Path Planning for a Vision-Based Autonomous Robot

Ronald C. Arkin

COINS Technical Report 86-48

October 1986

VISIONS / Laboratory for Perceptual Robotics
Department of Computer and Information Science
Graduate Research Center
University of Massachusetts
Amherst, MA 01003

A brief version of this paper was presented at the
SPIE Cambridge Symposium on Advances in Intelligent Robotics Systems
Conference on Mobile Robots

This research was supported in part by the General Dynamics Corporation under grant DEY-601550 and the U.S. Army under ETL grant DACA76-85-C-008. Publication as a COINS Technical Report was recommended by Prof. Edward Riseman.

Path Planning for a Vision-Based Autonomous Robot

Ronald C. Arkin

COINS Technical Report 86-48

Computer and Information Science Department
University of Massachusetts
Amherst, Massachusetts 01003

Abstract

The VISIONS research environment at the University of Massachusetts provides an integrated system for the interpretation of visual data. To provide a testbed for many of the algorithms developed within this framework, a mobile robot has been acquired. The test domains of interaction are twofold: the interior of a large research center building (rooms, halls, foyers, doorways, etc.) and the immediate outdoor area surrounding the building (sidewalks, gravel paths, grass patches, building entrances, etc.).

A multi-level representation and the accompanying architecture used to support multi-sensor navigation (predominantly visual) are described. A hybrid vertex-graph free-space representation based upon the decomposition of free space into convex regions capable for use in both indoor and limited outdoor navigation is discussed. This "meadow map" is produced via the recursive decomposition of the initial bounding area of traversability and its associated modelled obstacles. Of particular interest is the capability to handle multiple terrain types (sidewalks, grass, gravel, etc.). "Transition zones" ease the passage of the robot from one terrain type to another.

A hierarchical path planner, (mission planner, navigator, and pilot), that utilizes the data available in the above representational scheme is described. An A* search algorithm, present in the navigator, incorporates appropriate cost functions for diverse terrain navigation. Consideration is given to just what constitutes an "optimal" path in this context.

This research was supported in part by the General Dynamics Corporation under grant DEY-601550 and the U.S. Army under ETL grant DACA76-85-C-008.

I. Introduction

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.

HARV (fig. 1) is a mobile robot equipped with video and ultrasonic sensors. It is to be operated in two distinctly different domains:

- The confines of a research building; including halls, foyers, and large rooms
- The grounds of the UMASS campus; including sidewalks, building entrances, parking lots, and grassy areas.

Thus, any architecture and representation scheme to be used must be sufficiently general to handle both the indoor and outdoor case.

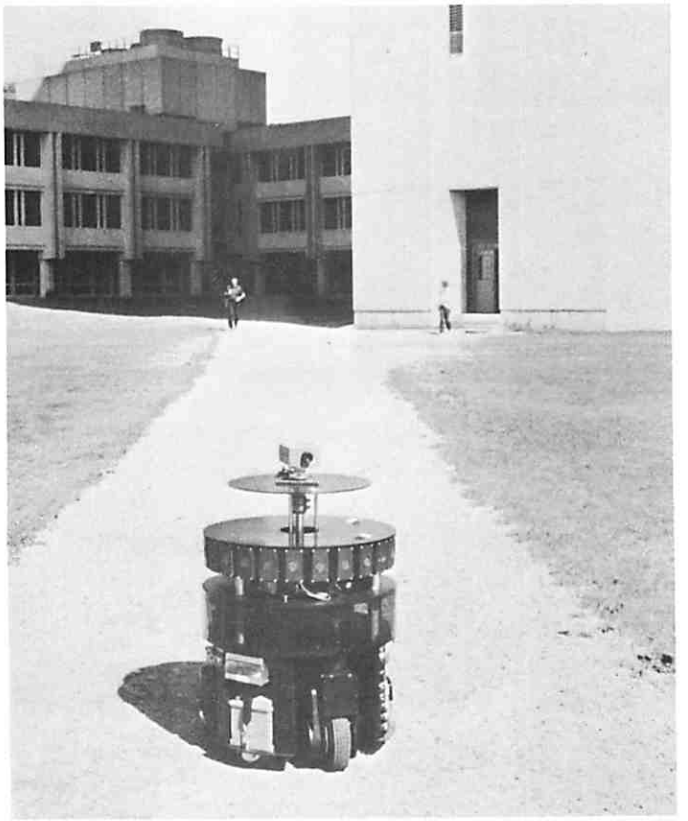
This paper is concerned primarily with path construction and navigation in a partially modelled environment. The UMASS autonomous robot architecture (AuRA) incorporates a hierarchical planner consisting of a pilot, navigator, mission planner and motor schema manager (the execution arm of the pilot).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

In the remainder of this introduction, relevant work will be cited followed by a description of the UMASS environment. Section II will describe the representation used by the navigator and the software that builds this map. The operation of the navigator will be described in Section III. The extension of the representation to include diverse terrain types will be related in section IV. A brief overview of the entire system architecture relating the position of the navigator and the representations described herein will appear in section V. A summary and conclusions will complete this report.

I.I Previous work

Early in the days of artificial intelligence, Amarel showed that a good representation is essential for the efficient solution of a problem [1]. Many different tacks have been taken by various researchers regarding appropriate navigational representations. These include pure free-space representations [12,16,18], vertex graphs, potential fields [15], regular grids [19,21,27,32] quadtree [2] and automaton representations [31]. A hybrid representation involving both vertex graph and free-space

(a)



(b)

Figure 1. HARV
(a) HARV inside the Graduate Research Center
(b) HARV outside the GRC.

representation techniques has been used by [7,20,33]. Each approach has its own strength and weakness. (A comparative survey appears in [5]). In several instances [13,30] multi-level representations are used to exploit the best of several different representation forms.

The UMASS VISIONS [11,26,36] system, a system used for the interpretation of natural scenes, also encompasses a multi-level representation scheme. (See fig. 2). Knowledge sources provide hypotheses and instantiated schemas based on top-down knowledge and video sensor input. This system will ultimately be the gateway for visual and other sensor data entering into AuRA. 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 present. The system proposed in section V 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 [34], hierarchical [13,14,20,25,28] and distributed [9,30]. Unfortunately, in developing hierarchical planners, such as the one used here (fig. 3), no clear guidelines are present demarcating which responsibility 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 [24] incorporates much of what would be considered a navigator in other systems). A navigator for the purposes of this paper 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 and turn right 85 degrees. 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 specified by the navigator can be made without reinvocation of the navigator as long as these alterations fall within these limits. The pilot does not utilize the same representation the navigator does, as it assumes that the navigator has correctly produced a path that avoids **any modelled** obstacle. Consequently the pilot is concerned solely with avoiding unmodelled obstacles (subject to certain constraints). Other work [13,14,24] describes the use of similar hierarchical planning systems.

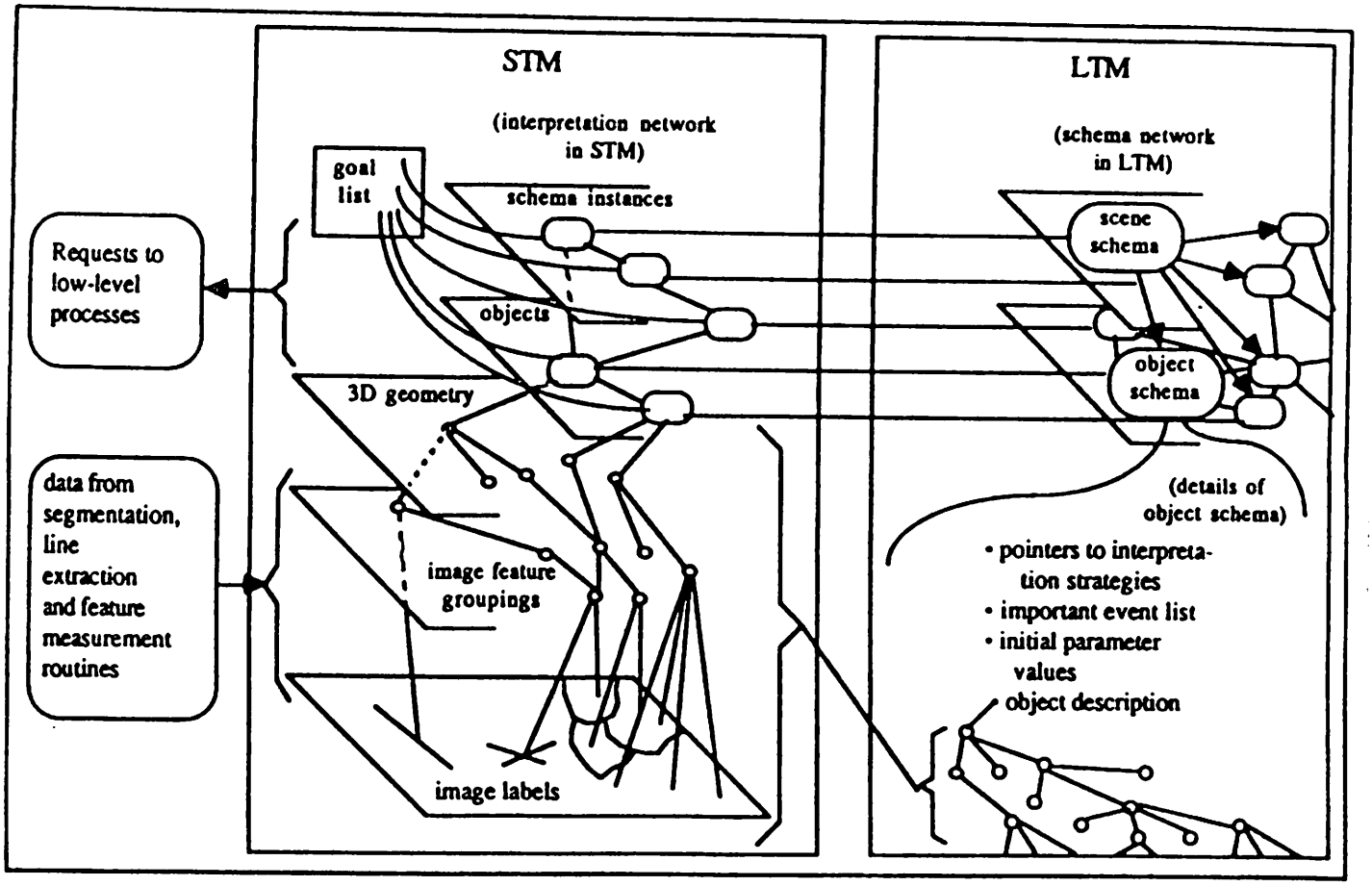


Figure 2.
Representations of Short and Long-Term Memory in VISIONS
Representations ranging from schemas to low-level visual data are organized in a multi-level network. The reader is referred to [36] for additional information (this diagram is reprinted from [36]).

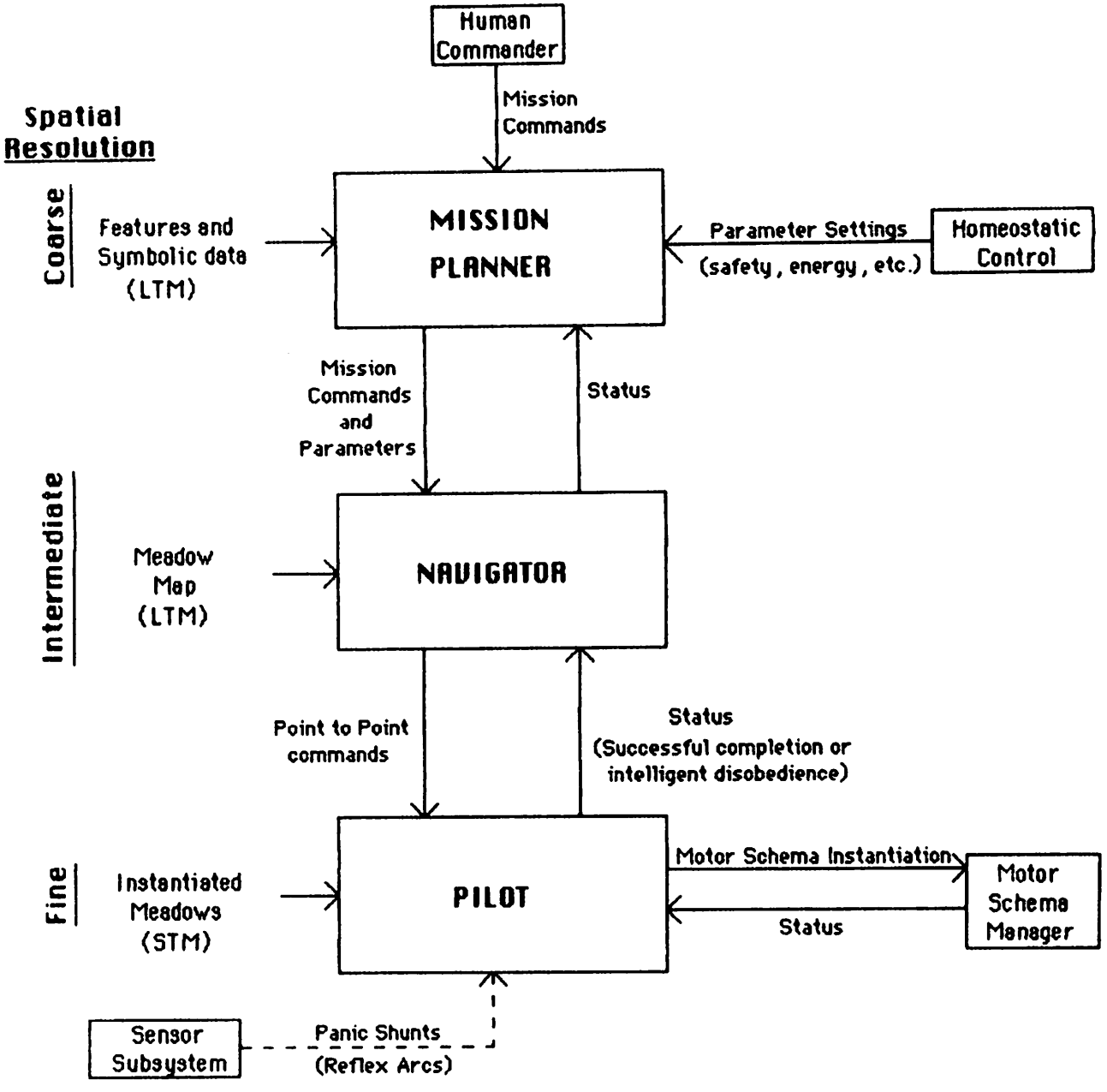


Figure 3. Hierarchical Planner for UMASS AuRA

This three level planner is a component of the overall AuRA system architecture (see figures 31-32)

Section V discusses how the AuRA pilot will be schema based; choosing motor actions from a library of appropriate behaviors (schemas) that are relevant to satisfying the navigator's subgoals. The scope of this paper, however, is primarily limited to navigational planning issues and the schema based pilot and its representations will be described in a forthcoming paper.

1.2 UMASS environment

HARV (fig. 1) is a mobile robot manufactured by Denning Mobile Robotics. It is equipped with 24 ultrasonic sensors and shaft encoders for both the steering and drive motors. A single video camera (the VISIONS system uses monocular images) is mounted on the vehicle and connected to a Gould digitizer. Although some of the sensor preprocessing (ultrasonic and encoder) is done on board the vehicle utilizing its two Z-80's and M68000, the bulk of the software runs on the Computer and Information Science (COINS) departments VAXen. At the time of this writing, approximately 12 concurrent processes of the system (fig. 32) are operational. It is anticipated that this heavy load will eventually be distributed over the NSF CER blackboard-based multiprocessor. These heavy computational demands must be satisfied in order to obtain real-time performance. The development of the UMASS Image Understanding Architecture and the CAAPP (Content Addressable Array Parallel Processor [35]) is expected to enhance the vision processing performance when it becomes available. In addition, the robot is to be equipped with a UHF TV transmitter enabling remote operation.

The two arenas in which the robot is to be operated include both indoor and outdoor environs. The first is within the confines of our building, the Lederle Graduate Research Center (GRC). The navigator assumes significant but incomplete a priori knowledge of the world. Blueprints for the building (fig. 4) constitute the basis on which the initial indoor representation is built. A digitizer is used to enter the relevant map features. 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 [10] and Neptune at CMU [22]. 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 second arena is the grounds surrounding the GRC. The model is derived largely from a map made via aerial photography (fig. 5a). Figures 5b-c show photos of the area from the robot's perspective. 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

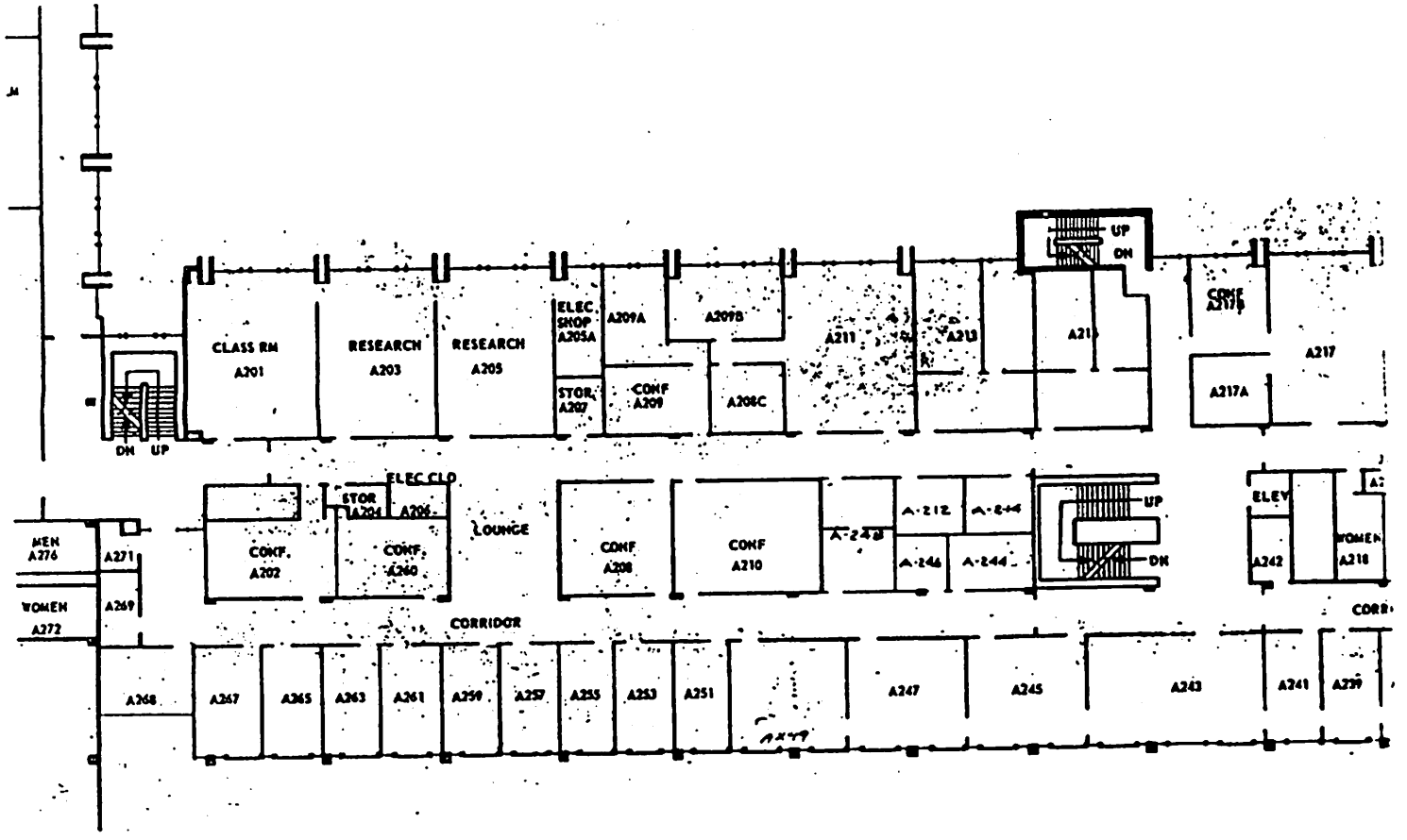


Figure 4. Indoor Environment Model

This diagram, a partial blueprint for the GRC, represents the starting point for building the long term memory model of the robot's world (indoor case).

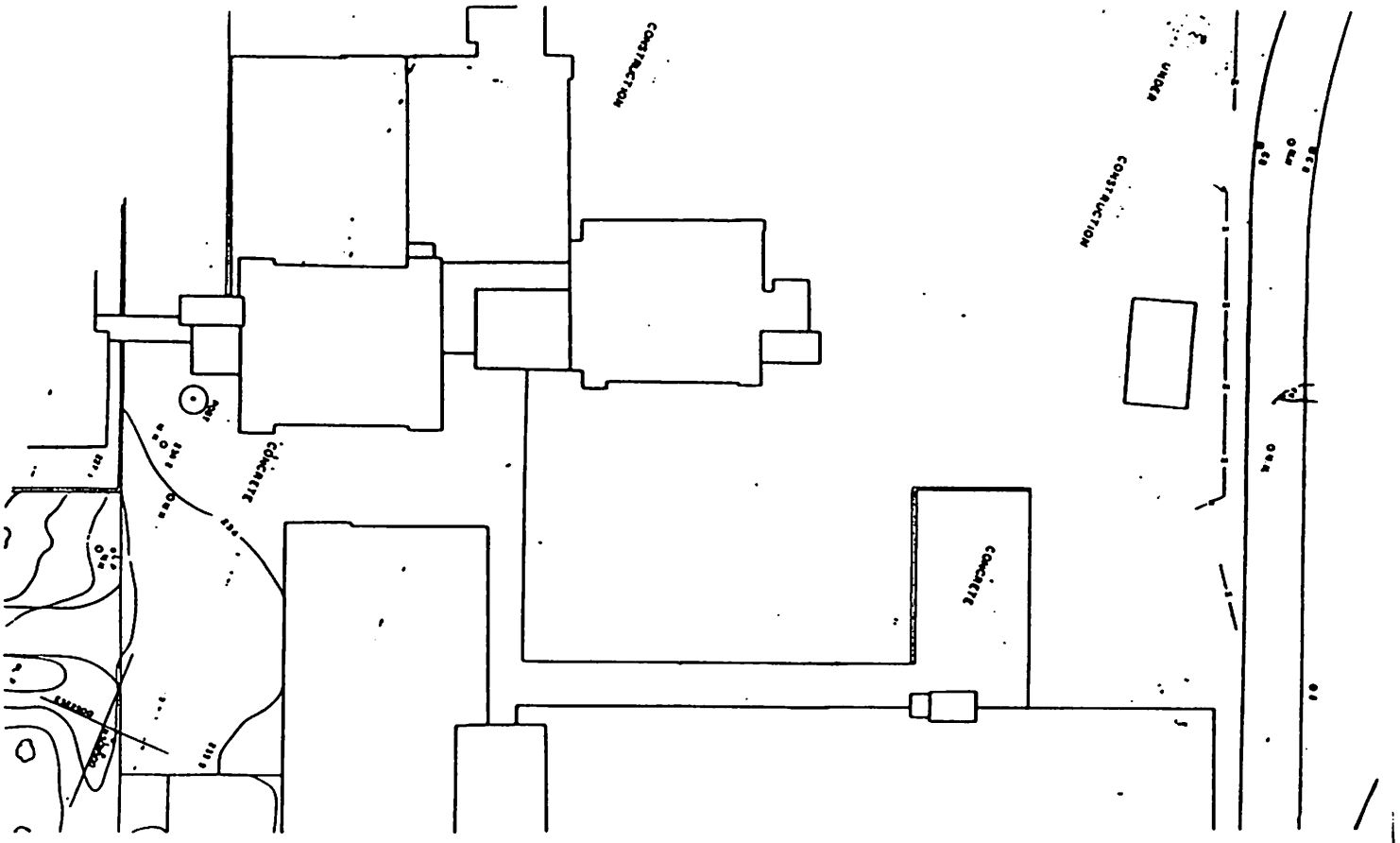
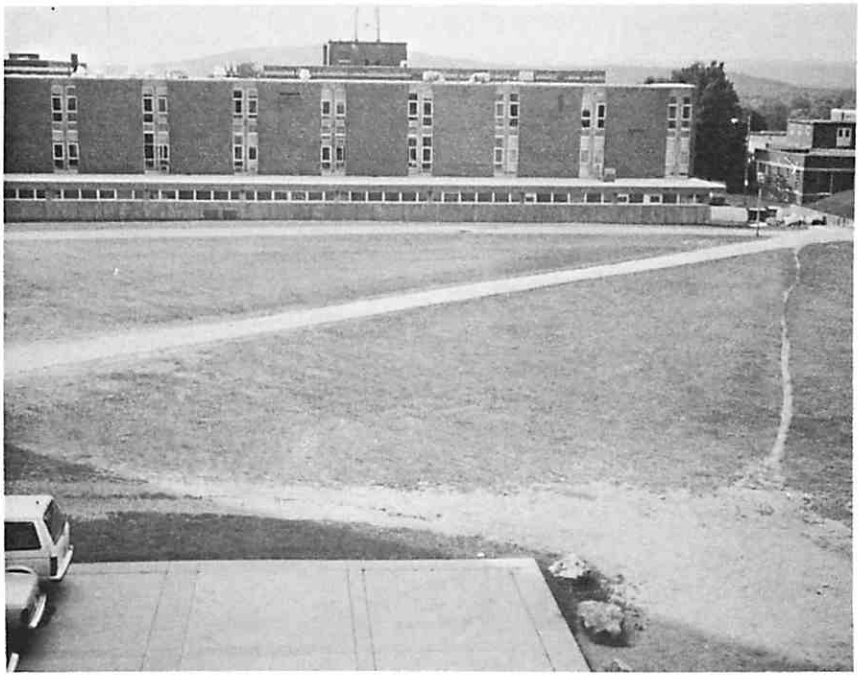
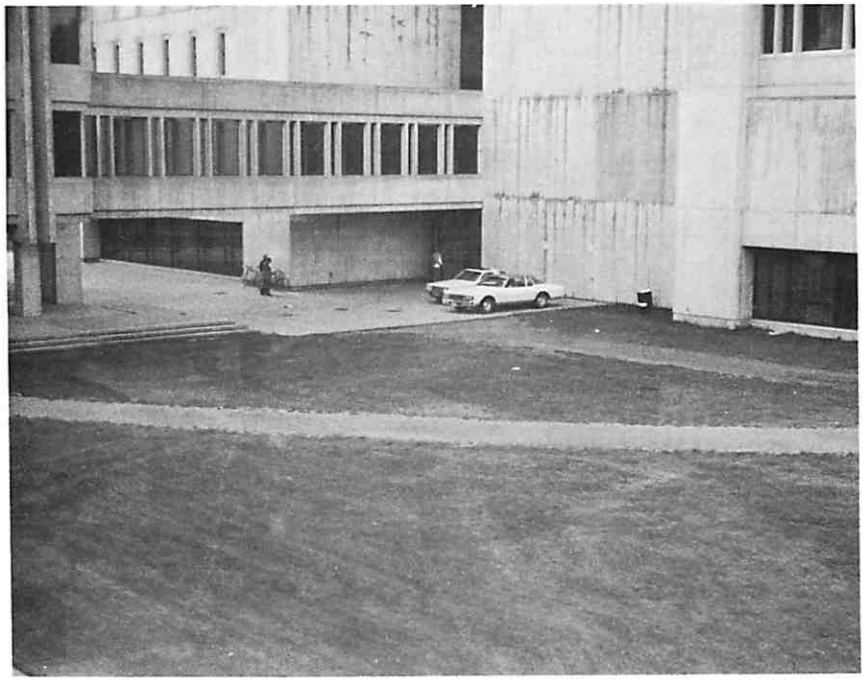


Figure 5. Outdoor Environmental Model

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



(b)



(c)

Figure 5. (continued)
b & c) Two outdoor photos of the area depicted in figure 5a.

precludes the use of surface models, (e.g. planar patches), to represent topographic features within the free space regions.

II. Representation

To address the issues of path planning, a static representation and a dynamic representation have been developed (the reader is referred to [36] 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 - 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 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. Although, as stated earlier, the current system does not include learning, it would not be difficult to develop an additional process that would take what is learned about the world from the short-term memory and incorporate that data into long-term memory where appropriate. Further discussion of the details of short-term memory for navigation will be deferred to a separate paper.

II.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 [8] and Chatila [7]. It models free space as a collection of convex polygons. Diagrams depicting an indoor scene and an outdoor scene appear in figures 13 and 27 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 III.2 and IV.2).

What distinguishes this representational form from the efforts that preceded it lies in its ability to embed both terrain and sensor data and its extension to include diverse terrain types (see section IV). 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 this paper.

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

FREE SPACE MAP BUILDING ALGORITHM

Initialization

Accept Bounding Region (border)
Shrink bounding region (a la configuration space)
Accept modelled obstacles
Grow 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
        done
else
        find (most,least,first) concave angle
        connect it to (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 6.

II.1.1 Meadow Map Construction

The algorithm for the construction of the long term memory meadow map representation is described in figure 6 and consists of the following phases: initialization, main map building and clean-up.

II.1.1.a Initialization

In the initial 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 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 - to follow) 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 in the configuration space manner as developed by Lozano-Perez [17]. (fig. 7) This enables the robot to be treated as a point thereafter for pathplanning 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 of 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. 8.a). Two alternatives are available to produce the required linear segments. First the resulting side line segments could be extended until they meet (fig. 8.b). 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. 8.c). 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 with the segments produced in the first method. (fig. 8.c and 8.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 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 concave angles to prevent passage occlusion, whereas relatively clear areas would prefer the straightforward extension method.

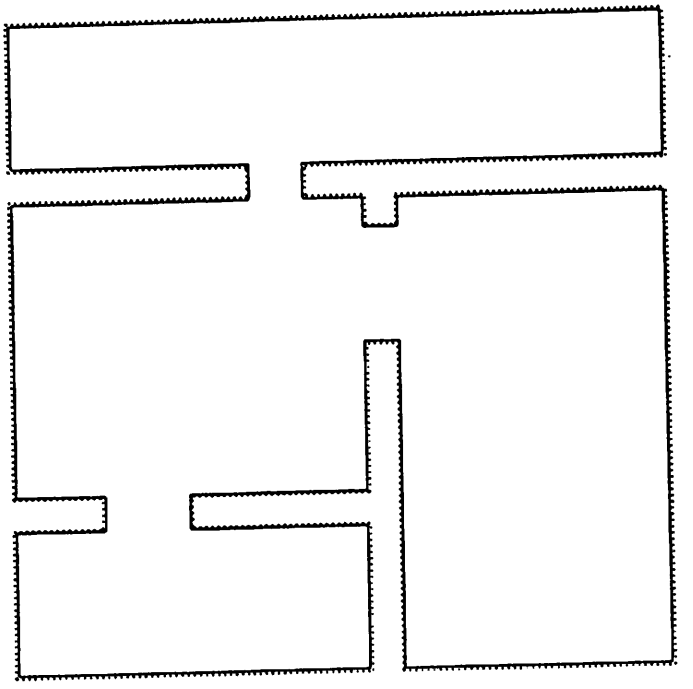


Figure 7. Grown Border

A test indoor case, based on an example from [9], shows the original ungrown border, (dotted line), and the resulting shrunken region (solid line). (Approximate scale 400' by 400', robot radius 18 inches, c-space safety margin 6 inches).

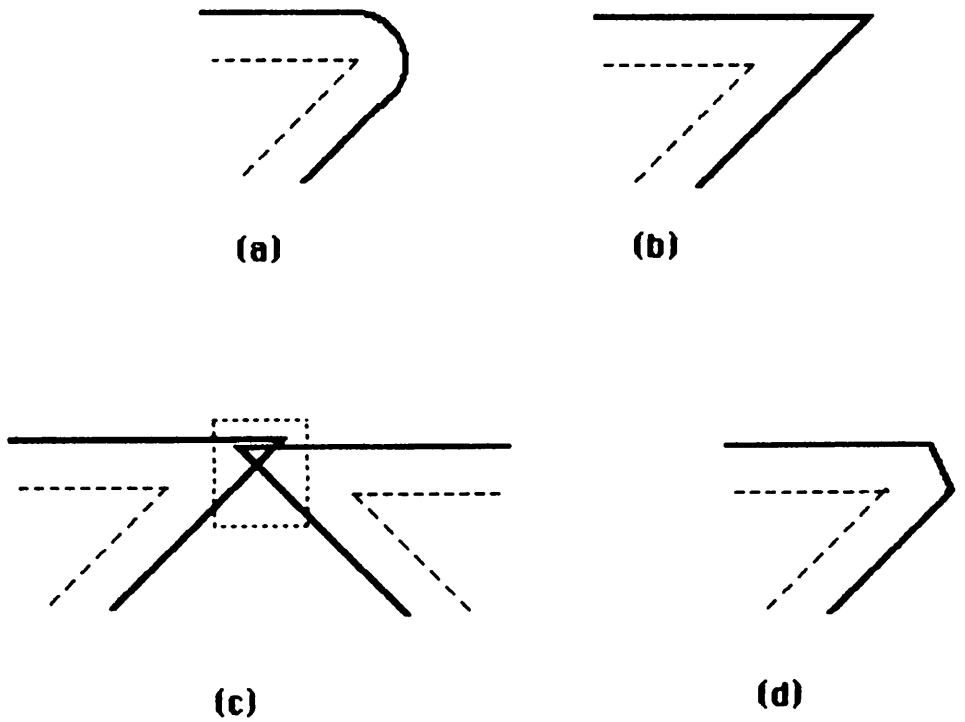


Figure 8. Chopping Concave Angles

------- Actual obstacle

——— Grown obstacle

..... Blocked by C-space but should be clear area

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

Known obstacles that are present in the environment (pillars, telephone poles, ... 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 bounding region was shrunk.(fig. 9). 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, these 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. 10). To the mathematical purist, this would not be a region as the two lines connecting the obstacle to the border are in identical positions. Nonetheless, this simplifies the recursive decomposition algorithm which appears in the step following.

II.1.1.b 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 are available for selection: the least concave, the most concave, or the first concave angle found can be chosen (fig. 11). 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. Three possibilities again exist (fig. 12): 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. 13a). Thus a graph of convex regions and their traversability is produced, facilitating search during the path finding process. This decomposition continues until all of the regions produced by

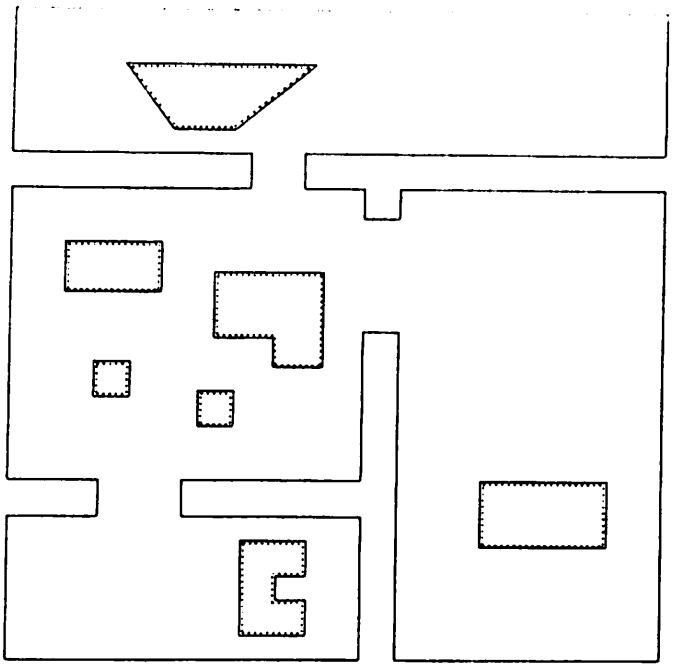


Figure 9. Obstacles

The obstacles have been added to the case shown in figure 7. 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.

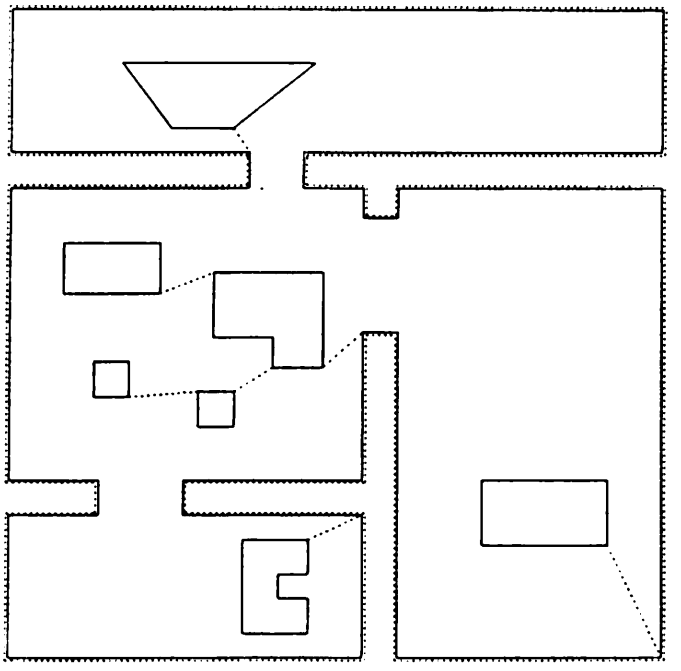


Figure 10. Attached Obstacles

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

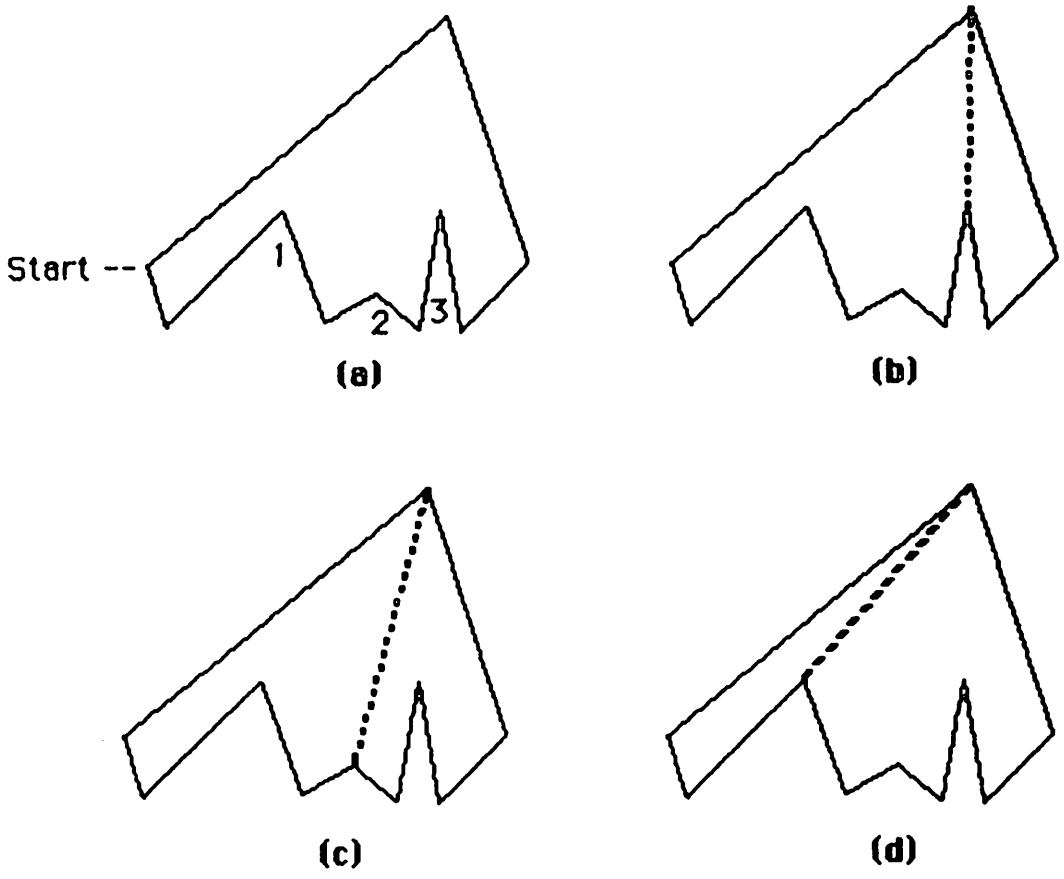


Figure 11. Effect of Vertex Selection on Decomposition

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

b) Most convex vertex selected

c) Least convex vertex selected

d) First convex vertex selected

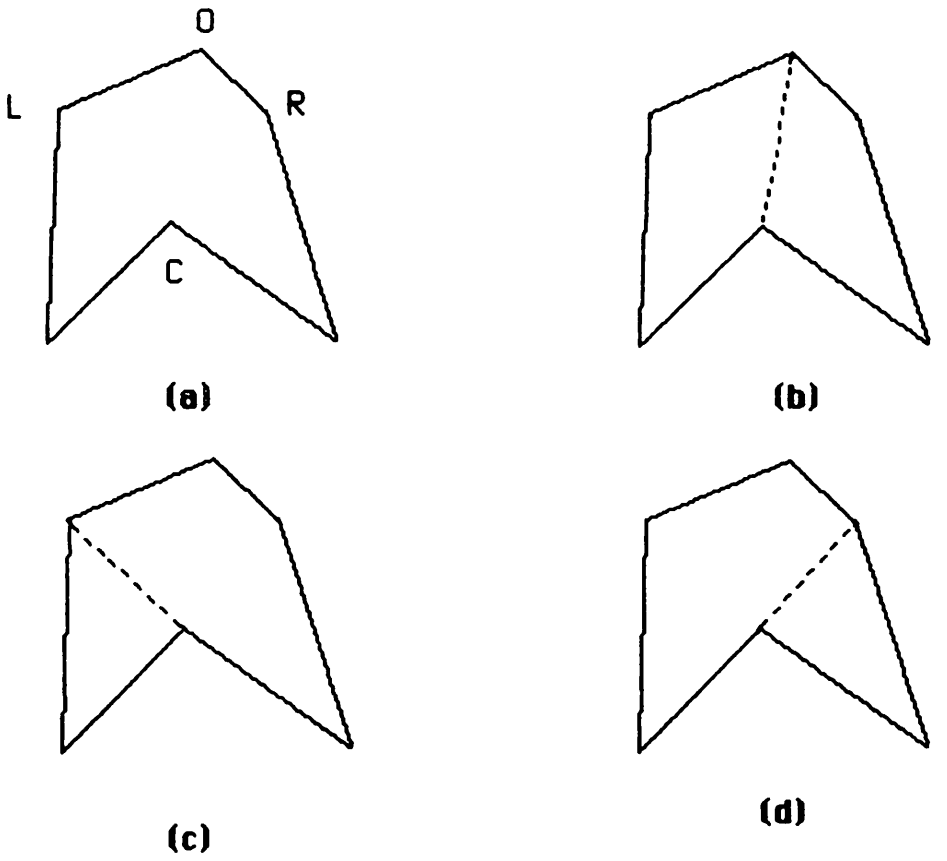


Figure 12. The effect of Victim vertex Selection on Decomposition

a) This figure contains one convex 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.

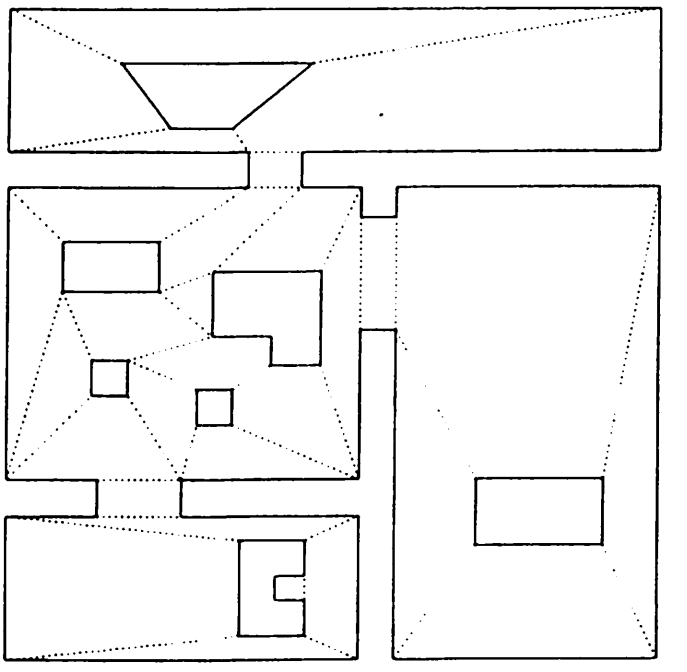
this algorithm are convex.

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. Figures 13a-e show 5 different decompositons on the same region. Just how to define what constitutes the best results is nebulous at best. 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 modelled obstacles during travel due to the inherent positional uncertainty found in the mobile robotics domain. Fewest overall legs in a particular path might is another possible choice. In one instance [Crowley - 8] 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.

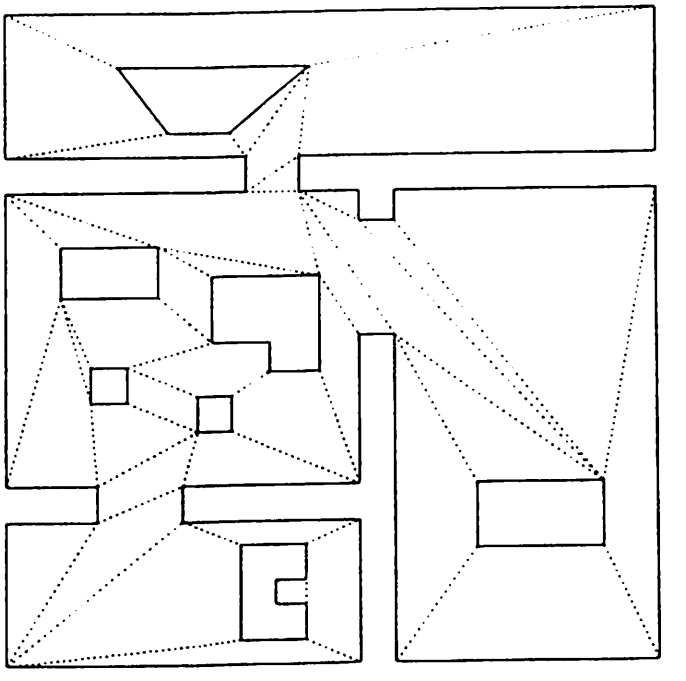
When the path search was restricted to the midpoints of the bounding regions, (A^*-1, see section III.1), the experimentation indicated, even based on the shortest distance metric, that the results obtained were more strongly influenced by the shape of the initial bounding region and the choice of start 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 (fig. 19 and 20); 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 mapbuilding 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). The efficiency of each mode and their impact on the path planning computation, and data regarding mapbuilding times appears in the appendix.

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



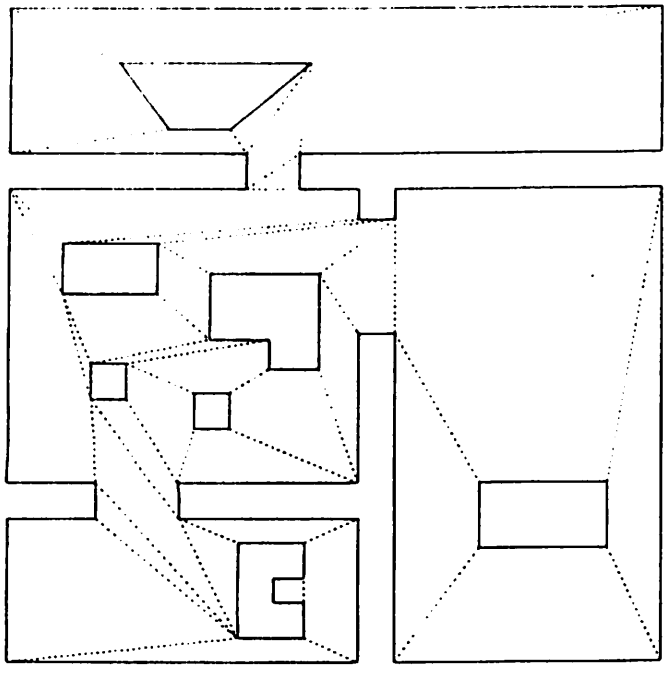
(a)



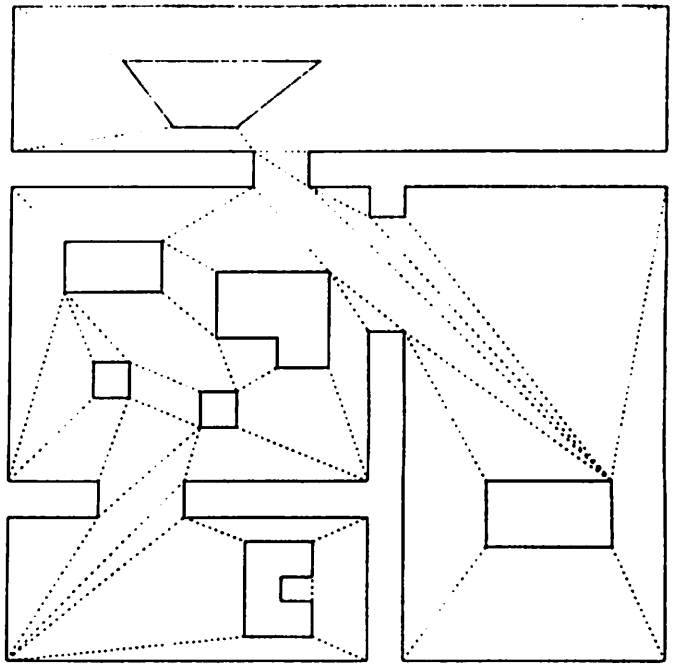
(b)

Figure 13. Different Convex Decompositions

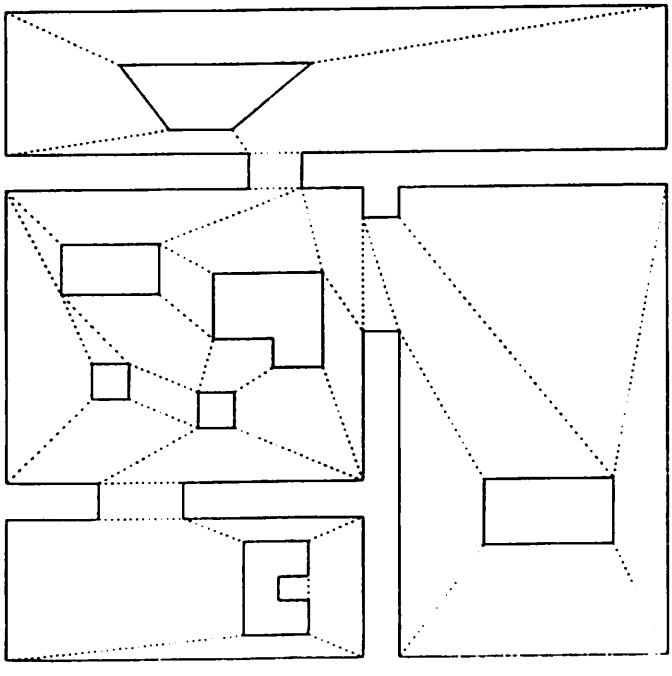
Figures 13a-e show 5 of the nine different decompositions available for the region shown in figure 10. Solid lines represent impassable obstacles and borders, dotted lines - passable boundaries between meadows. The selection modes are listed below.
(a) most concave angle, most opposite victim; (b) most concave angle, leftmost victim; (c) most concave angle, rightmost victim; (d) least concave angle, most opposite victim; (e) first concave angle, most opposite victim.



(c)



(d)



(e)

Figure 13. Different Convex Decompositions (continued)

maintained by pointer links in the passable edges connecting a region to its adjacent passable regions.

II.1.1.c 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. The mapbuilder 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 IV.

II.1.2 Feature editor

A major advantage that this representation affords is the ease in which additional representations of objects, landmarks, terrain features, etc. can be embedded. In an experimental system this is very important. The level of granularity can vary; for obstacles or walls, full 3d models of the entire object, 2D planar representations of surfaces (pop-up views) associated with sides (edges), or even simplistic line models for individual corners, projected up into the image plane, can be readily embedded via pointers. 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 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. The new representation is accepted by the editor, storage 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.

This data 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 should be 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

Some of the initial features our system will include (but is not limited to):

- **Terrain data**
  - **Traversability factor (ease of passage)**
  - **Translational and rotational error data - (to guide the spatial error map for managing positional uncertainty)**
  - **Data to guide labeling of regions obtained from visual region segmentation**
  - **Unmodelled obstacle density**
- **Obstacle data**
  - **2D pop-up views of sides (vertical image plane projections embedded in the ground plane)**
  - **Vertical edge data for particular vertices (sides of buildings, doorways, etc.)**

In the last stage of the mapbuilder, 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.

III. Navigation (Global Path Planning)

After the mapbuilder 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 reconaissance 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.

The navigator accepts a start and a goal point from the mission planner and, using the global map built by the mapbuilder, determines the "best" path to attain that goal. Just what constitutes such 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 in the hierarchical planner. As this paper is about navigational planning the remainder of this section will deal with how the navigator fulfills its responsibilities. But first, a brief discussion of the operation of the pilot.

The pilot is charged with implementing leg-by-leg the piecewise linear path that is output by the navigator. To do so the pilot chooses from a repertoire of available sensing strategies and motor behaviors (schemas) and passes them to the motor schema manager for invocation. As the robot proceeds, the sensors build up a model of the perceived world in short-term memory which is used when the pilot recognizes that it cannot satisfy the navigator's goals without making changes in the specified path. If the changes required are too severe, (as specified by the higher levels of the planner), the navigator will be reinvoked and a new global path computed. If the deviations are within acceptable limits, the pilot and motor schema manager will, in a coordinated effort, attempt to bypass the obstacle or other problems. Additionally, the problem of robot localization is constantly addressed through the monitoring of short-term memory and appropriate **find landmark** schemas. Multiple concurrent behaviors may be present during any leg, for example:

Stay on Path (a sidewalk)
Avoid static obstacles (parked cars, etc.)
Avoid moving obstacles (people etc.)
Find intersection (to determine end of path)
Find landmark(building) (for localization)

The control of the priorities of the behaviors, (e.g. when is it more important to follow the sidewalk than to avoid uncertain but possible obstacles) is dependent on the certainty associated with the STM representation and controlled by the motor schema manager. A sensor-independent short-term memory representation will be the fusion point for all the different sensor modalities and strategies. A forthcoming paper will discuss both the role of the pilot and short-term memory in implementing the navigator's plans. For now, back to the navigator and LTM.

The remainder of this section will deal 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.

III.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 [27] 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.

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 acheived? 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 I 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 modelled world). Since the robot's environment is subject to unmodelled 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. Reasonableness seems an acceptable criteria.

Just how do we gauge "reasonableness"? Only the successful completion of the robot's experiments can be the judge. Until the overall system is completed, however, subjective evaluation will remain the principal method. 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 travelling 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.

After stating that, let me contradict myself a bit by saying that the A* algorithm [23] is used with heuristics (at this point) that guarantee optimality. 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 [8]). 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. 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 than that of a regular grid or pure vertex graph representation. Finding the initial coarse path is a fairly rapid operation (see appendix). 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 admissability, the heuristic function $h^$ of A* must never overestimate the cost remaining to the goal. The easiest $h^$ heuristic function to guarantee an underestimate (or the exact cost) is the shortest straight-line Euclidean distance on the plane assuming the best terrain. Since the search space is relatively small, no effort has been placed 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 (up to 5 times as large - see appendix).

Path Finding Algorithm

Accept start and goal from mission planner.
Check for validity.

Search

Apply A* search algorithm through convex region connectors.
(A* - 1: midpoint only)
(A* - 3: midpoint + two points near the two endpoints)
Output Raw Path

Path Improvement Techniques

If desired
(based on given path optimality criteria - safest, shortest, etc):
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. 15)
Return "reasonable" piecewise linear path through world model.

Figure 14.

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 unmodelled obstacle density (perhaps a function of time of day - e.g. high between classes on a sidewalk, low otherwise), and ease of localization (based on numbers of readily discernible landmarks within a given region). These are expected to be incorporated into the algorithm once they are quantified.

In summary, the navigator algorithm (shown in fig. 14) accepts first 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 (fig. 17a). 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 safer paths from shorter paths (as the free space is not directly represented), hence the strategy used here will be

seen to be considerably more versatile. 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).

III.2 Path Improvement Strategy

Path improvement techniques are relatively common in the use of regular grids. Although the representation used for the UMASS AuRA 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 [32] while Mitchell and Keirsey use a compensation technique [19] to minimize inefficiency due to digitization bias.

The algorithm for path improvement for the simple single terrain type case is presented in figure 15. A graphic illustration of the process appears in fig. 16. 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 IV). 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. 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 the path.

Path Improvement Algorithm
(single terrain type case)

Accept the coarse path from the search component of the navigator

Tautness Component

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 Component

Get start of path
Do while not at end
    If clear path is available to any path node ahead of the current node from the current node
        delete all intervening nodes
    Endif
    Get next path node
End do

Clean up

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

Figure 15.

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.

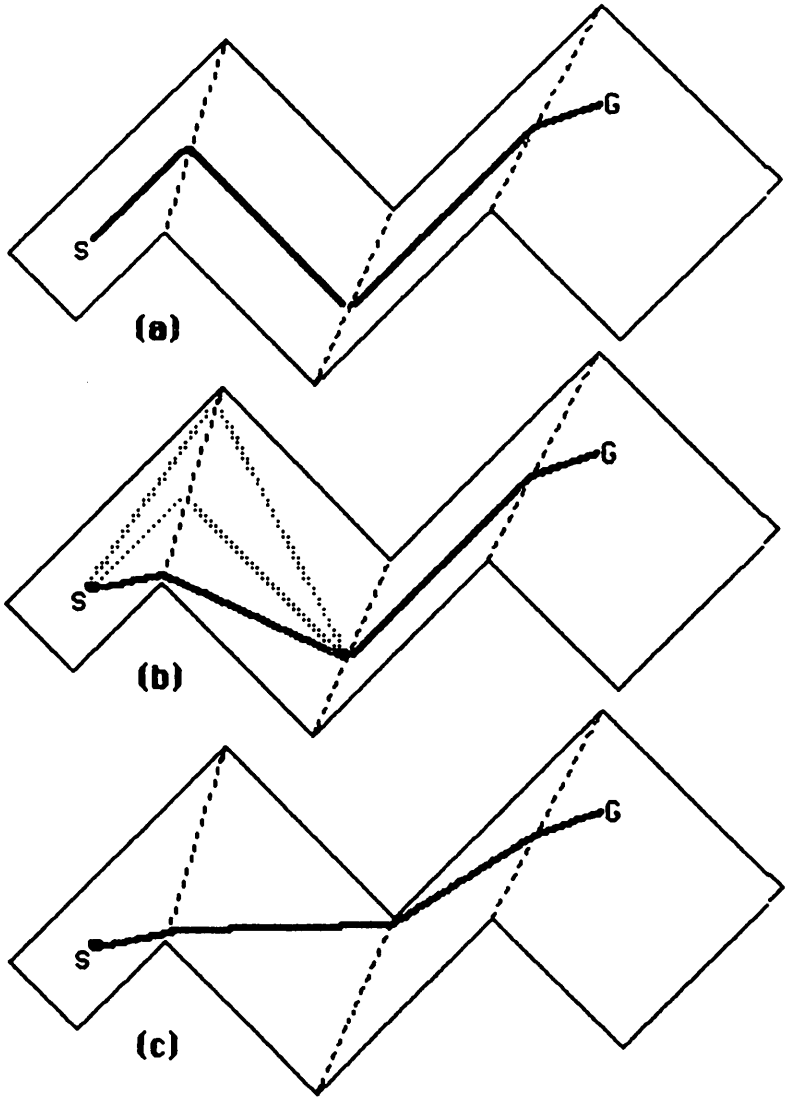


Figure 16. Path Improvement - Tautness Component

a) The initial raw path passed from the A*-1 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.

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.

III.3 Results

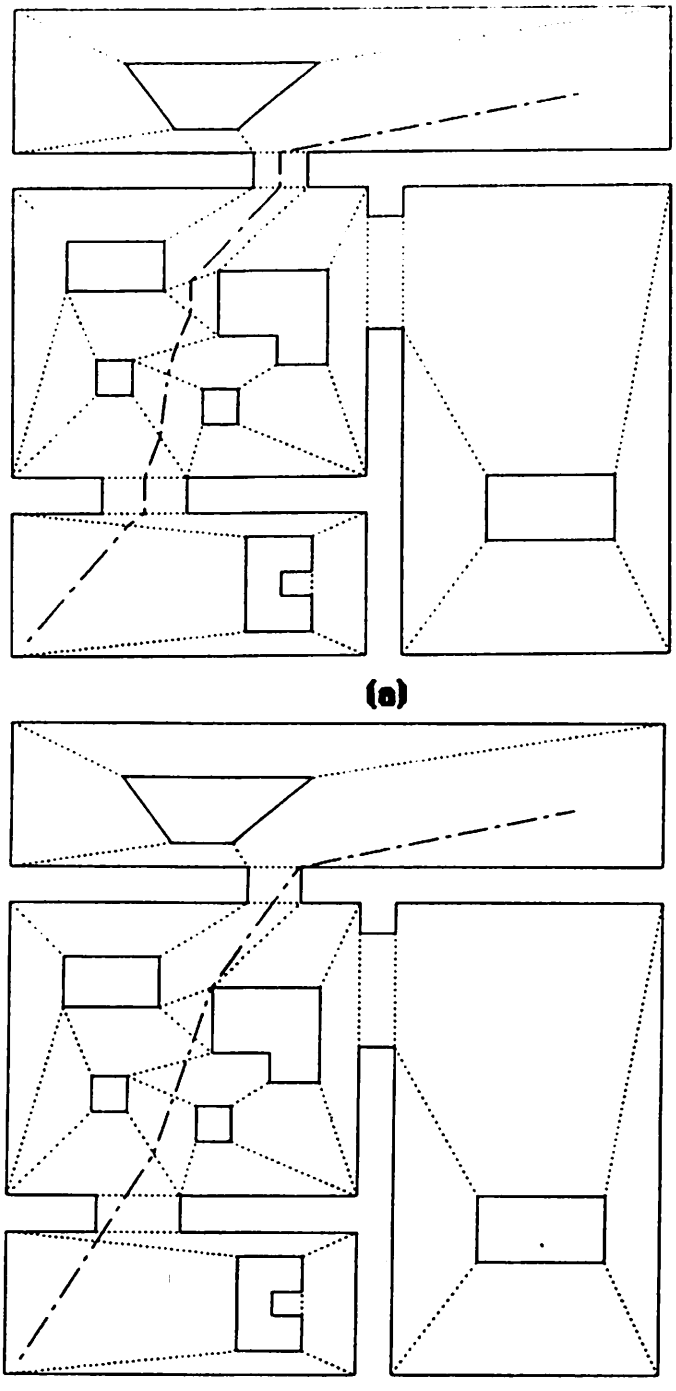
The results are presented in figures 17-23. The straightening component of the algorithm can be observed to remove unnecessary detours around obstacles, while the tightening component reduces the overall path cost. The cost function used is the same cost function used in the search algorithm (for these figures, shortest Euclidean distance is the cost).

A significant advantage to 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 17-20), 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 mapbuilding. Considerable experimentation was conducted trying to determine which if any of the nine modes available to the mapbuilder 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. 19 & 20). 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 possible start and end points within any given map, no single map building strategy was clearly superior.

For A*-3 noticeable improvement occurred. Figure 21 clearly shows the ability to produce a better path than was the case with the A*-1 method (fig. 19). Figure 23 confirms these results. The computational penalty however can be significant as the search space is considerably larger and is discussed in the appendix.

Hopefully the diagrams (fig. 17-23) 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 - fig. 23), one of the best will be chosen. No claims for overall optimality are made, only reasonableness. Restating that optimality is perhaps a misplaced notion



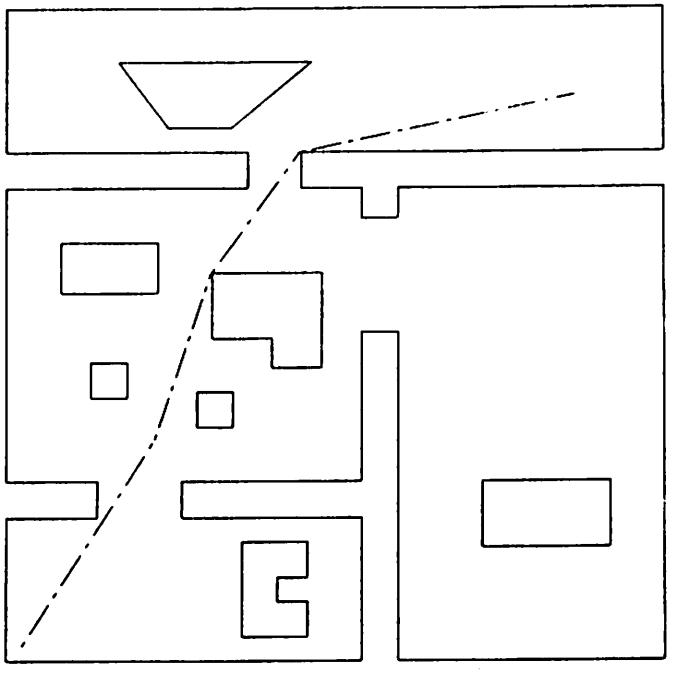
(a)

(b)

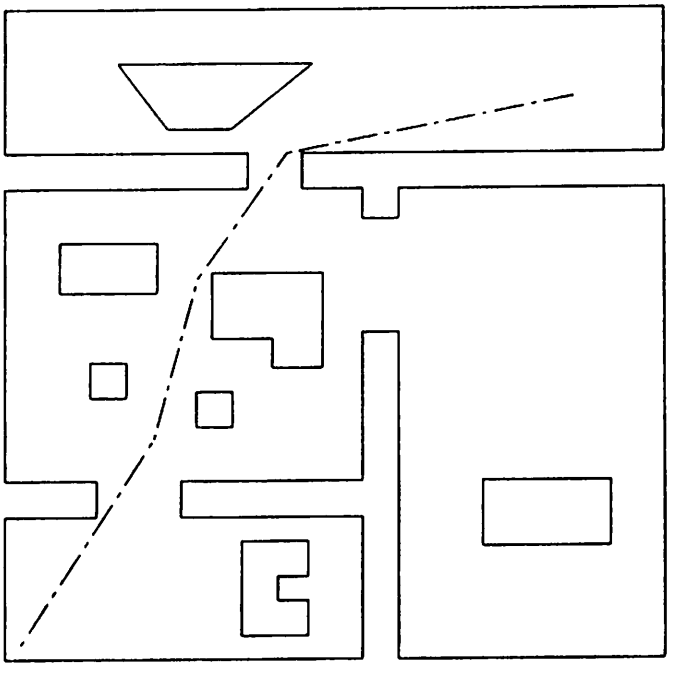
Figure 17. Single Terrain Path Planning Example (A*-1)

This sequence illustrates the path finding process of the navigator for a single terrain type. The convex decomposition of figure 13a is used for figures 17-19. 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 borders 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 and is not a measure of path distance from the vertex.



(c)



(d)

Figure 17. Single Terrain Path Planning Example (cont.) (A*-1)

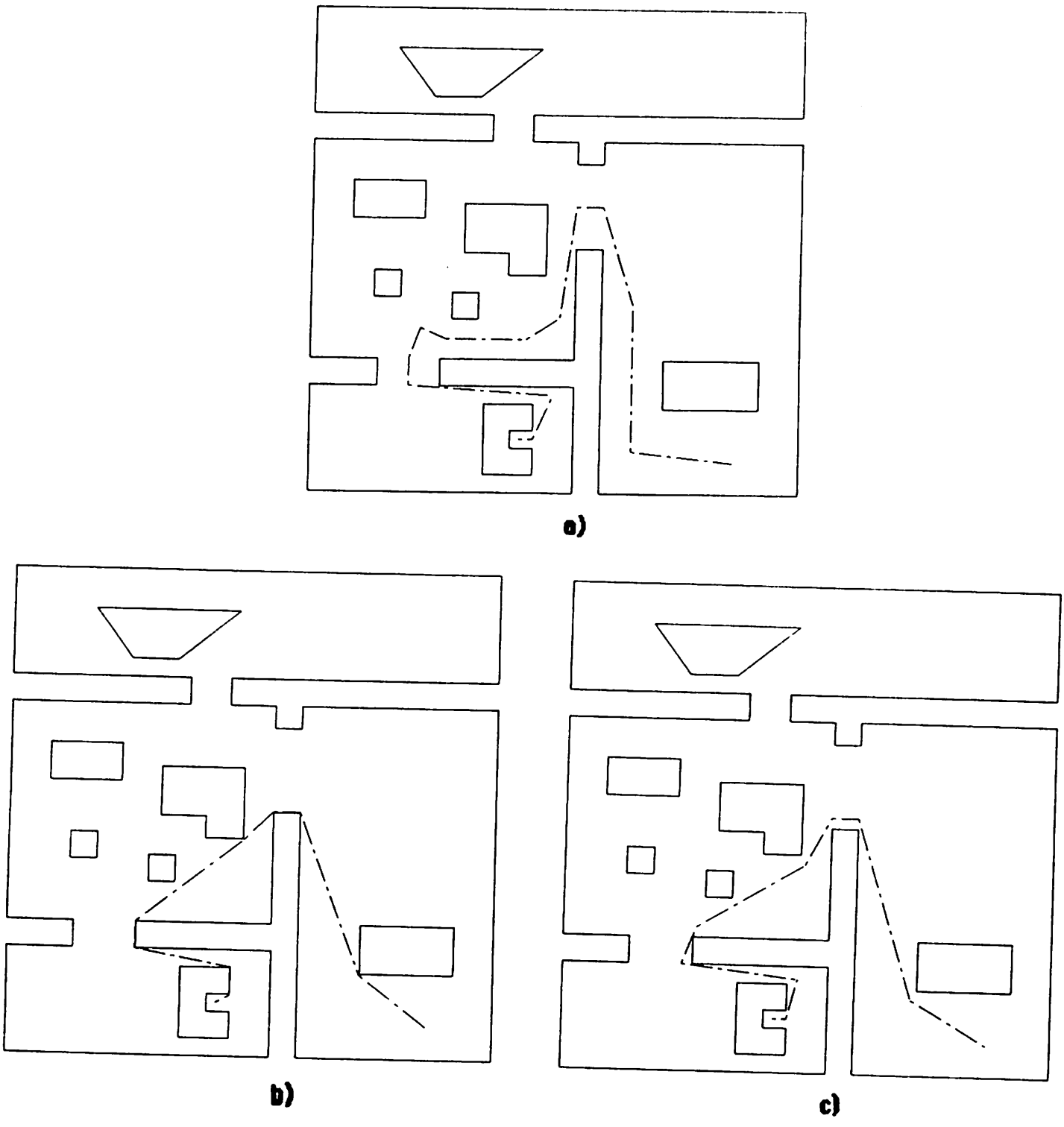
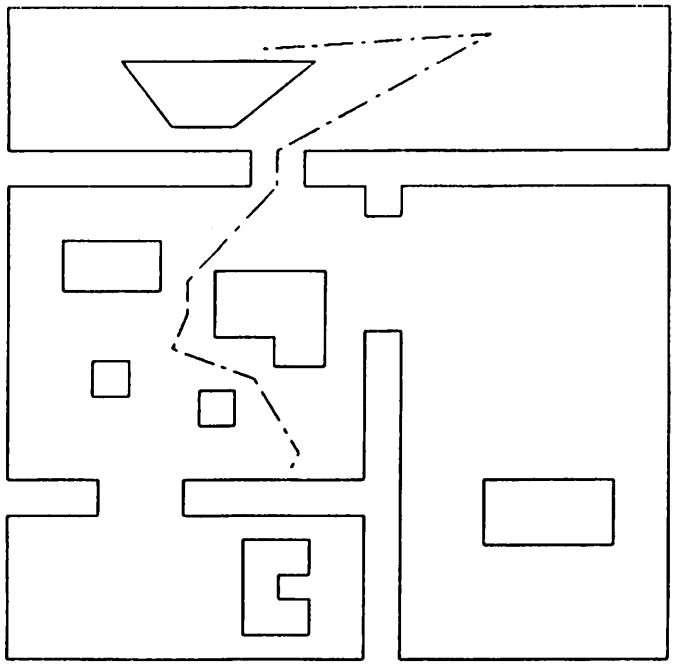
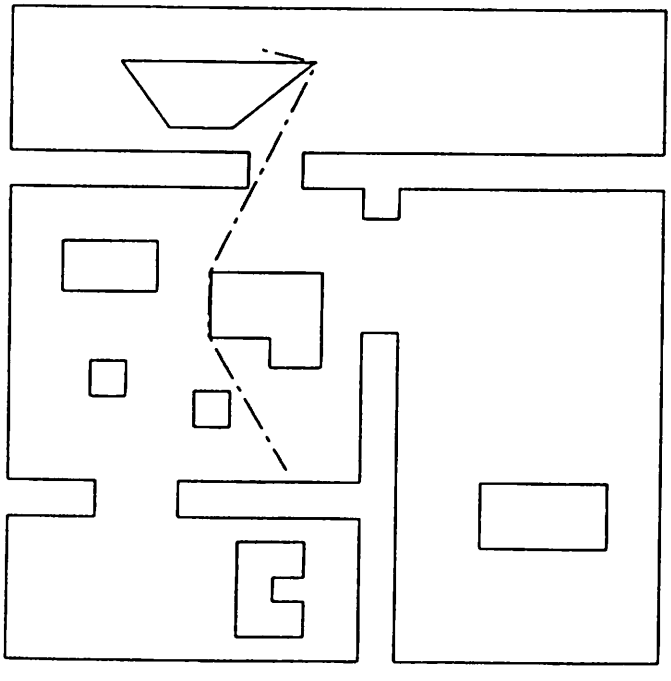


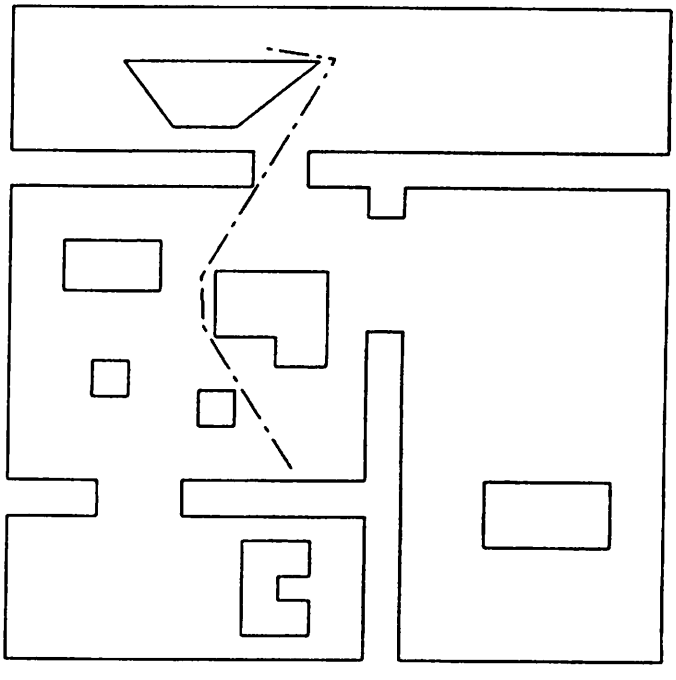
Figure 18. Another Single Terrain Planning Example (A*-1)
(a) Initial coarse path
(b) Improved path (safety margin 1 foot)
(c) Improved path (safety margin 10 feet) (see note for fig. 17d)



(a)



(b)



(c)

Figure 19. 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. 17d)

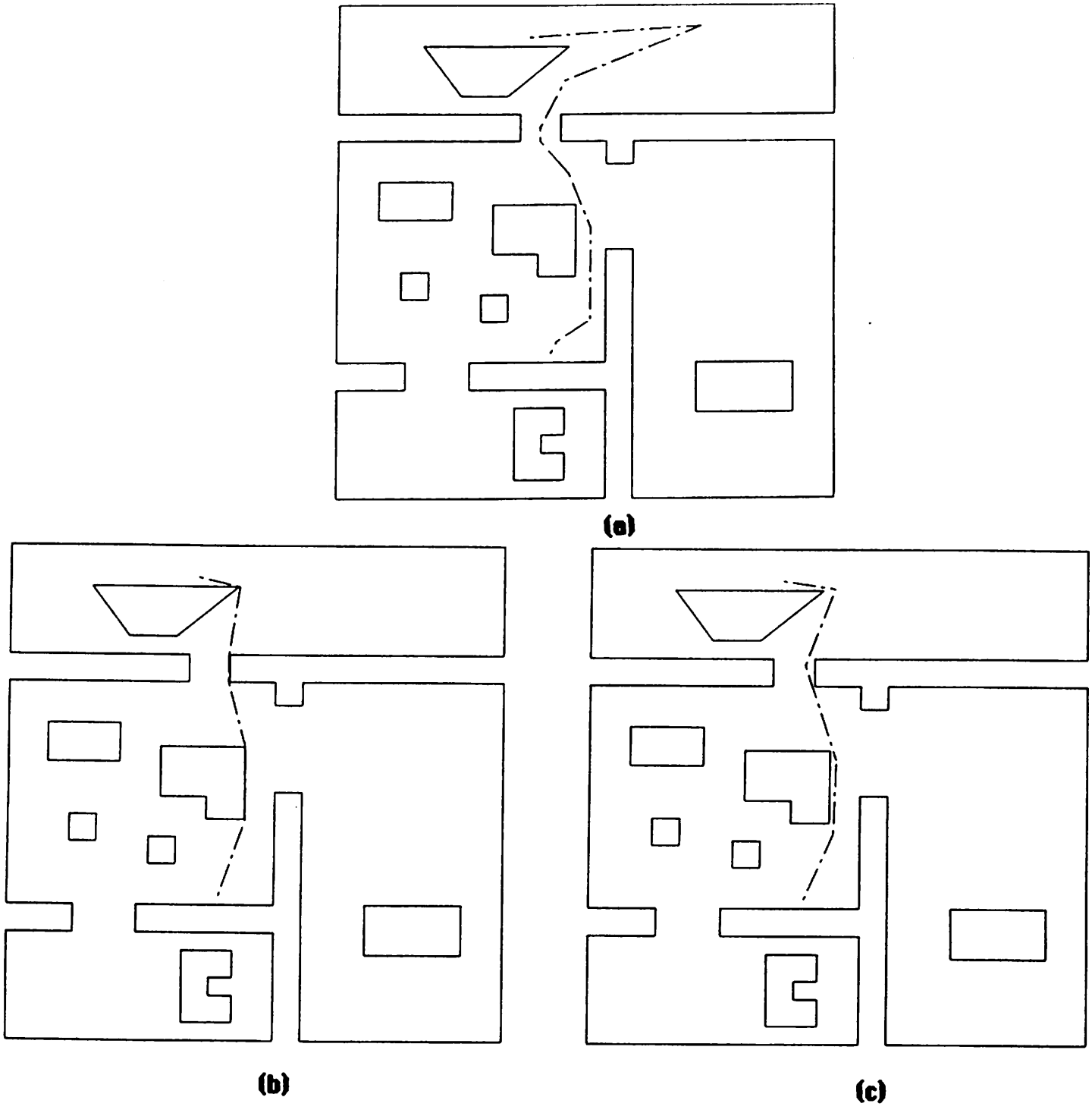
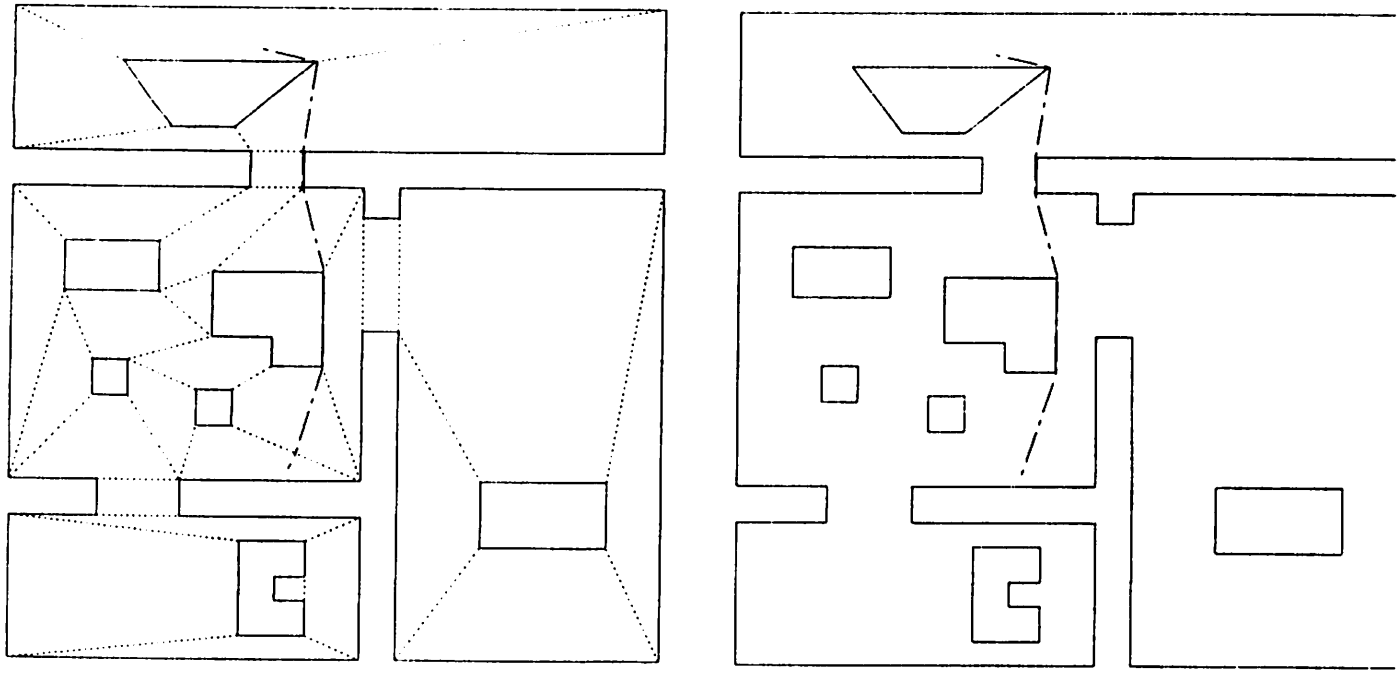


Figure 20. Dependency on Decomposition Method (A*-1)

In this case, the decomposition method of figure 13b (most concave vertex, leftmost victim) was used, not that of fig. 13a (most concave vertex, most opposite victim) as in the previous cases. Although the start-goal points and path improvement techniques are identical with the previous figure (fig. 19), 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)



(a) (b)

Figure 21. A*-3 Path Planning

Contrasting this figure against figure 19, (which uses the same decomposition method as is used here), A*-3 search provides the same lower cost path as was seen in figure 20 (fig. 20 used a different decomposition strategy). 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. Actually identical to raw path (a) in this case.

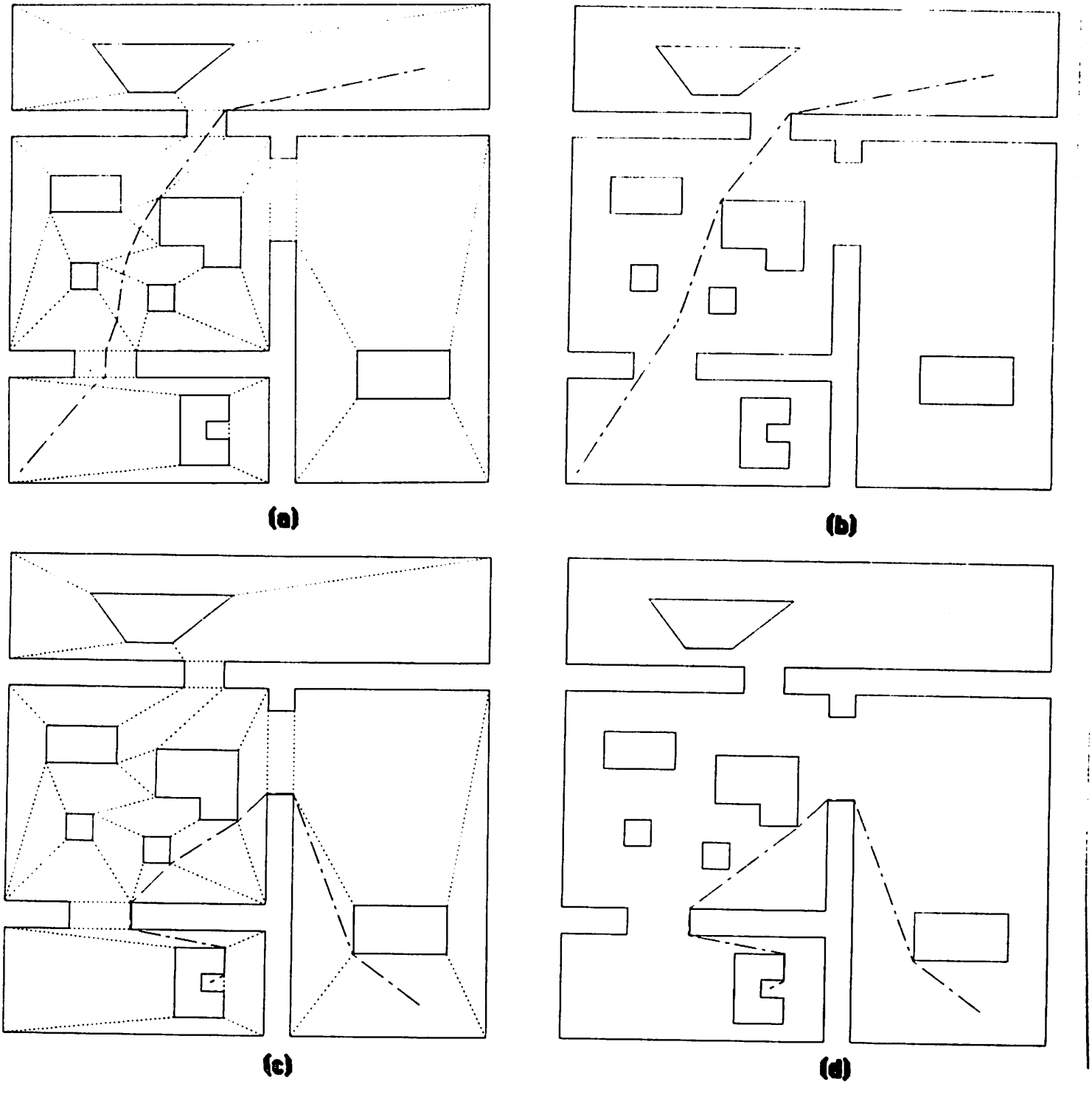


Figure 22. More A*-3 Path Planning

Here it can be seen that the A*-3 method gives no improvement over the A*-1 method for the cases in figures 17 and 18. The extra computational cost proves unnecessary.

(a) Same path as in figure 17 but using A*-3. 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 figure 18 but using A*-3. Initial raw path.

(d) Final improved path for (c).

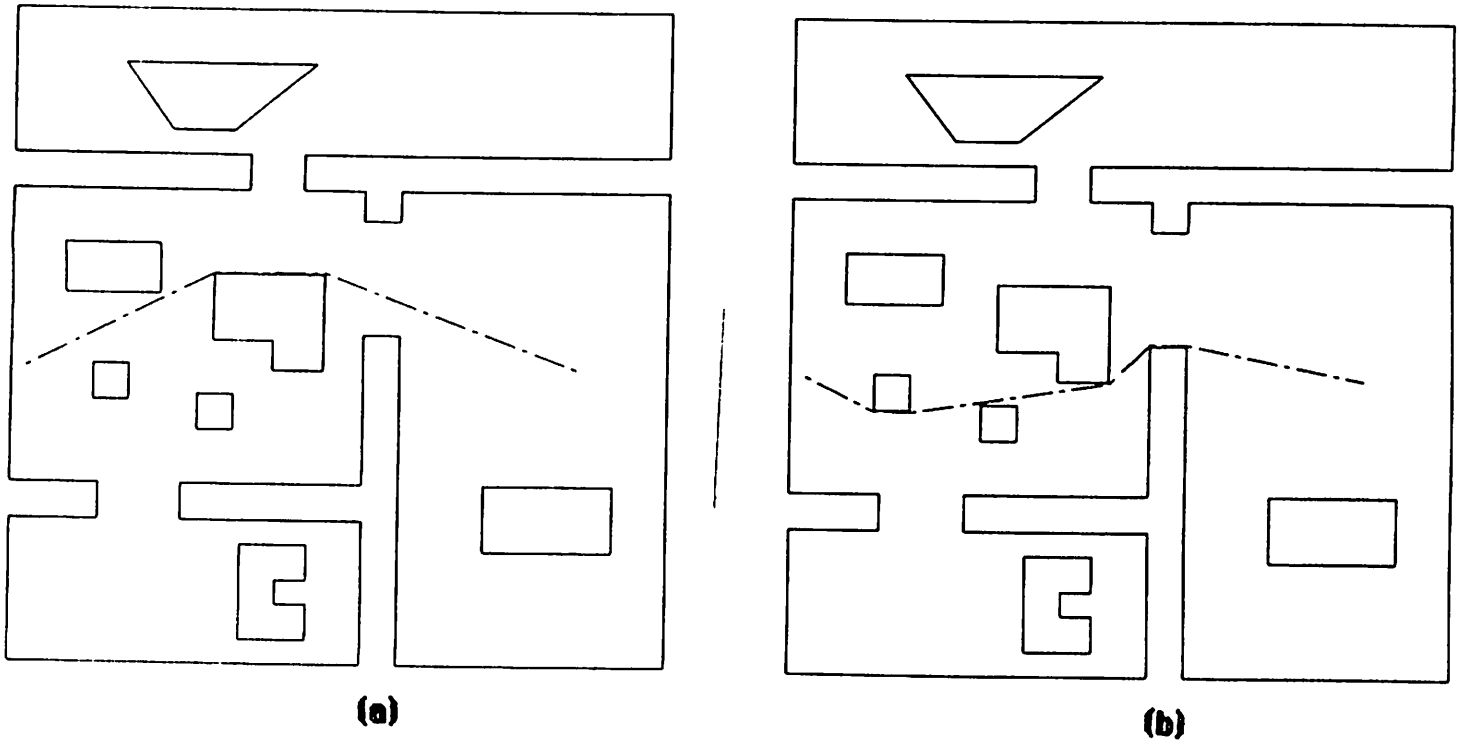


Figure 23. A*-1 versus A*-3

In tight quarters A*-3 can make a slight difference. The gain, however, is small, on the order of 2%.

(a) A*-1 final improved path.

(b) A*-3 final improved path.

in a dynamically changing world (without constant replanning) the value of spending high computational effort in ensuring absolute minimal costs in this mobile robot's domain is unjustified.

The limitations for other representation forms show the meadow map approach in a positive light. 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 discussion on feature editor - section II.1.2). If the paths produced are sub-optimal in the global context and only reasonable, that is a small price to pay for the versatility and lower memory costs afforded by this representational strategy.

IV. Multi-Terrain Extensions

One of the principal advances of this work lies in its extension to handle diverse terrain types. Previously the regular grid has been the principal approach used to deal with diverse ground covers [19]. 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 travelling 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 will be to prevent the robot from straddling terrain borders. This will be 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 expected to be very problematic in terms of maintaining proper localization.

This section will first describe how the mapbuilder accommodates multiple terrain types through the construction of transition zones and their appropriate features. The next section will describe how the navigator has been modified from the uni-terrain model described to accommodate path-planning through this extended representation.

III.1 Multi-terrain mapbuilder

The extended mapbuilder is built from the uni-terrain mapbuilder described in section 11.1.1. The algorithm appears in figure 24.

Multi-Terrain Mapbuilder Algorithm

```
DO WHILE (no more terrain to add)
   Run the Uni-terrain map builder (fig. 6) on a terrain region
   Tag all resulting free space regions with a new terrain identifier
   Match borders of new free space regions against
          old 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 24.

The input structure of a terrain region is identical to that of the previous mapbuilding algorithm: a list of border and obstacle vertices. This region is decomposed in exactly the same manner as was done previously. Nothing labelling 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 labelled 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 as such 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 mapbuilder algorithm keeps accepting new ones until none remain. After each terrain area is decomposed in isolation, a matching algorithm is run on the new terrain convex regions 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. 25a). 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 (see fig. 25b). 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 zone, implying a normal straightline path. Consequently the initial polygonal representation is converted into a rectangle (fig. 25c). The new edges produced (sides of the rectangle) are labelled 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 II.1.2).

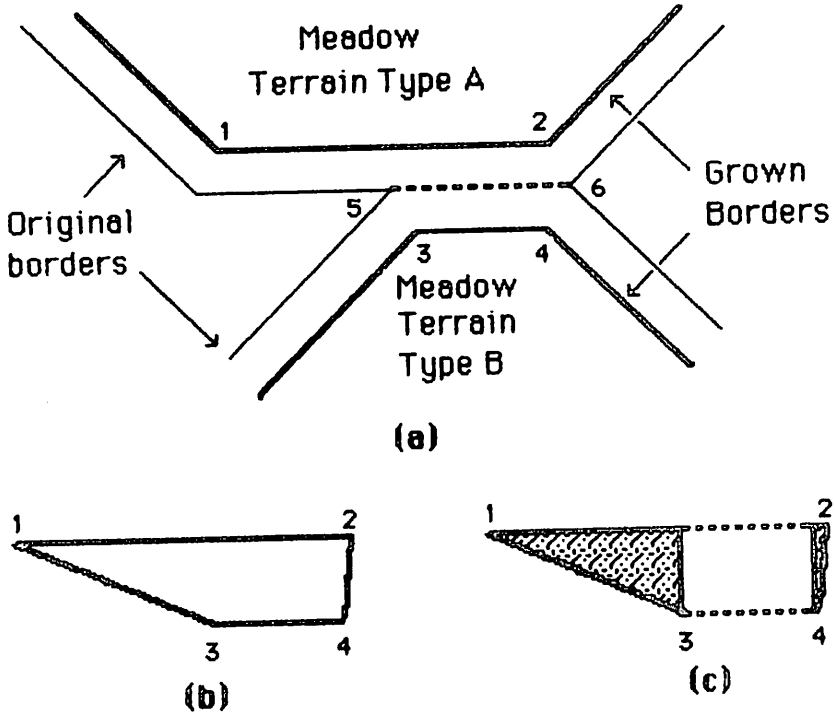


Figure 25. Transition Zone Construction

(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.

IV.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 component of the navigator that must be changed are the path improvement strategies. This includes both the straightening and tautness component. 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 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. This 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.

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. 15)
Slide only the transition zones only 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 26.

The algorithm for multi-terrain path improvement is shown in figure 26. Actually, the complexity is somewhat greater due to special case treatment (start or end within transition zone, entire path in transition zone, etc.). Sedgewick [29] states that "special cases .... are the bane of geometric algorithms", and I am in firm agreement with him.

Reviewing the algorithm: 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 III.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. This step generally reduces the overall distance the transition zone crossings will have to be moved during the relaxation phase, thus reducing computation time. Both crossing points on the transition zones are slid in tandem, insuring a perpendicular passage across the transition region. 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) 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). Best case (no transition crossings) is virtually identical to that of the uni-terrain results (section III.3), and average case results in slight increases in path improvement time. Finally the path is attempted to be restraightened within the context of each terrain type before final release to the pilot.

IV.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 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 27.

In figures 28 through 30, the results of the path planning algorithm are illustrated. The A*-1 search method was used for all these cases. Sub-figures 28a-30a 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 28-30b show the improved path before transition zone relaxation. Sub-figures c display the final path after relaxation and post-relaxation straightening.

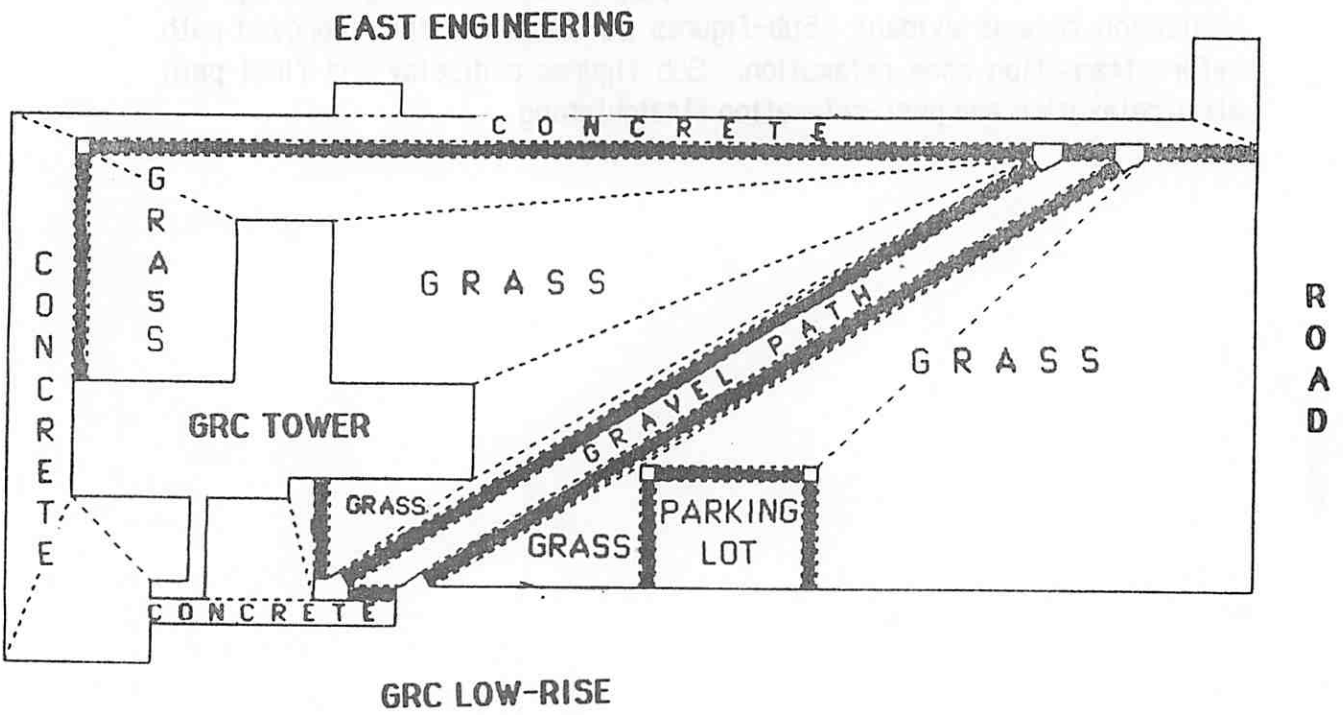


Figure 27. Multi-Terrain Map

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 shaded rectangles. The different terrains, grass, concrete, parking lot and gravel path, are labelled. (Scale approximately 320' by 180' - much smaller than actually the case, but necessary to clearly see the transition zone-path relationship in the figures to follow).

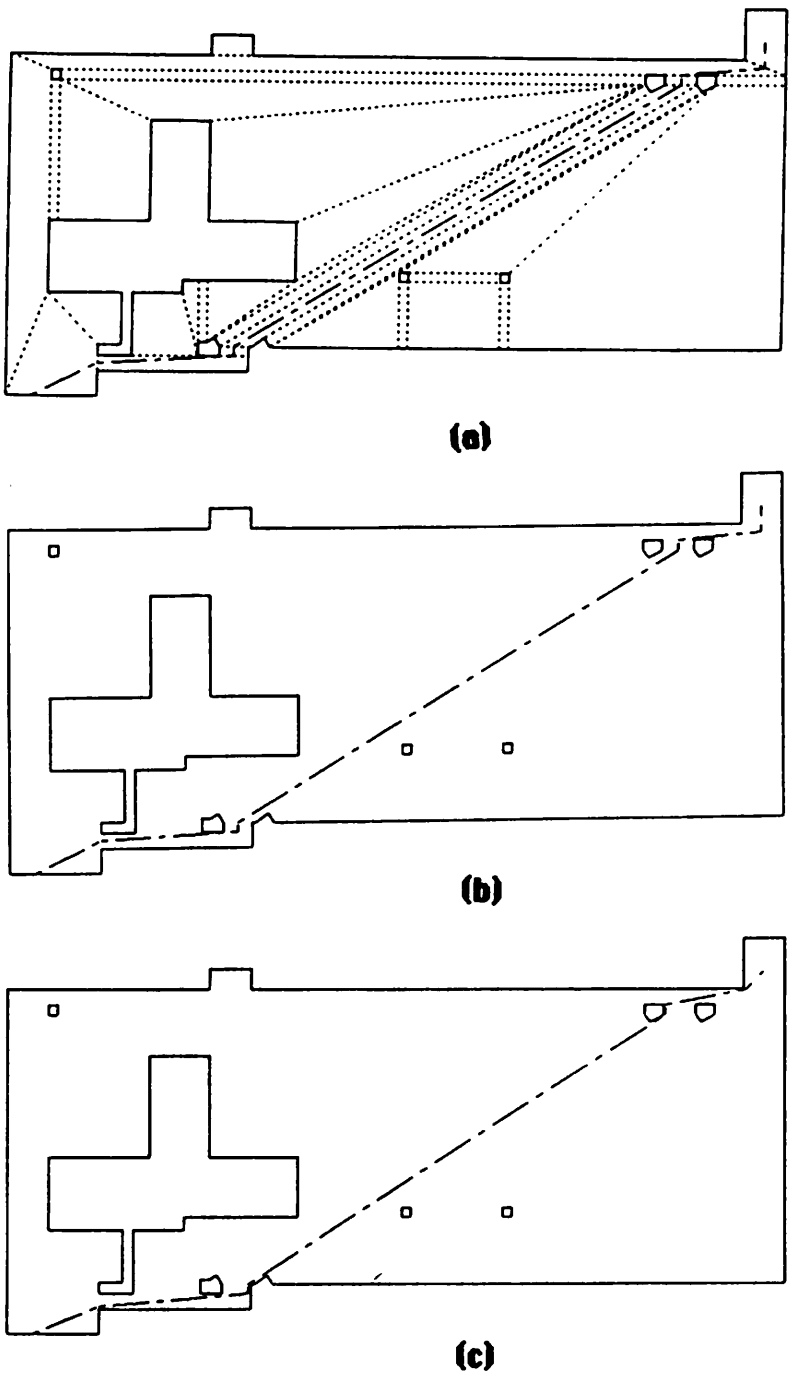


Figure 28. Multi-terrain Path Planning Example

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

(a) Initial raw path from A* search through midpoints of passable regions. The cost function included 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).

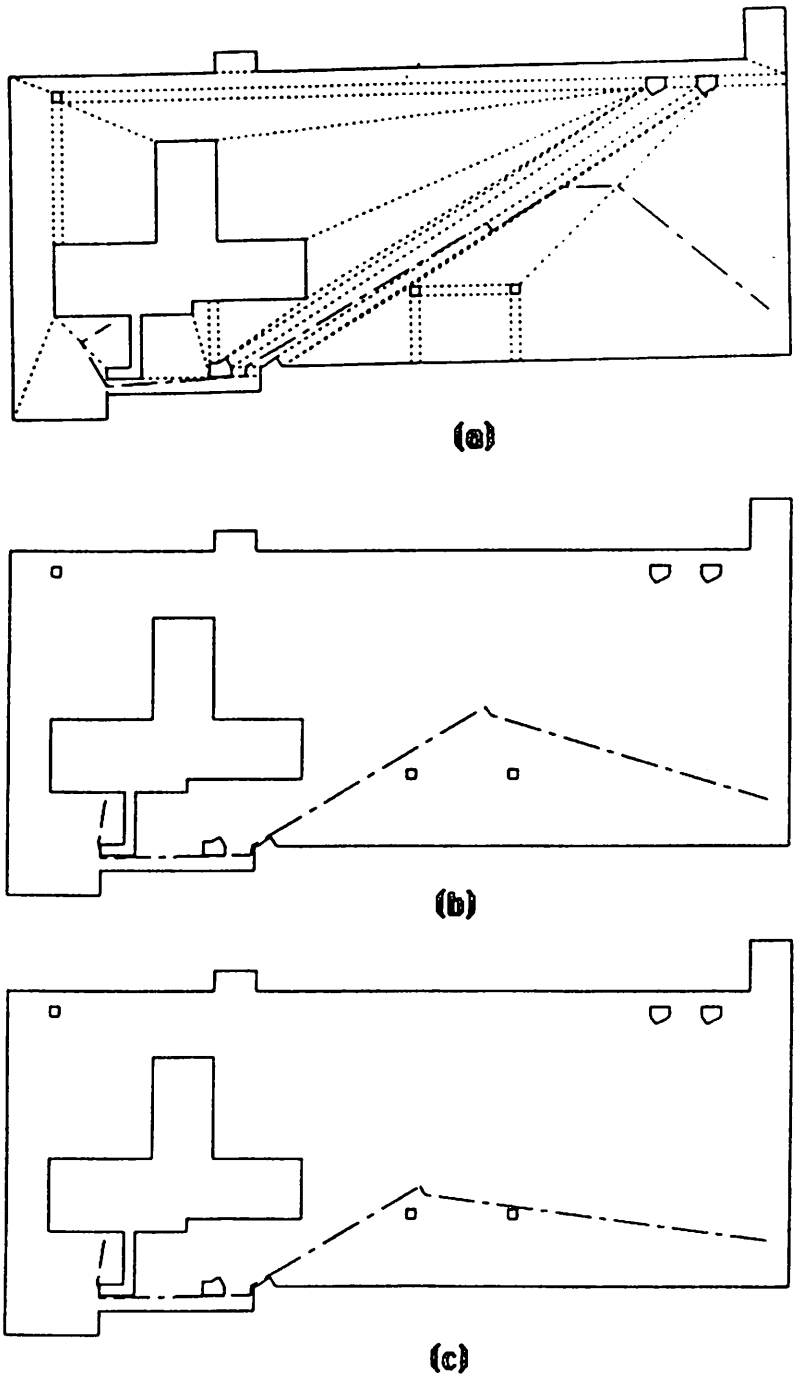


Figure 29. Another Multi-terrain Path Planning Example

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

(c) Final relaxed path - note the repositioning of the grass to gravel crossing.

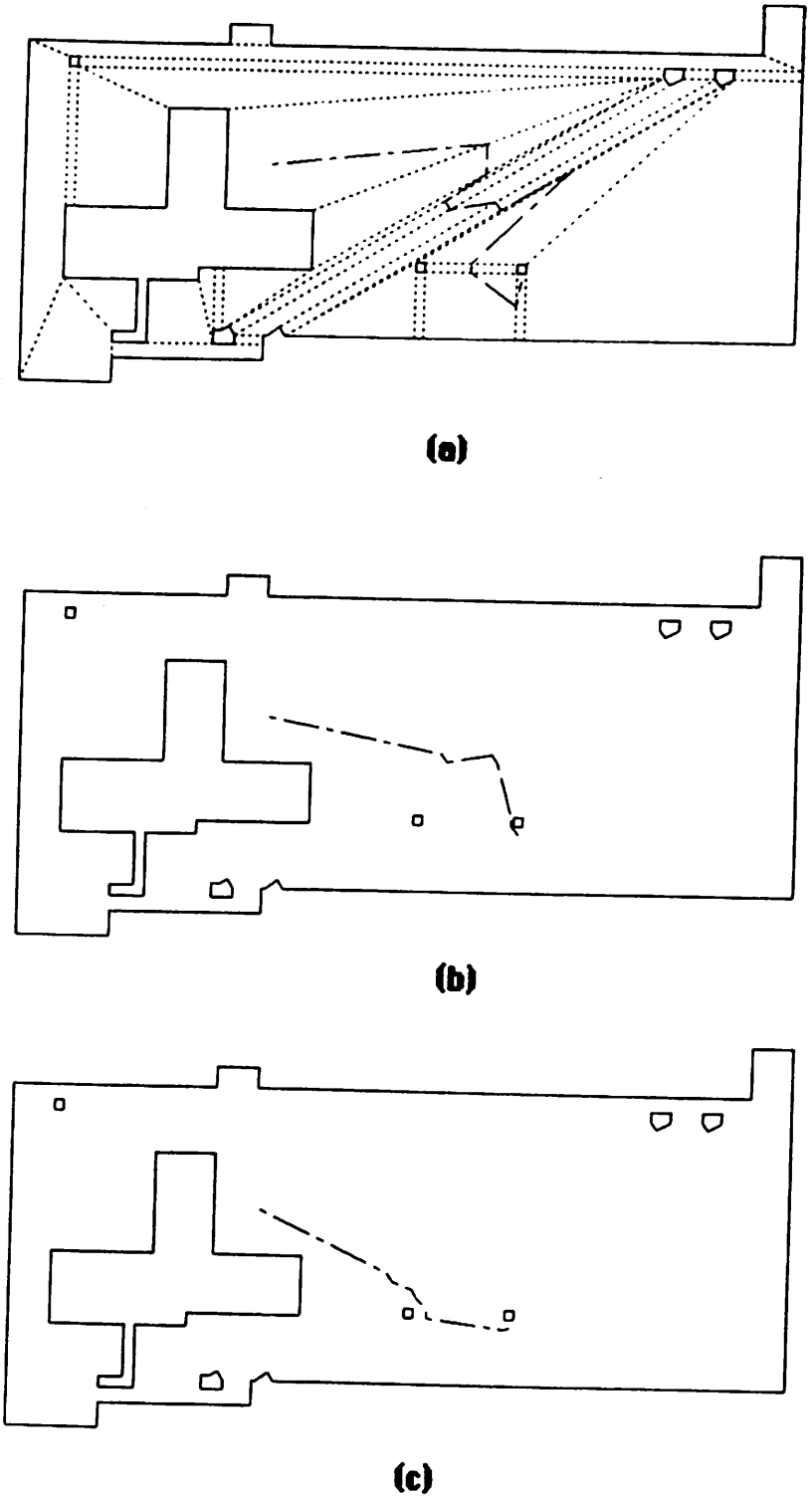


Figure 30. Yet Another Multi-terrain Path Planning Example
Start in lower right - goal in upper left.
(a) Initial raw path.
(b) Improved but unrelaxed path.
(c) Final relaxed path.

V. UMASS DRV Architecture Overview

A block diagram of the proposed system is presented in figure 31 with all control and communication links annotated. An initial implementation strategy, presented in more detail than fig. 31, appears in figure 32.

Central to the system is the hierarchical planner consisting of a pilot, navigator and mission planner and charged with the responsibility for decision making (see section III). A cartographer, whose task is to maintain and provide information on demand to requesting planning and sensory modules, is adjacent. The cartographer is solely responsible for maintaining the integrity of the world model. The mapbuilder described in section II.1.1 resides under the cartographer's jurisdiction.

A perception subsystem is delegated the task of fielding all sensory information from the environment, performing some preliminary filtering on that data to remove noise and then structuring it in a form that is acceptable to the cartographer. It is also the location where expectations are maintained to assist sensory processing.

The motor subsystem is the means by which the vehicle interacts with its environment in response to stimuli. Motors and motor controllers serve to effect the necessary changes. A vehicle interface is provided that guides the motors to carry out the requested response from the higher level processing. A homeostatic control subsystem is present, in the highest level of design (fig. 31), which is concerned with maintaining a safe internal environment for the robot.

This architecture has been strongly influenced by both Arbib's cybernetic views of intelligent systems [3,4] and the VISIONS image understanding system [11]. Concern with the action-perception cycle, the implementation of reflex arcs through panic shunts from the sensor interface to the motor subsystem, consideration of the global data structures in the context of long-term and short-term memory, and the use of perceptual and motor schemas, are important contributions to the overall design of a mobile robotic system. The incorporation of homeostatic control serves as the means to effect dynamic behavioral changes as a result of the robot's internal condition. For further information, this and the other modules are described in more detail in [6].

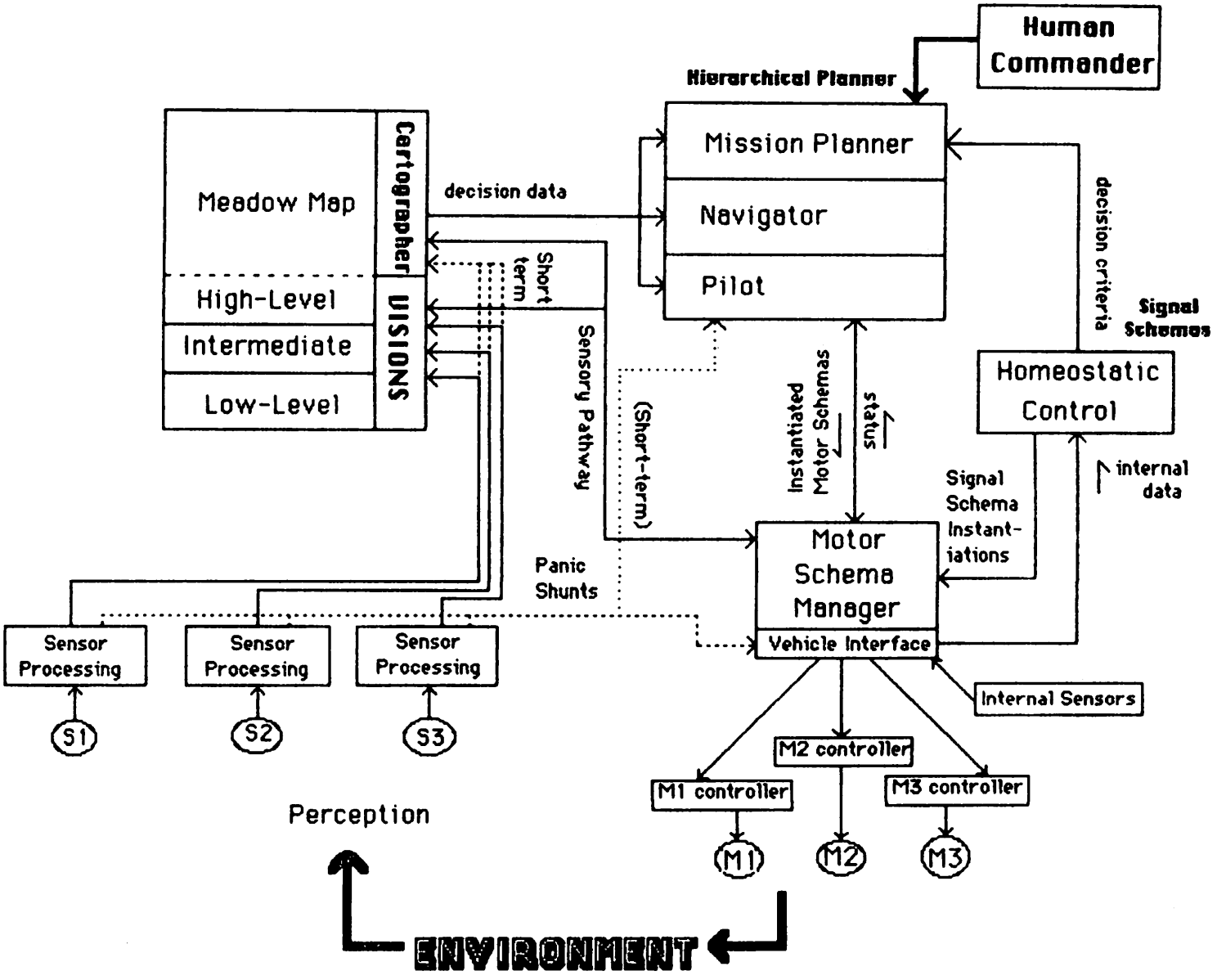


Figure 31. System Architecture for UMASS AuRA

Complete system design for the UMASS autonomous robot

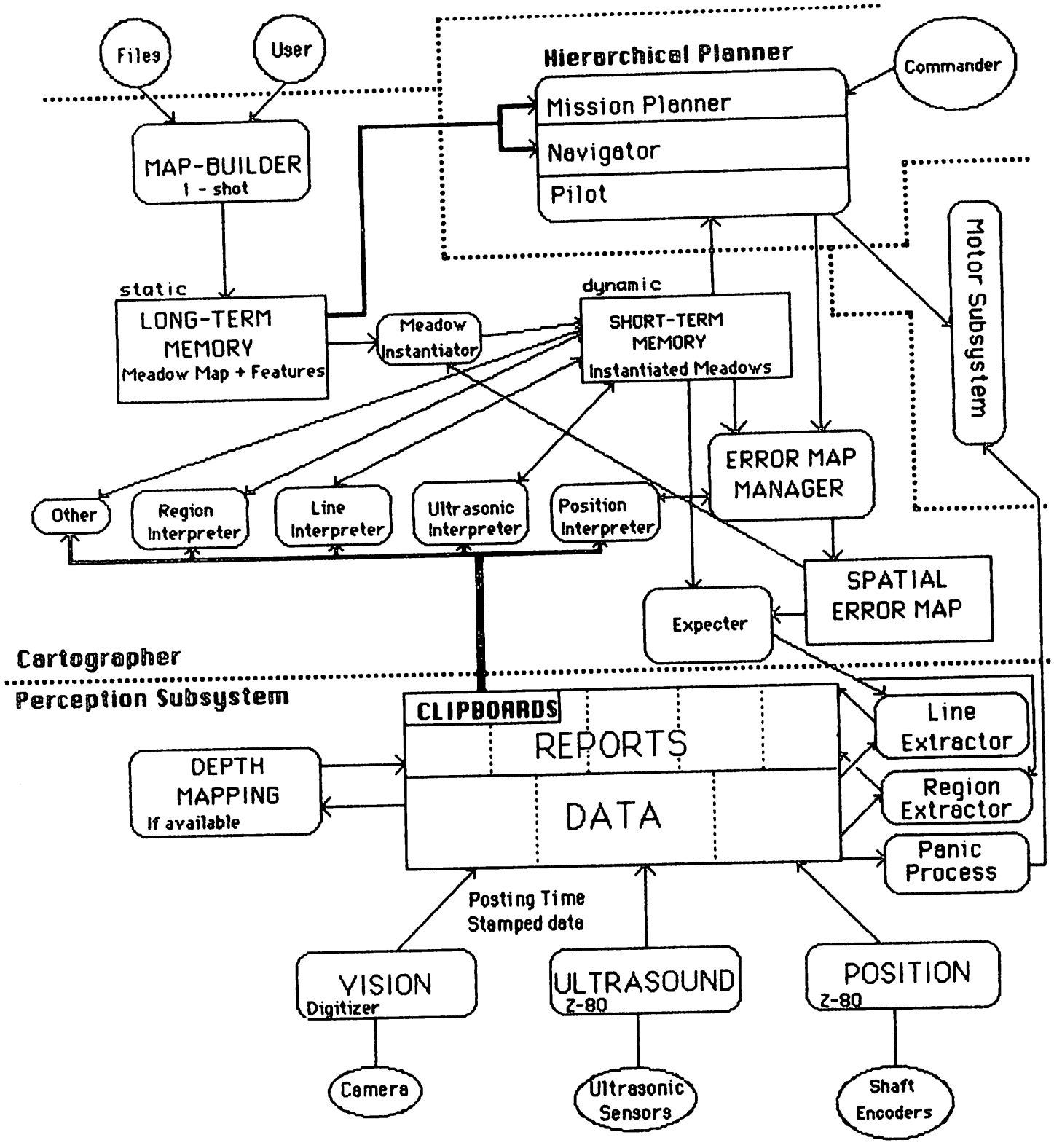


Figure 32.

First Design for Implementation of RuRR Architecture

VI. Summary and Conclusions

In order to produce a reasonable path through a partially modelled 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, error modelling, 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.

The output of the mapbuilder is consumed in part by the navigator component of the planner process whose duty it is to build a collision-free path through the partially modelled 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. This is then subjected to path improvement strategies which tighten and straighten the path subject to parameters specified by the mission planner. This allows 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.

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.

The navigation and long-term memory component of the overall UMASS architecture described within this paper is to be the basis for path planning for vision-based mobile robot experiments. The flexibility to accommodate diverse new representations, without any changes to the underlying path planning representation is one of the principal advantages of this approach.

Acknowledgements

This research was supported in part by the General Dynamics Corporation under grant DEY-601550 and the U.S. Army under ETL grant DACA76-85-C-008. The author would like to thank Prof. Ed Riseman, Prof. Al Hanson, Les Kitchen, Randy Ellis and the other members of the Laboratory for Perceptual Robotics and VISIONS groups for their encouragement and cooperation. This research is being conducted as part of the requirements for completion of the author's Ph.D. degree at the University of Massachusetts, Amherst.

References

1. Amarel, S., "On Representations of Problems of Reasoning about Actions", Machine Intelligence 3, 1968, reprinted in READINGS IN ARTIFICIAL INTELLIGENCE, ed. Webber & Nilsson, pp. 2 -22, Tioga, 1981.
2. Andresen, F.P., Davis, L.S., Eastman, R., and Kambhampati, S., "Visual Algorithms for Autonomous Navigation", Proc. IEEE Int. Conf. Robotics and Automation, St. Louis, Mo., pp. 856-861, 1985.
3. Arbib, M., "Perceptual Structures and Distributed Motor Control", HANDBOOK OF PHYSIOLOGY - The Nervous SYstem II, (V. Brooks, ed.), Chapter 33, pp. 1449-1465, 1981.
4. Arbib, M., THE METAPHORICAL BRAIN, J. Wiley & Sons, 1972.
5. Arkin, R.C., "Path Planning and Execution for a Mobile Robot: A Review of Representation and Control Strategies", COINS Technical Report, Computer and Information Science Department, University of Massachusetts, Spring 1986.
6. Arkin, R.C., Ph.D. Proposal, Computer and Information Science Department, University of Massachusetts, Spring 1986.
7. Chatila, R. and Laumond, J.P., "Position referencing and Consistent World Modeling for Mobile Robots", Proc. IEEE Int. Conf. Robotics and Automation, St. Louis, Mo., pp.138-145, 1985.
8. Crowley, J., "Navigation for an Intelligent Mobile Robot", CMU Robotics Institute Technical Report, CMU-RI-TR-84-18, August 1984.
9. Giralt, G., "Mobile Robots", Artificial Intelligence and Robotics, ed. Brady, Gerhardt, and Davidson, Springer-Verlag, NATO ASI series, pp. 365-394, 1984.
10. Giralt, G., Chatila, R., and Vaisset, M., "An Integrated Navigation and Motion Control System for Autonomous Multisensory Mobile Robots", Robotics Research, The First International Symposium, MIT Press, pp. 191-214, 1984.
11. Hanson, A. and Riseman, E., "VISIONS, A Computer System for Interpreting Scenes", COMPUTER VISION SYSTEMS (Hanson and Riseman eds.), Academic Press, pp. 303-333, 1978.
12. Iyengar, S. Jorgensen, C., Rao, S., and Weisbin, C., "Learned Navigation Paths for a Robot in Unexplored Terrain", IEEE Second Conf. On Artif. Intel. App., pp. 148-155, December 1985.
13. Keirsey, D., Mitchell, J., Payton, D., and Preyss, E., "Multilevel Path Planning for Autonomous Vehicles", SPIE Vol. 485, Applications of Artificial Intelligence, pp. 133-137, 1984.

14. Koch, E., Yeh, C., Hillel, G., Meystel, A., and Isik, C., "Simulation of Path Planning for a System with Vision and Map Updating", Proc. IEEE Int. Conf. Robotics and Automation, St. Louis, Mo., pp.146-160, 1985.
15. Krogh, B., "A Generalized Potential Field Approach to Obstacle Avoidance Control", RI Technical Paper, MS84-484, Soc. of Mech. Engr., 1984
16. Kuan, D.T., Zamiska, J., and Brooks, R., "Natural Decomposition of Free Space for Path Planning", Proc. IEEE Int. Conf. Robotics and Automation, St. Louis, Mo., pp.168-173, 1985.
17. Lozano-Perez, T., "Automatic Planning of Manipulator Transfer Movements", in ROBOT MOTION: Planning and Control, ed. Brady et al, pp. 499-535,1982.
18. Miller, D., "A Spatial Representation System for Mobile Robots", CH2152-7/85, IEEE , pp. 122-127, 1985.
19. Mitchell, J. and Keirsey, D., "Planning Strategic paths through Variable terrain data", SPIE Vol. 485, Applications of Artificial Intelligence, pp. 172-179, 1984.
20. Monaghan, G., "World Modeling and Path Planning for Autonomous Mobile Robots", manuscript 579-51.
21. Mobile Robot Laboratory Staff, "Towards Autonomous Vehicles", Annual Research Review, The Robotics Institute, Carnegie-Mellon University, 1984.
22. Moravec, H. and Elfes, A., "High Resolution Maps from Wide Angle Sonar", CH2152-7/85 1985 IEEE, pp. 116-121, 1985.
23. Nilsson, N., PRINCIPLES OF ARTIFICIAL INTELLIGENCE, Tioga Publishing Co., 1980.
24. Nitao, J. and Parodi, A., "An Intelligent Pilot for an Autonomous Vehicle System", IEEE Second Conf. On Artif. Intel. App., pp. 176-183, December 1985.
25. Ooka, A., Ogi, K., Wada, Y., Kida, Y., Takemoto, A., Okamoto, K., and Yoshida, K., "Intelligent Robot System II", ROBOTICS RESEARCH, Second Int. Symposium, ed. Hanafusa and Inoue, MIT Press, pp. 341-347, 1985.
26. Parma, C., Hanson, A. and Riseman E., "Experiments in Schema-driven Interpretation of a Natural Scene", COINS Tech. Report 80-10, Computer and Information Science Department, University of Massachusetts, 1980.
27. Parodi, A. M., "Multi-Goal Real-Time Global Path Planning for an Autonomous Land Vehicle using a High-speed Graph Search Processor", Proc. IEEE Int. Conf. Robotics and Automation, St. Louis, Mo., pp. 161-167, 1985.
28. Pearson, G. and Kuan, D., "Mission Planning System for an Autonomous Vehicle", IEEE Second Conf. On Artif. Intel. App., pp. 162-167, December 1985.
29. Sedgewick, R., ALGORITHMS, Addison-Wesley, p. 313, 1983.

30. Stentz, A. and Shafer, S., "Module Programmer's Guide to Local Map Builder for ALVan", CMU Computer Science Department, July 1985.
31. Tachi, S. and Komoriya, K., "Guide Dog Robot", ROBOTICS RESEARCH, Second Int. Symposium, ed. Hanafusa and Inoue, MIT Press, pp. 333-340, 1985.
32. Thorpe, C., "Path Relaxation: Path Planning for a Mobile Robot", CMU Robotics Institute Technical Report CMU-RI-TR-84-5, April 1984.
33. Thorpe, C., Kanade, T., and Shafer, S., "ALV System Integration Plan", DRAFT, The Robotics Institute, CMU, May 1985.
34. Tsuji, S., "Monitoring of a Building Environment by a Mobile Robot", ROBOTICS RESEARCH, Second Int. Symposium, ed. Hanafusa and Inoue, MIT Press, pp. 349-356, 1985.
35. Weems, Charles, "Image Processing on a Content Addressable Array Parallel Processor, COINS Technical Report 84-10, Computer and Information Science Department, University of Massachusetts, 1984.
36. Weymouth, T.E., "Using Object Descriptions in a Schema Network for Machine Vision", Ph.D. dissertation, Computer and Information Science Department, University of Massachusetts, May 1986.

Appendix

Computational Costs

All CPU times are from a moderately loaded VAX-11 750 (8 MB memory).

A. Mapbuilder

Recognizing that the mapbuilder code is far from optimized, still some basic interpretations of the performance of the underlying algorithms can be made. Each component of the mapbuilding algorithm will be discussed in turn.

Border Growing

The border growing algorithm is dependent on the number of vertices to be grown. The time required to grow the border for figure 11 was approx. 5.2 CPU sec (28 vertices). A larger border (57 vertices) took approx. 22 seconds.

Obstacle attachment

The obstacle attachment component of the mapbuilding algorithm is currently the least efficient part of the code. Massive improvements in CPU speed can be made. This phase of the mapbuilding is the most time consuming and appears to be exponential in order. The total time to grow and attach 1 typical obstacle is 0.05 sec (to fig. 11). This increases to 28 seconds for 7 obstacles and is dependent on the number of vertices of the border, number of obstacles, and number of obstacle vertices. Efficiency is not a prime factor during the long term memory mapbuilding phase as it is only compiled once at the start of the run. Nonetheless, for more complex environments this part of the code must be cleaned up.

Decomposition Times

Perhaps of more interest is the time required for the recursive decomposition of the single region output by the obstacle attachment algorithm. Surprisingly rapid results were obtained. For figure 11, computational times from 3.2 to 12.2 CPU seconds were observed (including merge). The general trends were as expected; choosing the first concave vertex consistently outperformed the least or most concave modes. Of greater significance was the choice of the victim vertex: the opposite vertex performed best, followed by leftmost then rightmost victim. In many cases choice of the most opposite vertex resulted in decomposition times about 1/4 of the time required for rightmost victims.

B. Navigational path finder

8 different paths using four different start and end point pairs and two different safety margins were computed using the decomposition shown in figure 11a. Many of these paths are shown in figures 14-16 and 18-20.
No claim is made for the efficiency of the implementation. Significant time savings could probably be acheived (especially in the path improvement section) if the code was rewritten. These should be considered relative figures, not those that indicate the lower bounds of the algorithm.

Search

As would be expected the search time for the A*-3 algorithm is considerably higher than that for the A*-1 method. The total number of possible nodes to be expanded triples, while the path solution space is cubed. Although the A*-3 method is definitely more costly, it is not prohibitive as the data below confirms.

Range for A*-1 (search): 0.46-1.38 sec (avg: 0.95 sec)
Range for A*-3 (search): 1.51-6.15 sec (avg: 4.07 sec)
Percentage Increase for A*-3 over A*-1 (search):
Range: 188% - 781%
Average: 455%

On the average, it costs 4.5 times as much to search using the A*-3 method as compared to the A*-1 method on the decomposition shown in figure 11a.

Path Improvement

A*-3 will perform better than A*-1 in the path improvement component of the algorithm as it can completely bypass the initial tautness part. This results in significant time savings which offset some of the additional computation as described above.
Using the same 8 paths as in the search component, these results follow:

Range for A*-1 (path imp.): 0.76-6.91 sec (avg: 3.18 sec)
Range for A*-3 (path imp.): 0.43-4.20 sec (avg: 1.46 sec)
Percentage of A*-3 over A*-1 (path improvement):
Range: 12% - 83%
Average: 58%

Total Path time (simple terrain - no relaxation)

In all but one case the A*-1 algorithm outperformed the A*-3 method in CPU time. The results are not too widely separated due to the shorter path improvement times required for A*-3. This shows the feasibility of this method in certain instances. It should also be noted that in every case (except one) the A*-3 algorithm came up with a path equal or lower in cost to that of the A*-1 algorithm. Only when high safety factors were involved did A*-1 derive a lower cost path. This is misleading as the goal was to produce safe, not short paths. If the straightening was turned off the A*-3 algorithm would have outperformed A*-1 as expected.

Range for A*-1 (total): 1.22-7.59 sec (avg: 4.13 sec)
Range for A*-3 (total): 2.14-8.66 sec (avg: 5.54 sec)
Percentage increase per run for A*-3 over A*-1 (total):
Range: 75 - 271%
Average: 162%

On the average it took 62% longer to find a path using A*-3 than A*-1 when all path improvement techniques are in use.

Path cost savings

Of the three paths produced by the A*-3 method that were of lower cost than those produced using the A*-1 method, the first was 8% longer, the second was 7% longer, and the third was 2% longer. These are rather paltry cost savings for the mobile robot domain in which dynamic obstacles and/or uncertainty can render any precomputed path useless.

Multi-terrain transition zone relaxation

Path relaxation figures were harder to collect and are highly dependent on the relaxation step size. The increment was set to 3.0 feet, about the diameter of the robot. Obviously, the number of transition zones to be relaxed also has a profound effect on the relaxation time.

For a single relaxation zone, (the longest concrete-grass border: fig. 24), relaxation times observed ranged from a minimum of 0.40 CPU sec to a maximum of 4.13 sec. For two zones, (grass across gravel path to other grassy section), times from 3.06 to 5.23 seconds resulted.

The relaxation for Figure 25 (two zone) took 0.38 seconds (overhead only). Fig. 26 (2 zones) took 3.73 seconds and fig. 27 (3.5 zones) took 10.7 seconds. The longest observed relaxation time (4 zones) was 22.81 seconds. It is anticipated that most paths for the mobile robot will not involve numerous transition zone crossings.