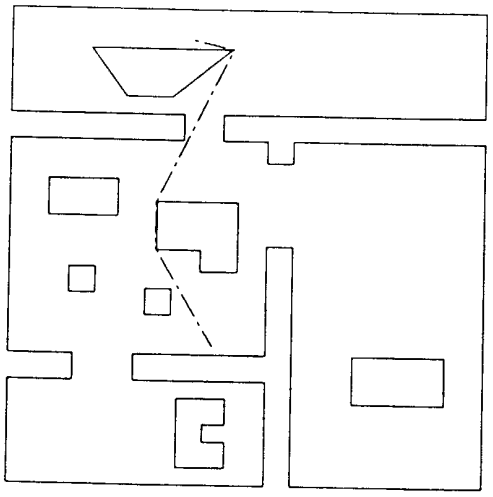
a) Initial coarse path.

b) Improved path (safety margin 1 foot).

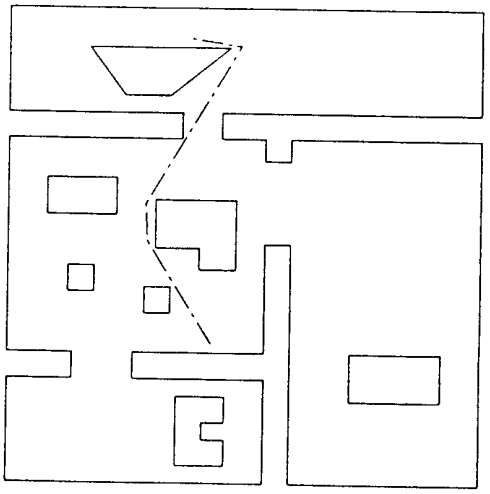
c) Improved path (safety margin 10 feet - see note for Fig. 26d.)



(a)



(b)



(c)

Figure 28: Yet another single terrain planning example (A*-1)

a) Initial coarse path.

b) Improved path (safety margin 1 foot).

c) Improved path (safety margin 10 feet - see note for Fig. 26d.)

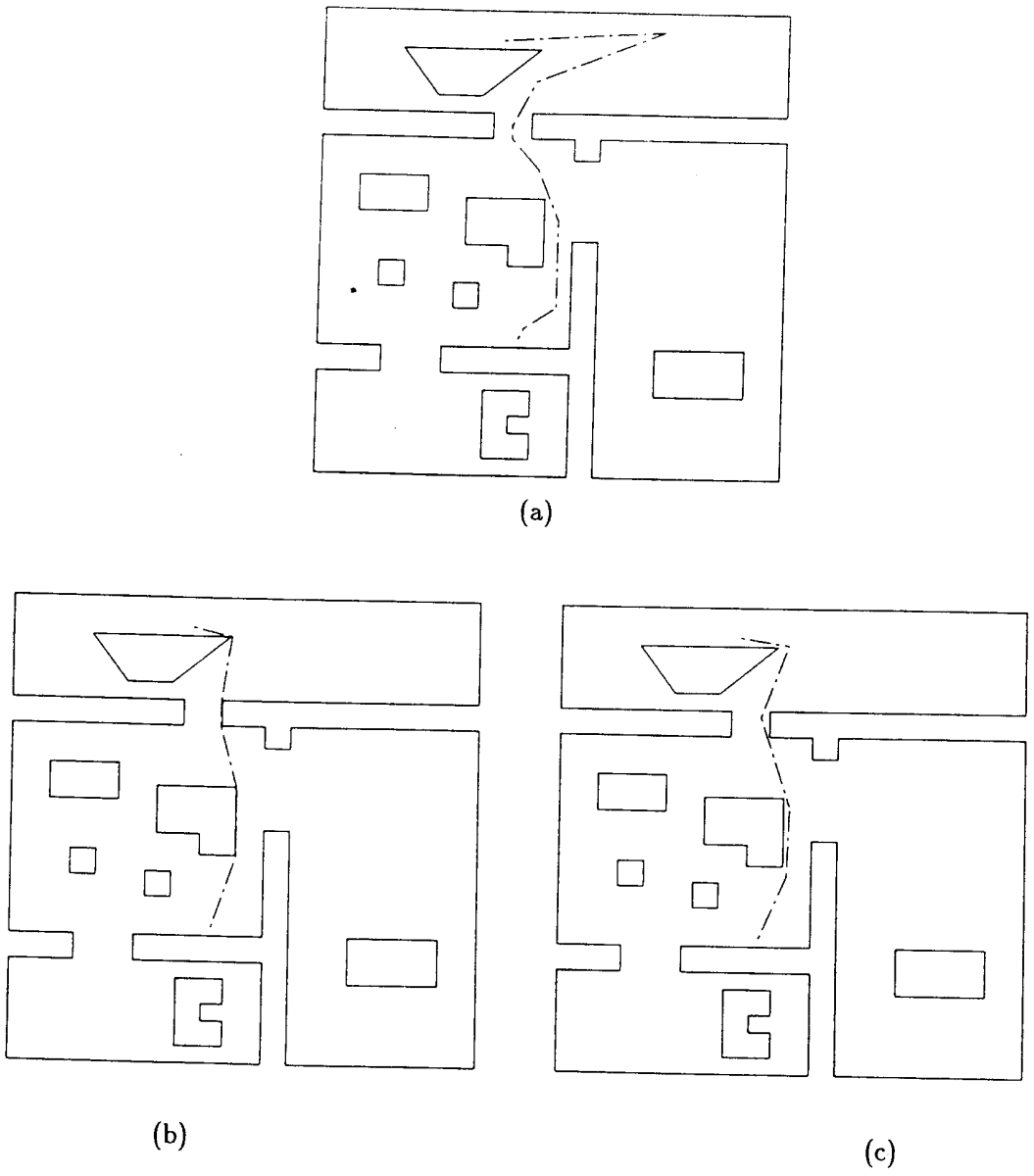


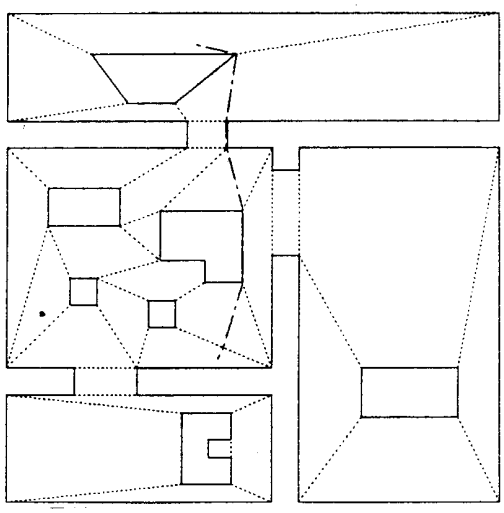
Figure 29: Dependency on decomposition method (A*-1)

In this case, the decomposition method of Figure 22b (most concave vertex, leftmost victim) was used, not that of Fig. 22a (most concave vertex, most opposite victim) as in the previous cases. Although the start-goal points and path improvement techniques are identical with Figure 28, the path produced here is of lower cost. This is a consequence of the decomposition strategy used.

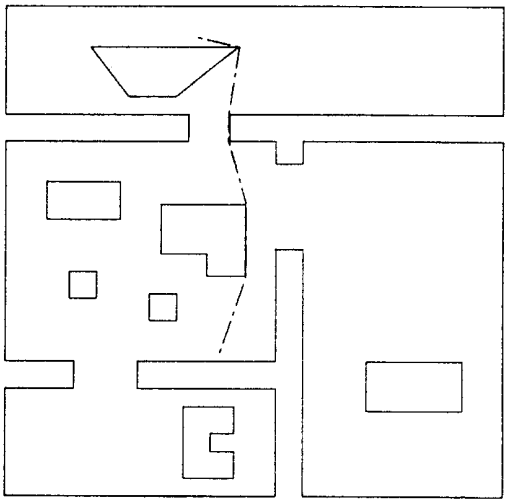
a) Initial coarse path.

b) Improved path (safety margin 1 foot).

c) Improved path (safety margin 10 feet - see note for Fig. 26d.)



(a)



(b)

Figure 30: A*-3 path planning

Contrasting this figure against Fig. 28 (which uses the same decomposition method as is used here), A*-3 search provides the same lower cost path as was seen in Fig. 29 (although a different decomposition strategy was used in Fig. 29). A partial decoupling of the decomposition method and path finding strategy is in evidence.

a) Initial A*-3 search path. Note the selection of points near edges as well as midpoints.

b) Final improved path. Almost identical to raw path (a) in this case.

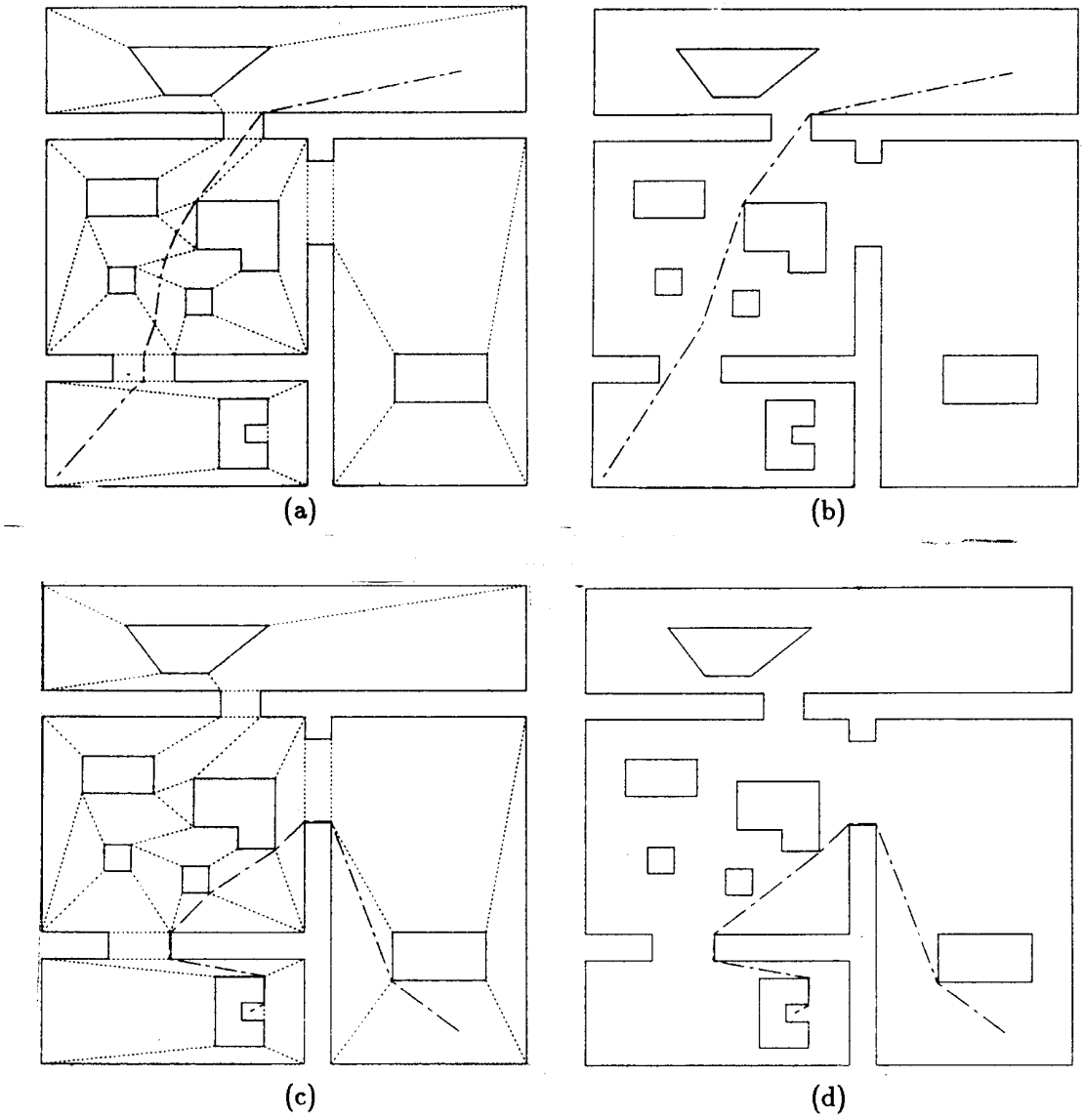


Figure 31: More A*-3 path planning

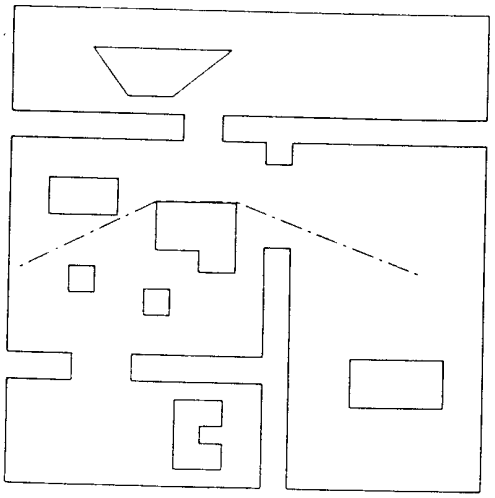
Here it can be seen that the A*-3 method gives no improvement over the A*-1 method for the cases in Figs. 26 and 27. The extra computational cost proves unnecessary.

a) Same start and goal as in Fig. 26 but using A*-3. The initial raw path is shown as a dashed line in relation to passable meadow boundaries (dotted lines).

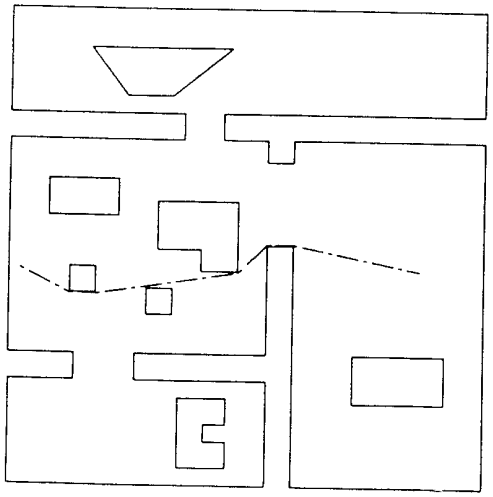
b) Final improved path for (a).

c) Same path as in Fig. 27 but using A*-3. Initial raw path.

d) Final improved path for (c).



(a)



(b)

Figure 32: A*-1 versus A*-3

In tight quarters, A*-3 can make a slight difference. The gain in this example, however, is small (on the order of 2% of the overall cost).

a) A*-1 final improved path.

b) A*-3 final improved path.

possible start and end points within any given map, no single map-building strategy was clearly superior.

For A*-3 noticeable improvement occurred. Figure 30 clearly shows the ability to produce a better path using the same convex decomposition than was the case with the A*-1 method (Fig. 28). Figure 32 confirms these results. The raw paths of A*-3 are generally close to or the same as the final path, whereas this is much rarer in the case of A*-1. The computational penalty, however, can be significant as the search space is considerably larger and is discussed in Appendix A.

Hopefully the diagrams (Fig. 26-32) give a feel for just what a reasonable path is. There are no unnecessary or unexpected turns. When two or more choices are available, if one is significantly more advantageous than the others, the better one will be chosen. If there is only a slight advantage (you might need a ruler to tell as in Fig. 32), one of the best will be chosen. No claims for overall optimality are made, although if suboptimal results are produced for these cases they still qualify as reasonable. Restating that optimality is perhaps a misplaced notion in a dynamically changing world (without constant replanning), the value of spending high computational effort in ensuring absolute minimal costs in the mobile robot domain is unjustified.

The limitations for other representation forms show the advantages inherent in the use of the meadow map approach; these include:

- regular grid: high memory and search cost and digitization bias resulting in sub-optimal paths;
- pure vertex graph: optimal paths only in the context of shortest distance and not amenable to safe path production;
- Voronoi diagrams: more difficult to arrive at short paths and additional representational features not easily embedded.

A major advantage lies in the ability of the meadow map to incorporate virtually any additional representation desired to guide vehicle localization, sensor processing, etc. (see the discussion on the feature editor in Section 2.2). If the paths produced are sub-optimal in the global context and thus are only "reasonable", that is a small price to pay for the versatility and lower memory costs afforded by this representational strategy.

§4. Multi-Terrain Extensions

One of the principal contributions of this work lies in its extension to handle differing terrain types. Previously the regular grid has been the principal representation used to deal with diverse ground covers [88]. Certainly, for the planner to produce realistic paths in outdoor scenarios, a reflection of the different terrain types must be taken into account by the navigator. Some terrain types will be more costly to traverse than others (e.g. gravel or grass as opposed to concrete). We do not want to exclude these different terrains as navigable areas, but yet we don't want to lump them into one uniform terrain type. The traction of the vehicle will depend on the specific surface encountered and more slippage is expected to occur on gravel than on pavement. The cost in terms of positional uncertainty can be high on loose ground. On the other hand, if a significant reduction in the total distance to be traversed from start to goal can be obtained (and associated reduction in time cost), the tradeoff of increased positional uncertainty for greater time savings may be warranted. In some cases the total amount of positional uncertainty gained by traveling over poor surfaces may be substantially less than that garnered by traveling over a superior cover due to the much shorter distance the robot may travel by taking a rougher terrain short-cut.

Another sticky point lies in terrain borders where one ground cover type ends and another begins. If the robot keeps one wheel on one terrain type and the other(s) on a different cover, disorientation can be rapid. One of the goals of the representational strategy used here is to prevent the robot from straddling terrain borders. This is accomplished by the creation of transition zones which separate the ground covers and define clean traversal points. Forbidden zones are also produced which prevent the robot from navigating at the corners of terrain boundaries (regions which typically are expected to be very problematic in terms of maintaining proper localization).

This section first describes how the map-builder accommodates multiple terrain types through the construction of transition zones and their appropriate features. A description of how the navigator has been modified from the uni-terrain model to accommodate path-planning through this extended representation is then presented.

§4.1 *Multi-terrain map-builder*

The extended map-builder is built from the uni-terrain map-builder described in Section 2.1. The algorithm appears in Figure 33.

The input structure of a terrain region is identical to that of the previous map-building algorithm: a list of border and obstacle vertices. This region is decomposed in exactly the same manner as was done previously. Nothing labeling a terrain border is present to identify it as such to the algorithm. Initially, all borders of each terrain region and its enclosed obstacles (which may later turn into other terrain regions) are initially labeled as impassable.

Wherever two different terrain types are found to touch, rectangular transition zones are built allowing a limited type of traversability between them. As a side effect, forbidden zones (corners of intersecting bounding regions), are marked as off-limits for later path planning purposes. This restriction ensures that any path taken across a transition zone will result in a minimal distance path. The transition zone is tagged for recognition by the path planner and other components of the overall system dealing with long-term memory. The details of this process follow.

After the initial terrain area is decomposed, the map-builder algorithm keeps accepting new ones until none remain. After each terrain area is decomposed in isolation, a matching algorithm is run on each new terrain convex region to see if it shares any common edges with any of the previously decomposed regions. This match is performed on the ungrown vertices (or else they would never match). If a match is identified, evidenced by at least the partial overlap of any impassable edges of two different terrain types, a transition zone is built.

The transition zone is a special region connecting two differing terrain types. Most of the data for transition zone construction is already available from the matching process. Basically, the two grown edges, each representing the common border of each matched region, are used for two of the edges of the transition zone (Fig. 34a). This gives a distance across the zone equal to the robot's diameter plus two times any safety margin that was used in the growing (or shrinking) of the initial terrain regions (Fig. 34b). The initial zone consists of the four vertices of the two matched edges.

It is highly desirable to minimize the time it takes for the robot to cross a transition

MULTI-TERRAIN MAP-BUILDER ALGORITHM

```
DO WHILE no more terrain to add
    Run the uni-terrain map-builder (Fig. 15) on a terrain region
    Tag all resulting free space regions with a new terrain identifier
    Match borders of new free space regions against the
       terrain free space regions already produced
    IF matches exist
        Build transition zones connecting terrain types
        Add these transition zones to free space regions
    ENDIF
ENDDO
```

Figure 33: Multi-terrain map-builder algorithm

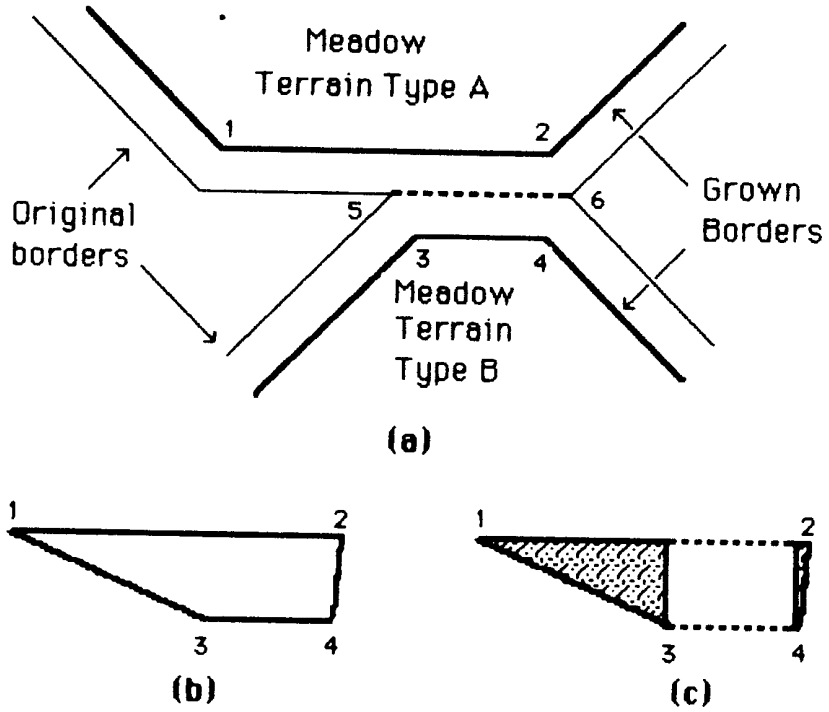


Figure 34: Transition zone construction

a) Initial bordering terrain types. Terrains A and B share a common edge from vertex 5 to 6.

b) The initial transition zone is built by connecting the four vertices of the bordering C-space lines.

c) The initial region is converted into a rectangle yielding the final transition zone. The resulting forbidden zones are shown as shaded areas.

zone, implying a normal straightline path. Consequently the initial polygonal representation is converted into a rectangle (Fig. 34c). The new edges produced (sides of the rectangle) are labeled as impassable, producing small forbidden zones which the planner construes as unnavigable. Any path that is produced by the path planner is guaranteed to be normal to the original matched edges, thus ensuring the smoothest and fastest transition possible from one terrain type to another. Finally, appropriate passable links are made to connect the new transition zone and the two bounding free space regions of the different terrain types.

The traversability factor (used for costing in path planning) should be high for transition zones due to the problems associated with terrain changes. Currently the traversability of a transition zone is defaulted to the sum of the traversabilities of the two bordering terrain types. This value can be readily changed if appropriate via the feature editor (Section 2.2).

§4.2 *Multi-terrain Navigator*

The navigator must be modified somewhat to ensure that the path produced is reasonable in the multi-terrain case. The only components of the navigator that must be changed are the path improvement strategies. This includes both the straightening and tautness components. No modifications whatsoever are necessary for the search component because the terrain cost is included in the cost function. As the transition zone is rectangular, any path produced by the A*-1 method through the midpoints of passable regions crossing over different terrain types is guaranteed to result in a straightline across the transition zone. The A*-3 case occasionally requires slight path preprocessing to ensure a perpendicular crossing of the transition zones prior to improvement. Unfortunately, modifying the path improvement strategies (for both A*-1 and A*-3) was non-trivial and required the implementation of a relaxation algorithm, limited to relaxing the terrain crossings only. Previously, terrain traversability costs were ignored during the path improvement process, yielding shorter but more costly paths over more difficult terrain. The relaxation approach readily ensured that traversals across the transition zone remained perpendicular to the border edges while still producing low cost paths dependent on the nature of the ground cover.

The algorithm for multi-terrain path improvement is shown in Figure 35. Actually,

the complexity is somewhat greater due to special case treatment (start or end within transition zone, entire path in transition zone, etc.). Sedgewick [114] states that "special cases ... are the bane of geometric algorithms", and I am in firm agreement with him.

The algorithm proceeds as follows: the previous path improvement strategy is first run within the framework of each terrain type in isolation. This is the identical algorithm as described in Section 3.2 but restricted to individual terrain types. To reduce the cost of the relaxation later, the transition zone crossings are then slid in the same manner as was done for the individual meadow border passages, with one exception. Both crossing points on the transition zones are slid in tandem, insuring a perpendicular passage across the transition region. This step generally reduces the overall distance the transition zone crossings will have to be moved during the relaxation phase, thus reducing computation time. Any previously unmoved vertices within the regions themselves are then retested to see if sliding will lower the overall cost. If necessary, additional path straightening is then performed.

Although avoiding a relaxation method for path improvement was an initial design goal due to perceived high computational costs, (as in relaxation on a regular grid), it eventually became necessary to resort to one. The cost associated with this relaxation (see Appendix A) is not particularly high however, due to the preprocessing on the path and, more importantly, only the transition zone crossings are relaxed, not all passable borders. The algorithm used is fairly standard: displace the transition zones an increment in both directions and measure the lowest cost. Use the new lower cost point as the starting point for the next displacement. Keep repeating until any displacement results in a higher or equal cost path. Convergence is guaranteed using this standard hill-climbing methodology. The time for convergence is determined to a large extent by the displacement size and on the number of terrain crossings. The results for the worst test cases in the lab have yielded times for the relaxation component that are not disproportionate (typically the same order of magnitude) with the other components of the algorithm (see Appendix A). The best case (no transition crossings) is virtually identical to that of the uni-terrain results (Section 3.3); the average case results in slight increases in path improvement time. Finally, path straightening is reattempted within the context of each terrain type before final release to the pilot.

MULTI-TERRAIN PATH IMPROVEMENT ALGORITHM

Accept a coarse path from search component of navigator.
Run tautness and straightness component of uni-terrain path planner
    on each part of path within a given terrain type (Fig. 24).
Slide only the transition zones as in previous tautness algorithm.
Run tautness and straightness component again on each part of path
    within a given terrain type (only on previously unmoved vertices).
Relax path by settling transition zone crossings into
    a minimal cost point.
Restraighten if necessary.

Figure 35: Multi-terrain path improvement algorithm

§4.3 *Results*

A schematic model of the environment outside the Graduate Research Center was used for the outdoor terrain examples. Five different terrain regions are present: concrete, two disjoint grassy regions, a gravel path and a parking lot. For the purposes of path planning: the traversability of the concrete and the parking lot was set to 1.0, grass 1.5, and gravel a factor of 1.2 (these are relative values: the higher the number, the more difficult to traverse). The gravel path, although rough, has the decided advantage of path borders, which make path-following strategies available that are not useful on grass. The terrain types and their associated transition zones can be seen in Figure 36.

In Figures 37 through 39, the results of the path planning algorithm are illustrated. The A*-1 search method was used for all these cases. Sub-figures 37a-39a shows the initial path through the search space. Note in this and all other cases the perpendicular passage through the transition zone is evident. Sub-figures 37b-39b show the improved path before transition zone relaxation. Sub-figures 37c-39c display the final path after relaxation and post-relaxation straightening.

§5. Cartographic subsystem

The details and construction of long-term memory by the cartographer's map-builder process has been described in Section 2. The uncertainty management subsystem is described in chapter 7. What remains to be discussed here is the structure and maintenance of short-term memory. The subsections following will present the STM structure, the STM manager's role in creating and maintaining the perceptual level of STM, and the cartographer's meadow instantiator process which provides the LTM context for STM that is drawn upon by the pilot in the event of motor schema navigation failure.

§5.1 *Short-term memory structure*

Short-term memory is a bi-level structure (Fig. 40). Its primary purpose is to provide information for navigational purposes (i.e. it does not serve the same purpose as VISIONS STM). At the base level, it consists of a group of meadows from LTM which define the context for the current pilot goal. Recall that LTM consists, to a large extent, of a

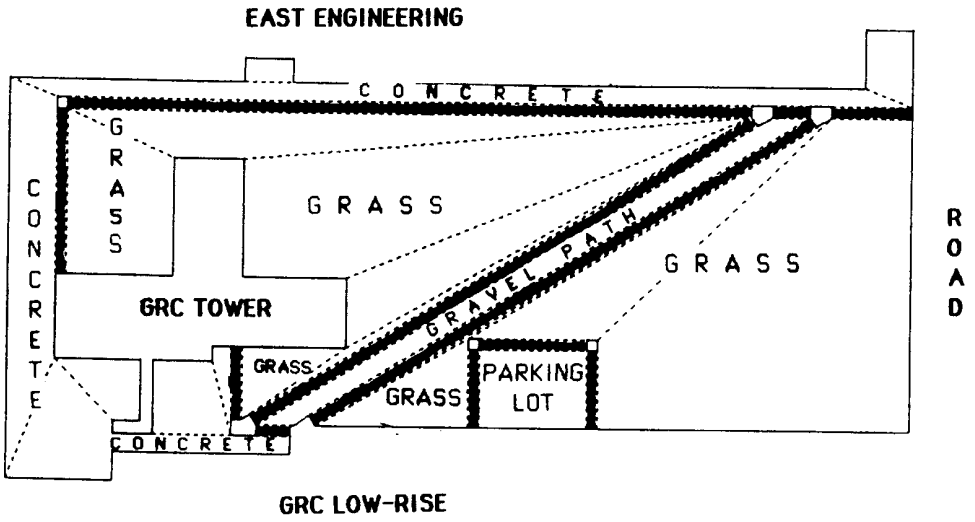


Figure 36: Multi-terrain map

A *schematic* diagram of the area surrounding the Graduate Research Center. All impassable regions are represented as solid lines, passable meadow boundaries as dotted lines. The passable transition zones are the solid rectangles. The different terrains (grass, concrete, parking lot and gravel path) are labeled. (Scale approximately 320' by 180'. This is much smaller than is actually the case, but necessary to clearly show the transition zone-path relationship in the figures to follow, i.e. the transition zones are larger than they would appear otherwise).

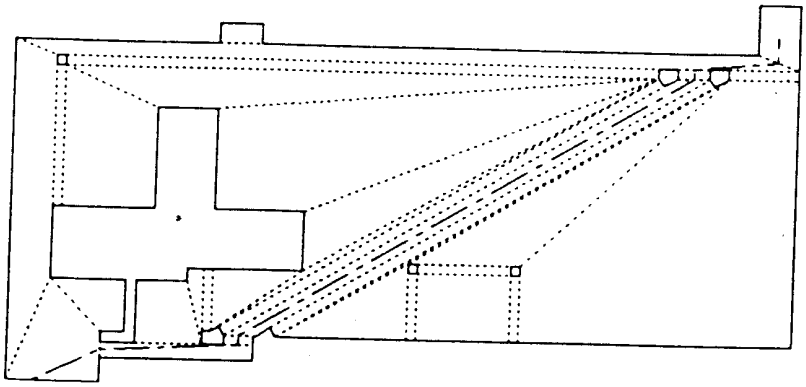
Figure 37: Multi-terrain path planning example

The start point is in the lower left corner on concrete with the goal in the upper right on concrete. The path planner decides it is more efficient to take the gravel path to achieve its goal, requiring the traversal of two transition zones.

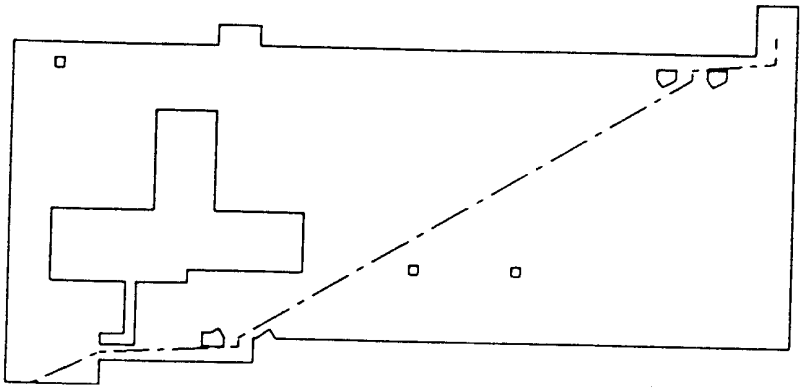
a) Initial raw path from A*-1 search through midpoints of passable regions. The cost function includes a traversability factor dependent on terrain.

b) The same path without the passable borders. Note the forbidden zones present at the edges of transition zones.

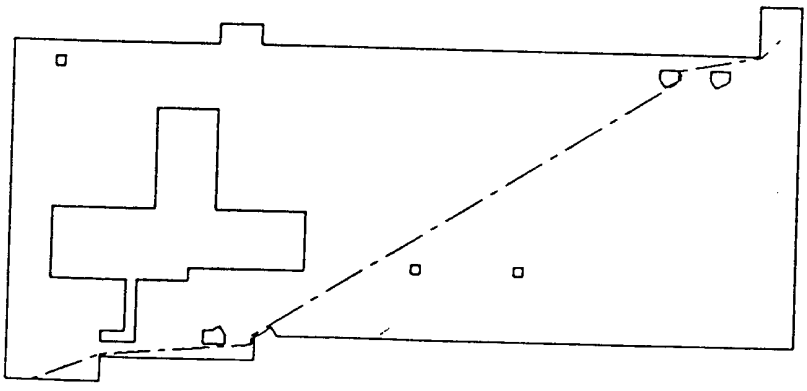
c) The final improved path. Note how the total length over concrete was lengthened while the distance over gravel shortened (safety margin 1 foot).



(a)

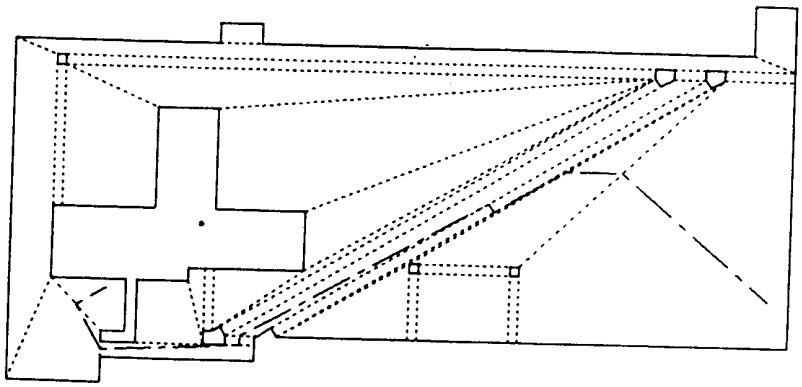


(b)

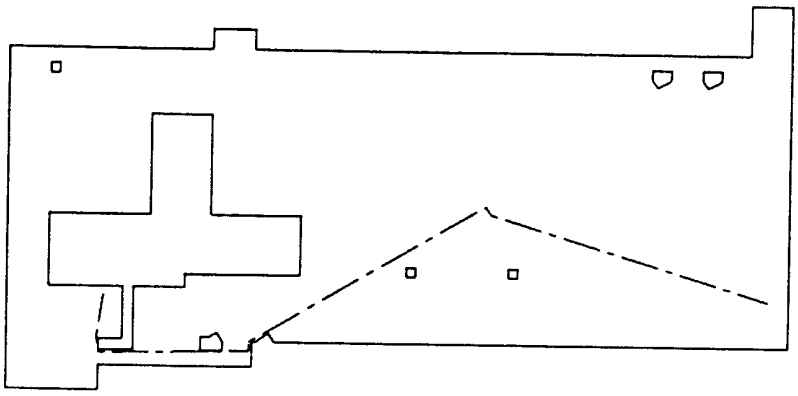


(c)

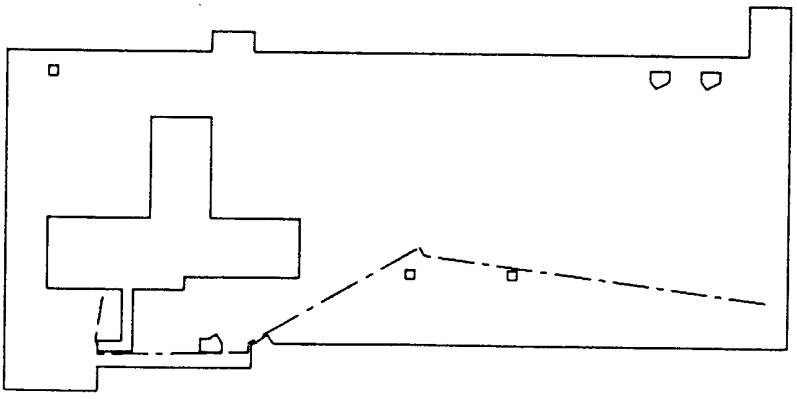
Figure 38: Another multi-terrain path planning example
Start in lower right, goal in lower left.
a) Initial raw path - crosses two transition zones, grass to gravel, gravel to concrete.
b) Improved but unrelaxed path. Grass to gravel crossing is at midpoint of transition zone.
c) Final relaxed path - note the repositioning of the grass to gravel crossing.



(a)

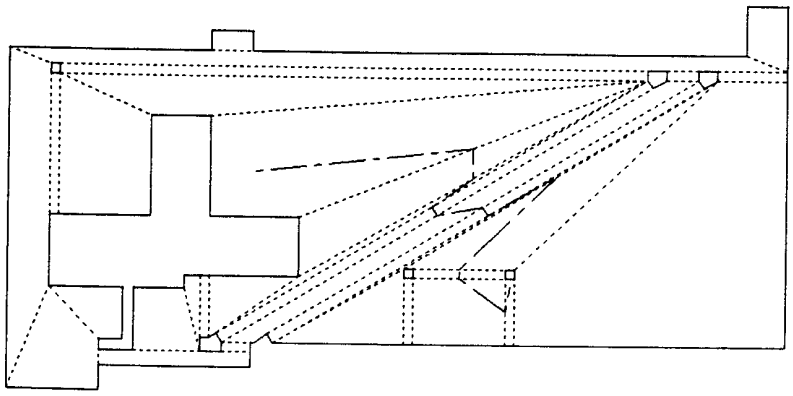


(b)

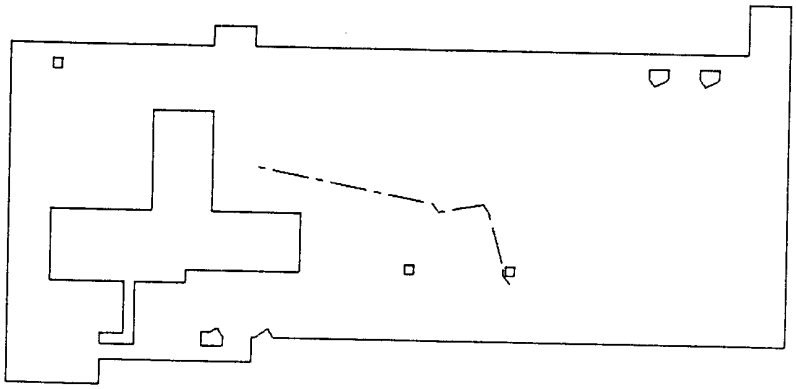


(c)

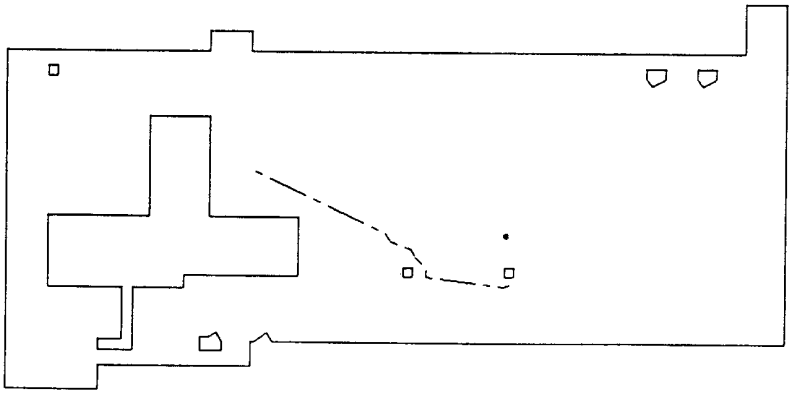
Figure 39: Yet another multi-terrain path planning example.
Start in lower right, goal in upper left.
a) Initial raw path.
b) Improved but unrelaxed path.
c) Final relaxed path.



(a)



(b)



(c)

connectivity graph of these meadows. The base level defines the context of the world map against which sensory acquired information to be applied. These meadows are moved into STM by the meadow instantiator process (Sec. 3.2). The overlying STM top level contains a sensory-based model built up by the cartographer as the robot travels through the world. A local frame of reference based on the current pilot leg is used for this level. The STM manager incorporates the sensor data as it is received into this top layer. The details of each level follow.

The instantiated meadow level is crucial for the correct selection of motor schemas by the pilot. Basically it contains pointers to the parent LTM meadows and tags indicating if the meadow is the start, end, on-path, adjacent, or other type of meadow. This enables the retrieval of terrain characteristics, potential landmarks and other information pertinent to the robot's current position in the world without a time-consuming search of LTM. The provision of context for the sensory built top-level of STM is also of major significance. The top level of STM, consisting of a regular grid embedded with sensor information, should be viewed as an overlay on these meadows (Fig. 40). Thus, when the pilot needs to reorient the path for whatever reason, spatial occupancy information from the sensor (top) level can be moved into the base-level instantiated meadows. The meadow "fracturing" process that is performed by the pilot, when the need arises (described in Section 6.2 below), recomputes locally the robot's path based on information from both sensor data (represented in high-level STM) and LTM models. This is more efficient than reinvoking the navigator to compute a global path anew.

The top level of STM uses a grid representation of space. It is based on extensions of the method used by Moravec and Elves [92,45] for interpreting sonar data. Navigational space is tessellated into a grid, with each square containing information regarding the occupancy of the area (whether it is free space or filled with an obstacle). This map is built up from sensor readings acquired from the robot as it travels. These readings taken from the robot's egocentric frame of reference must be combined into a robot position-independent representation. The frame of reference used for the grid representation is a "local-global" coordinate system. It does not correspond directly to the robot's egocentric vantage point nor does it match the LTM global world model. It is a local model of the world that is built solely from egocentric sensor measurements taken from the vehicle. The resulting grid must be correlated against the lower instantiated meadow level of

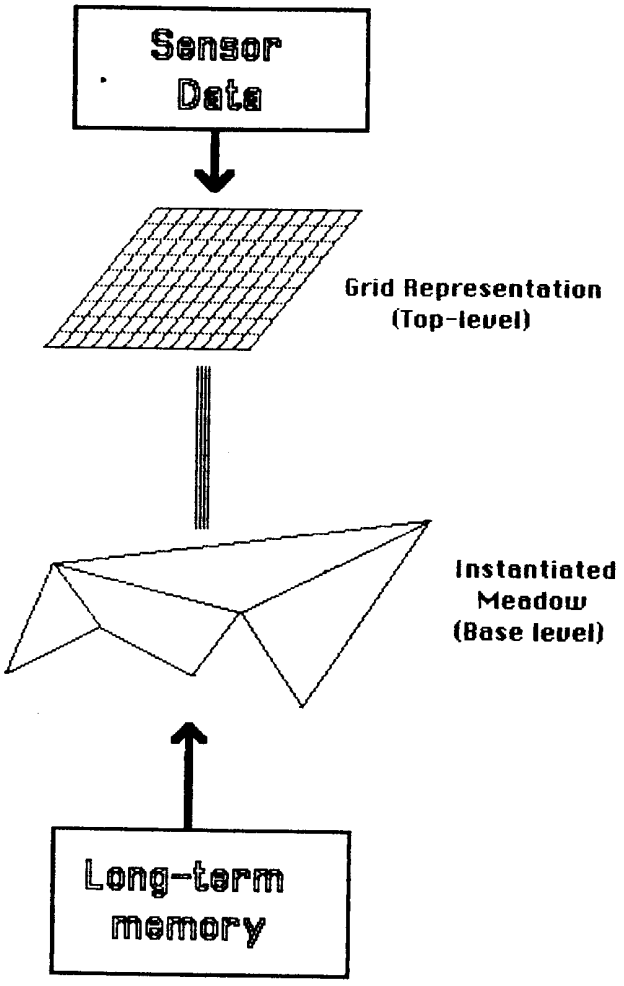


Figure 40: Short-term memory

STM (which is represented in world coordinates from LTM) so that the robot's bearings can be measured against known world features. The uncertainty in the top level of STM is expressly represented through the use of a numerical value whose absolute value increases for a particular cell based on confirmatory sensor data and decreases based on contradictory data. The STM manager, described below, handles the maintenance of this level of STM.

The spatial resolution of the top-level grid is typically 64 feet by 64 feet. Its structure need not be square but was so decreed, for the first implementation. As the path leg is generally longer in one dimension a 128 by 32 grid might be more appropriate for future generations of AuRA. This grid moves through navigational space each time the robot undertakes a new leg on its journey. The cells consist of the following data:

- Occupancy Value
  range: −1 (highly probable to be unoccupied and hence unnavigable)
  to 1 (highly probable to be occupied. 0 denotes no information - (as in [45])

- Symbolic tags
  Again with certainty level (range 0 to 1) This would include not only LTM landmarks but also path edge information and terrain-type identifier where appropriate. Several may be active for each cell.

A major problem for this approach, as discussed by Brooks [25] and handled by Smith and Cheeseman [118], is the incorporation of multiple measurements from different uncertain positions into a single representation (i.e. the STM regular grid). There is no simple solution. In this case, the assumption must be made that the position from which the robot is currently taking its measurements is relatively well known. Fortunately, the STM representation need only be used when reflexive motor schema-based navigation fails, which should be relatively rare.

A form of environmental learning is also feasible when information regarding obstacles or features of high certainty could be moved from STM into LTM for future reference. This is left for future AuRA implementations.

Certain other information is stored in STM. This includes the route-list developed by the navigator. Event flags are also present indicating items such as the deposition of the route by the navigator as well as route completion by the pilot. Current leg pointers are

also maintained here. These flags are used for inter-process synchronization. Bounds for STM extent are also present. A point estimate of the robot's position is also maintained based on shaft encoder data (independent of the spatial error map).

§5.2 *STM meadow instantiator process*

The meadow instantiator is a separate process running under the control of the cartographer. Its operation is fairly straightforward. On start-up it initializes the base-level of STM. It then waits until the navigator has placed a route into STM and has set the route-deposited flag. Meadows are then moved into STM based on the first leg of the route. The meadow instantiator then waits until the pilot signals it has completed its leg. The old meadows are then deinstantiated (made inactive) but not removed from STM (unless available memory requirements necessitate it). The next leg of the navigator's route is fetched and the relevant inactive meadows already in STM are reactivated, while any new meadows needed are accessed from LTM. This process repeats until the navigator's route is completed.

§5.3 *STM Manager*

The STM manager's role is to modify short-term memory based on arriving interpreted sensor data. The approach of Moravec and Elves [92] for incorporating multiple sensor readings from different spatial locations into a single grid representation is used. Although their work to date deals solely with ultrasonic data, information from other sensor modalities such as vision can be incorporated.

No single sensor reading is sufficient to guarantee that the existence of an obstacle or other environmental object is present. Instead, multiple readings from different locations are merged using a probabilistic approach to build up certainty in the position of an obstacle. The ultrasonic reading does not give a precise environmental location of an object. Due to the nature of the sonar scan, only a wedge of possible locations is available. The multiple readings are folded together to yield a measure of the uncertainty of occupancy or free space for each grid square [92]. The STM manager is built around code imported from CMU (Moravec and Elves' work). Only minor modifications have been made to enable it to be tied into the AuRA architecture. Figure 41 shows a typical map built up in STM from multiple sonar readings of the UMASS robot lab.

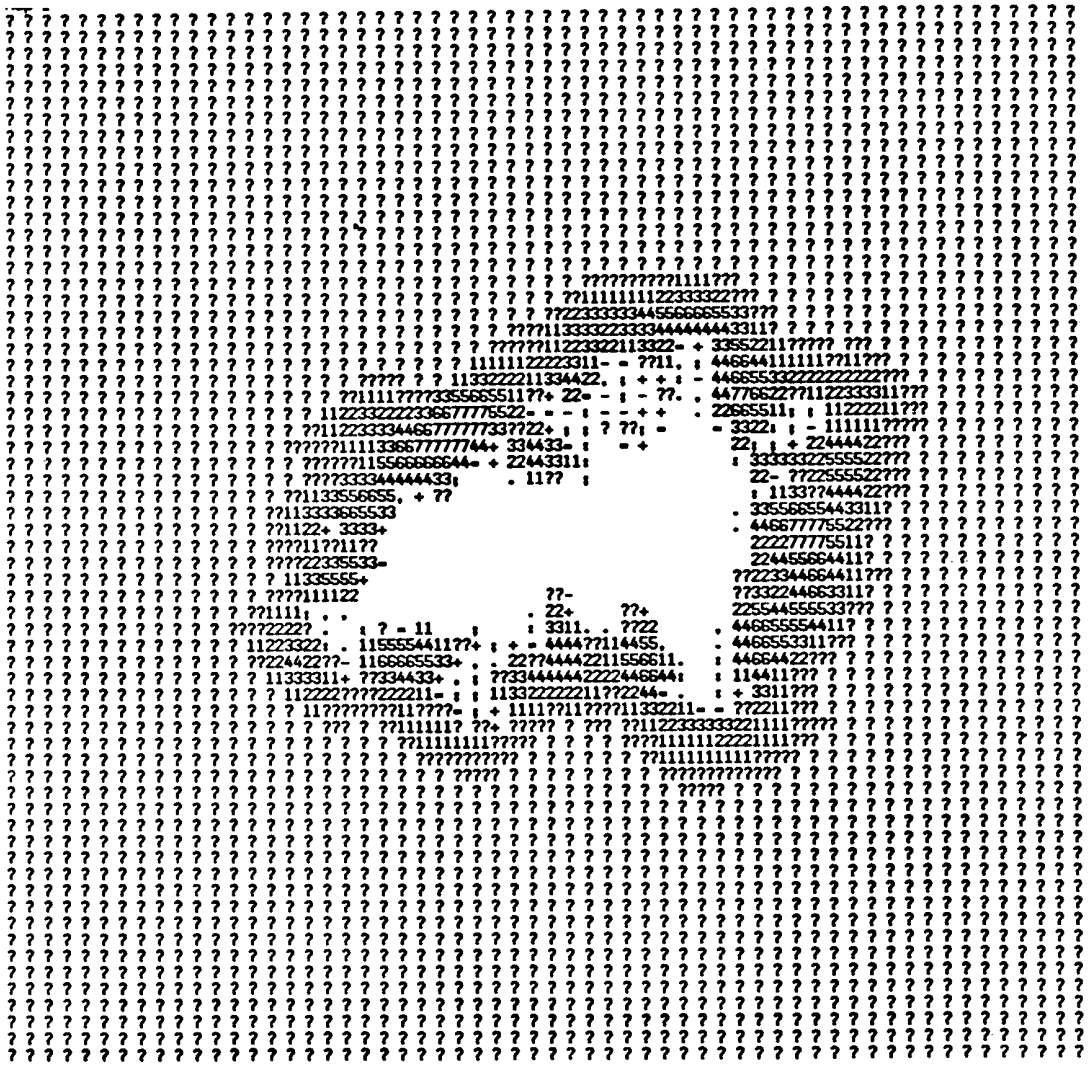


Figure 41: Top-level of short-term memory (Occupancy view of the robot lab)

Grid values indicate certainty of occupancy - numbers denote the likelihood of being occupied, while whitespace and punctuation marks denote the likelihood of being unoccupied. (This is a view of the ground-plane from above).

Based on code supplied by Hans Moravec [92]

The STM manager accepts arriving reports from the interpreter processes in the perception subsystem and incorporates this data into the STM grid. This perceived world model is built up concurrently and independently of the motor schema manager. It is referred to by the planning system only upon failure of the motor schema manager to achieve the specified pilot goal. This is detected by the exceeding of a hard real-time deadline for goal attainment or by the robot's velocity dropping to levels that are deemed too slow for successful completion. The pilot then draws on this data, merging it into the instantiated meadows in STM and then fracturing the meadows (Figure 42) to allow local navigational replanning. This meadow fracturing strategy used for local path reorientation by the pilot is described below in Section 6.2.

§6. Mission planner and pilot

AuRA's planning subsystem consists of the mission planner, navigator, pilot and motor schema manager. Motor schema based navigation is described in Chapter 5 and the experimental motor schema system is described in Chapter 8. What remains to be discussed in this section are the implementation details of the mission planner and pilot.

§6.1 *Mission planner*

The mission planner's implementation for the first pass design of AuRA is at best rudimentary. Reviewing the goals of the mission planner, they are: to perform spatial reasoning, the determination of navigation and pilot parameters and modes of operation, the selection of optimality criteria and the handling of navigator failure. As the mission planner serves as the interface to the human commander, a natural language front-end facility is also desirable. The mission planner's chief function is to provide a series of specific subgoals, based on a high-level request, for the navigator to act upon. These requests might include such things as:

- Survey or reconnoiter an area
  (search for something lost, make reports on unusual events, etc.)
- Obtain a particular list of items found in different locations
  (requiring the determination of where the items are and then an appropriate order-

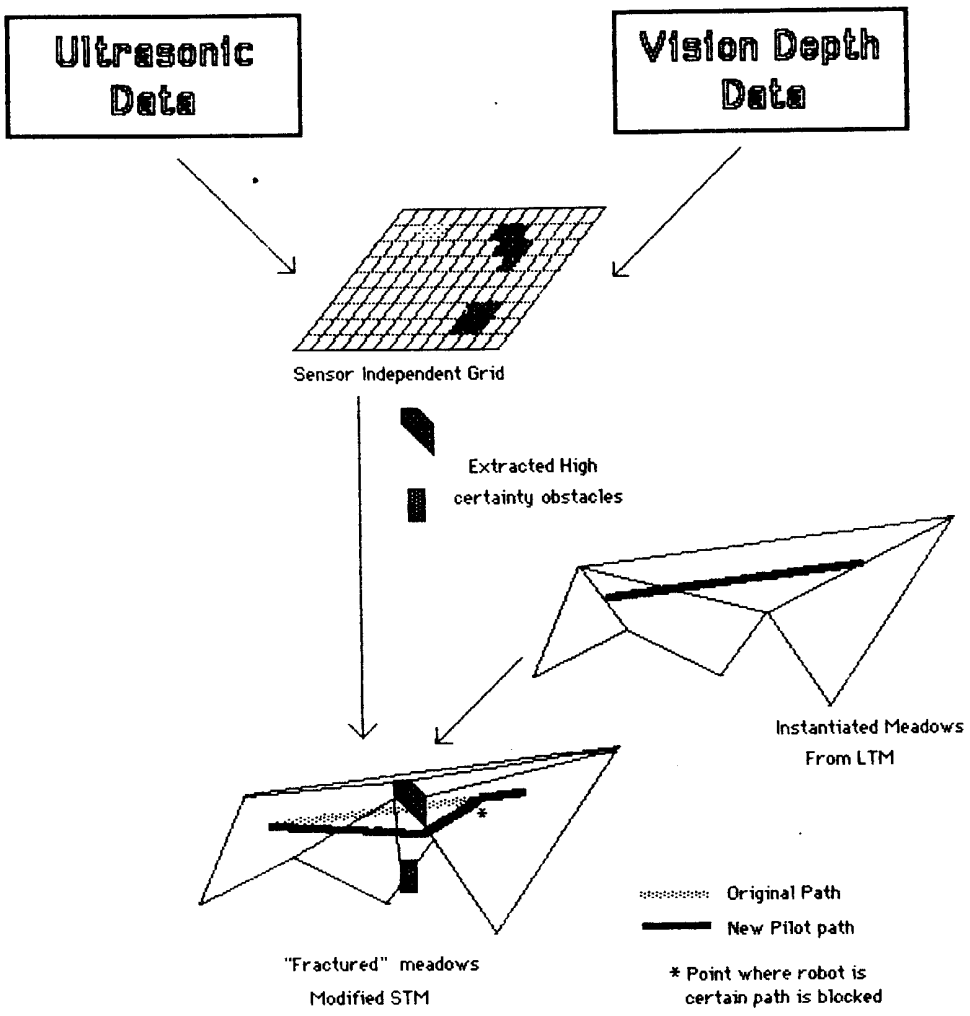


Figure 42: Meadow fracturing
The two polygonal obstacles with relatively high certainty are extracted from the top-level grid and are embedded in the low-level instantiated meadows. This enables the pilot (Sec. 6.2) to recompute the path within a local context, thus avoiding reinvoking the navigator.

ing for their retrieval)

Currently, the mission planner performs none of these functions other than acting as the interface to the commander (without natural language ability). All goal ordering is established by the human agent and is passed through the mission planner without interpretation. The mission planner also provides a facility to set the parameters that affect navigator and pilot operation interactively, but does not interpret the "mission" to automatically determine those settings.

This rudimentary form reserves the rightful function of the original mission planner design for a more complete implementation. The responsibility is passed upward to the human commander and is not relegated to lower levels of the planning hierarchy. It is hoped, and research interest exists at UMASS, that other researchers will complete and/or enhance the capabilities of the existing mission planner. New versions should be readily incorporable into AuRA.

§6.2 *Pilot*

The pilot serves a dual purpose in the AuRA architecture. First and foremost is its responsibility to analyze available data regarding the current navigator leg to be completed. From this information (gleaned from both LTM and STM) it selects appropriate motor and perceptual schemas to be instantiated within the motor schema manager. The pilot then suspends itself while the motor schema control system manages the actual path traversal. The second role of the pilot is to handle failure of the motor schema manager to reach its desired goal. This can be evidenced in two ways: by the cessation of motion by the robot without goal attainment, or by a clock time-out signaling failure to reach the goal within a pilot-established hard real-time deadline. The pilot then attempts to reroute the path *locally* based on information stored in STM by the cartographer. Only if this pilot-based reorientation procedure proves unsuccessful is the navigator reinvoked to recompute a new *global* path in light of the newly discovered environmental blockage.

The following subsections describe these two roles of the pilot.

Pilot schema selection process

In order to provide appropriate motor action for a specific path traversal, the pilot must access information that is present in both long- and short-term memory as well

as the current goal that the navigator has established for the pilot. Long-term memory contains the terrain characteristics, landmarks and other relevant characteristics which the pilot is required to draw upon for motor schema slot-filling. Short-term memory contains the instantiated meadows (Sec. 5.1) that are relevant for the current pilot leg. Blackboard values specifying motion characteristics (velocity, acceleration, etc.) are also drawn upon. These are generally set by the mission planner and possibly modified by the homeostatic control system. The navigator's subgoal is passed to both the pilot and cartographer through global memory and is used by the pilot to establish the hard real-time deadline for the navigator and to set failure limits for certain motor-schemas (e.g. **move-ahead**). Information is extracted by the pilot from all these sources and moved into a *fact-base* at the start of each pilot leg.

The approach used in the first pass implementation uses a simple set of rules and a primitive inference engine to select appropriate motor schemas based on the current navigational subgoal. A *rulebase* is maintained separately for the pilot's use. The rules contained therein are applied by the pilot to the current context (the *fact-base*). The result is a list of parameterized motor and perceptual schemas (drawn from a set of schema templates) customized for the particular navigational leg. Some schemas are invariably produced (i.e. **avoid-static-obstacle** schema), while others are produced as the current context dictates. Slot-filled **find-landmark** schema templates, in particular, are produced by the pilot by application of the rules in the *rulebase* to the specific landmark data in the *fact-base*. Activation conditions, useful perceptual schemas for identification, and other related criteria are passed with the schema itself to the motor schema manager.

These motor and perceptual schemas effectively exist in three forms: the naked unparameterized (but with specified defaults) schema templates, residing as structures in LISP code in the schema database; parameterized, but as yet uninstantiated, schemas passed from the pilot to the motor schema manager; and the schema instantiations themselves (SIs) existing within the motor schema manager. The schemas passed to the motor schema manager undergo a transformation into a form compatible with the schema shell. Only the information necessary to flesh out the schema shell SIs is passed from the pilot to the motor schema manager, keeping the communication bandwidth at reasonable levels.

Local path reorientation

The other function of the pilot is to handle failure of the motor schema manager to attain its specified goal. The potential field methodology is vulnerable to failure due to local potential minima or isolated peaks (see Chapter 5). Although this should be relatively rare due to the navigator's selection of a path that avoids all modeled obstacles, the presence of significant numbers of unmodeled obstacles can cause the potential field based system to falter. When this occurs, as evidenced by exceeding a real-time deadline to achieve the pilot's established goal (for cycle detection) or by the vehicle's velocity dropping to too low a level, the pilot is reinvoked.

The pilot, upon motor schema manager failure, attempts to reach the navigator's subgoal by moving obstacles from the STM grid representation into the instantiated meadows so that they now serve as modeled obstacles for the path planner. These newly discovered obstacles are first merged into the border created by the instantiated meadows. The same algorithms used by the cartographer to build LTM are then used to decompose these instantiated meadows into a new series of convex regions. The navigator's path finding algorithm is then run within this limited context. The new series of path legs gives the robot a new approach towards negotiating these unmodeled obstacles. If a path is unattainable, the navigator is reinvoked to find a new path on a global basis. This level of planning within the pilot is "local-global" as it draws both upon the perceived world model as well as the instantiated meadows taken from LTM. Figure 42 illustrates this process.

It should be remembered, however, that this role of the pilot should be infrequently used. Nonetheless, due to the fact that potential fields are susceptible to local sensing problems, techniques to handle this difficulty must be available.

§7. Summary

In order to produce a reasonable path through a partially modeled environment, a hybrid vertex-graph free space representation was chosen for a long-term memory model of the world. This meadow map decomposes free space into a group of connected convex regions. Data for landmark recognition, localization, uncertainty modeling, and the like,

can be associated with these regions or their obstacles through the use of a feature editor. Although LISP might be a preferable language due to its symbol manipulation capabilities, all coding was done in C to insure rapid processing to meet the demand for real-time response.

Multiple terrain situations are accommodated by extending the basic algorithms to include the construction of transition zones. These zones assure minimum distance traversal when the robot changes terrain types, minimizing the increase in positional uncertainty inherent in this maneuver. When facet models are applied to the meadows to reflect topography, and the ground plane assumption discarded, this method will become even more powerful for navigation in outdoor terrain.

It is conceivable that this map could be acquired dynamically by interaction with the environment as has been demonstrated by work with HILARE at LAAS [52] and Neptune at CMU [92]. Environmental acquisition via learning will not be addressed in the near future in our work, although other UMASS researchers may be involved in this research.

The output of the map-builder is utilized in part by the navigator component of the planner process whose duty it is to build a collision-free path through the partially modeled world represented in LTM. An A* search is conducted through the midpoints (A*-1) or triads (A*-3) of the bordering passable convex regions to arrive at a coarse path. The A*-3 method offers a tighter initial path at the expense of a greater search space. In a highly cluttered environment, this can be of value. Generally, the A*-1 search method works almost as well and is less costly to produce.

The resulting raw path is then subjected to path improvement strategies which tighten and straighten the path subject to parameters specified by the mission planner. These path improvement strategies are extremely important for producing quality paths and are an important contribution to the overall path planning process. They allow for the production of short paths, safe paths, or other types of paths, attempting to optimize across a set of paths more freely than other representations might allow. This flexibility offers distinct advantages over a Voronoi diagram or vertex graph approach to world representation. Certainly the regular grid's search space makes it computationally infeasible for all but the simplest domains. The regular grid's susceptibility to digitization bias (quadtrees even more so) also make it a less desirable representation. The flexibility to accommodate diverse new representations, without any changes to the underlying path

planning representation is another of the advantages of our approach. The meadow map, in our estimation, is the best general purpose navigational representation, and as such will, in various forms, continue to be a major force in mobile robot navigation.

Regarding implementation, the navigator, mapbuilder, and LTM representational structure for AuRA are all complete. Knowledge acquisition is ongoing, especially regarding 3D landmark models. There is always more knowledge to be added. The mission planner is currently rudimentary, serving as a command interface. The pilot and STM are not yet fully integrated. For the experimental system in Chapter 8, the schema handoff from the pilot to the schema system requires user intervention. The rulebase has a limited set of rules for schema generation and will be expanded in the near future. The STM instantiator process is complete, as is the first pass version of the STM manager. These exist as separate processes running under the control of the cartographer. The meadow fracturing process will require additional work on the obstacle extraction from STM component for its implementation, but as this is used only in the relatively rare circumstance of schema navigation failure, it is not a top priority.