# General Disclaimer

## One or more of the Following Statements may affect this Document

- This document has been reproduced from the best copy furnished by the organizational source. It is being released in the interest of making available as much information as possible.

- This document may contain data, which exceeds the sheet parameters. It was furnished in this condition by the organizational source and is the best copy available.

- This document may contain tone-on-tone or color graphs, charts and/or pictures, which have been reproduced in black and white.

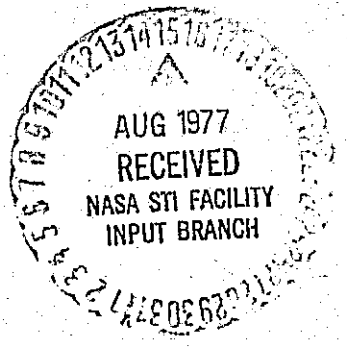
- This document is paginated as submitted by the original source.

- Portions of this document are not fully legible due to the historical nature of some of the material. However, it is the best reproduction available from the original submission.

JPL PUBLICATION 77-20

# The Navigation System of the JPL Robot






National Aeronautics and Space Administration

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California 91103

TECHNICAL REPORT STANDARD TITLE PAGE

| 1. Report No. JPL Pub. 77-20 | 2. Government Accession No. | 3. Recipient's Catalog No. |
|---|---|---|
| 4. Title and Subtitle<br>The Navigation System of the JPL Robot | | 5. Report Date<br>July 15, 1977<br>6. Performing Organization Code |
| 7. Author(s)<br>Alan M. Thompson | | 8. Performing Organization Report No. |
| 9. Performing Organization Name and Address<br>JET PROPULSION LABORATORY<br>California Institute of Technology<br>4800 Oak Grove Drive<br>Pasadena, California 91103 | | 10. Work Unit No.<br>11. Contract or Grant No.<br>NAS 7-100<br>13. Type of Report and Period Covered<br>JPL Publication |
| 12. Sponsoring Agency Name and Address<br>NATIONAL AERONAUTICS AND SPACE ADMINISTRATION<br>Washington, D.C. 20546 | | 14. Sponsoring Agency Code |

15. Supplementary Notes

16. Abstract

The control structure of the JPL research robot and the operations of the navigation subsystem are discussed. The robot functions as a network of interacting concurrent processes distributed among several computers and coordinated by a central executive. The results of scene analysis are used to create a segmented terrain model in which surface regions are classified by traversibility. The model is used by a path-planning algorithm, PATH*, which uses tree search methods to find the optimal path to a goal. In PATH*, the search space is defined dynamically as a consequence of node testing. Maze-solving and the use of an associative data base for context-dependent node generation are also discussed. Execution of a planned path is accomplished by a feedback guidance process with automatic error recovery.

| 17. Key Words (Selected by Author(s)) | 18. Distribution Statement |
|---|---|
| Mathematical and Computer Sciences (General)<br>Cybernetics<br>Lunar and Planetary Exploration (Advanced) | Unclassified - Unlimited |

| 19. Security Classif. (of this report) | 20. Security Classif. (of this page) | 21. No. of Pages | 22. Price |
|---|---|---|---|
| Unclassified | Unclassified | 45 | |

JPL PUBLICATION 77-20

# The Navigation System of the JPL Robot

Alan M. Thompson

July 15, 1977

National Aeronautics and
Space Administration

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California 91103

# PREFACE

The work described in this report was performed by the Information Systems Division of the Jet Propulsion Laboratory.

CONTENTS



FIGURES

## ABSTRACT

The control structure of the JPL research robot and the operations of the navigation subsystem are discussed. The robot functions as a network of interacting concurrent processes distributed among several computers and coordinated by a central executive. The results of scene analysis are used to create a segmented terrain model in which surface regions are classified by traversibility. The model is used by a path-planning algorithm, PATH*, which uses tree search methods to find the optimal path to a goal. In PATH*, the search space is defined dynamically as a consequence of node testing. Maze-solving and the use of an associative data base for context-dependent node generation are also discussed. Execution of a planned path is accomplished by a feedback guidance process with automatic error recovery.

## 1. Introduction

The Robotics Research Program at the Jet Propulsion Laboratory is aimed at developing the capabilities in machine intelligence systems required for a semi-autonomous vehicle to be used in remote planetary exploration. To achieve this end, a "breadboard" robot has been constructed (Fig. 1) to serve as test bed and demonstration tool for the programs and hardware. Research is being conducted in the areas of robot vision, problem-solving and learning, hardware and system architecture, motor-function control in manipulation and locomotion, as well as the terrain modeling and navigation tasks described herein.

In the JPL experiments, the robot is deposited in an unknown laboratory environment consisting of many arbitrary obstacles (rocks, walls, and other objects) and is given tasks such as finding and collecting selected rock samples. As a robot subsystem, the navigation system has the responsibility of finding an unobstructed path to a designated goal and then controlling the vehicle's movement along the path. To do this, it must maintain an internal representation of its environment from sensory input for use in the planning phase and then use additional sensory input to monitor execution of movement. The environment is initially completely unknown, and the path planner requests updates to the terrain model as needed for planning.

A terrain model was chosen for the robot that simplifies the task of path planning while simultaneously providing a means of representing large areas of terrain in a compact, segmented, hierarchical structure that is easily updated or extended. Having a numeric description of the location and shape of obstacles allows the path planner to accurately model the characteristics of the vehicle while conducting the optimal path search, so that the resulting path is in a form that is readily executable by the guidance programs to within a known error tolerance.

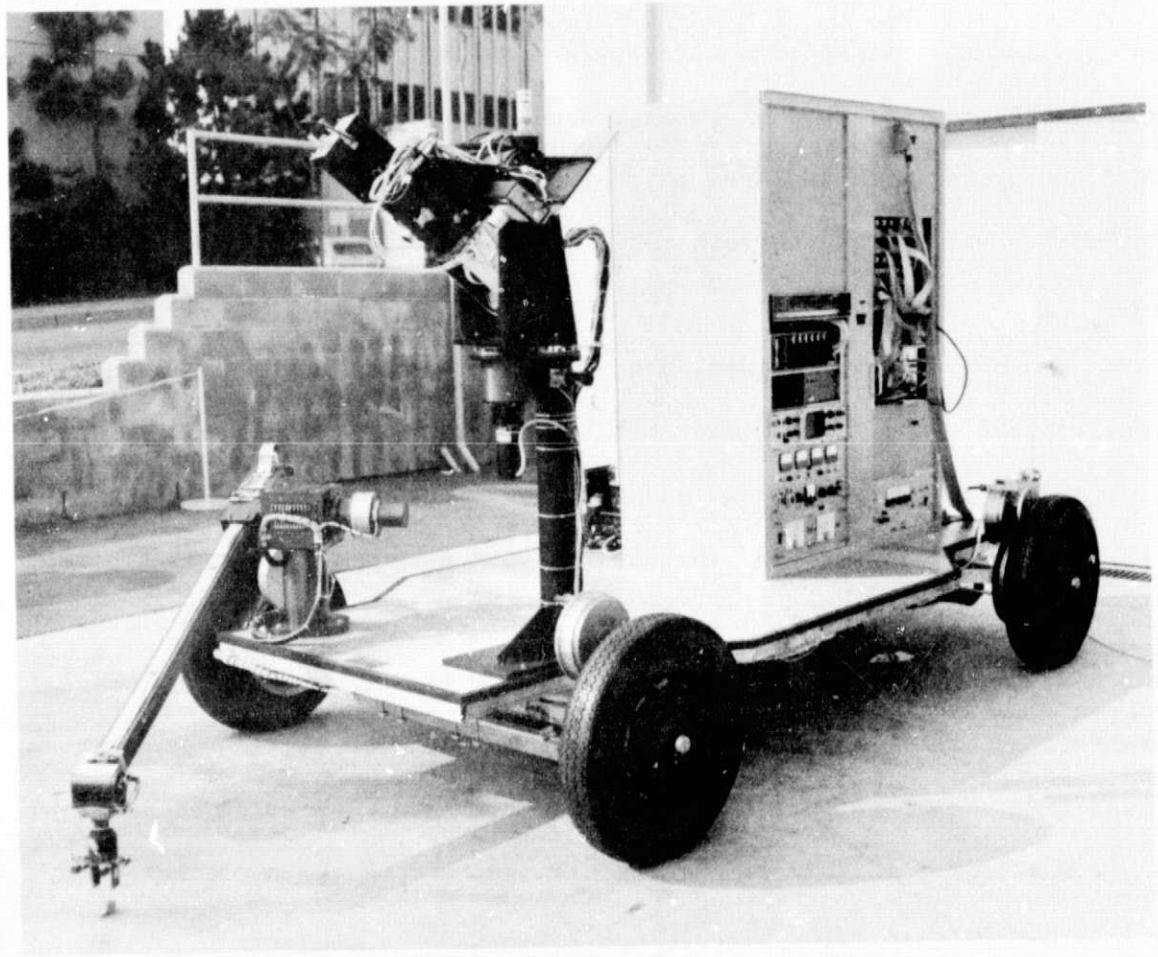


Fig. 1. The JPL Research Robot

## 2. Robot System Structure

The JPL robot operates as a hierarchy of separate concurrent processes which are distributed among three computers. The main control structure (Fig. 2) consists of a Robot Executive (REX) which communicates with the operator via the "ground system." Other processes, whose functioning is coordinated by the executive (though not necessarily determined by it), perform the tasks of vision, manipulation, and navigation. The control hierarchy is not strictly enforced, however, as processes may interact freely in such functions as hand-eye coordination, etc. Recent additions to the system include processes for error recovery and problem-solving, which will be the nucleus of a system for automatic planning, error correction and learning. The sensory-motor processes have subprocesses on the minicomputer containing the actual vehicle interfaces. Processes suspend themselves when not needed.

Communication between the processes is handled by a shared program segment "mailbox" method, similar to that of the Stanford Hand-Eye System (FS1). Messages passed within one computer are merely stored in the appropriate slot in the shared segment, whereas intercomputer messages are transmitted by a separate communications process to the appropriate machine and then deposited. This structure is extensible to multiple processors. A process need not know on which CPU another process runs. At present, three CPU's are connected: a DEC PDP-10, a GA SPC-16 minicomputer, and an IMLAC PDS-1D graphics system. The "high-level" processes run on the PDP-10, and are implemented in the SAIL language and, recently, in LISP. In the minicomputer, FORTRAN and assembly codes are used. All interfaces to the vehicle hardware are currently contained in the minicomputer, but since it has limited capacity for parallel processes, the use of a microcomputer network is being investigated.

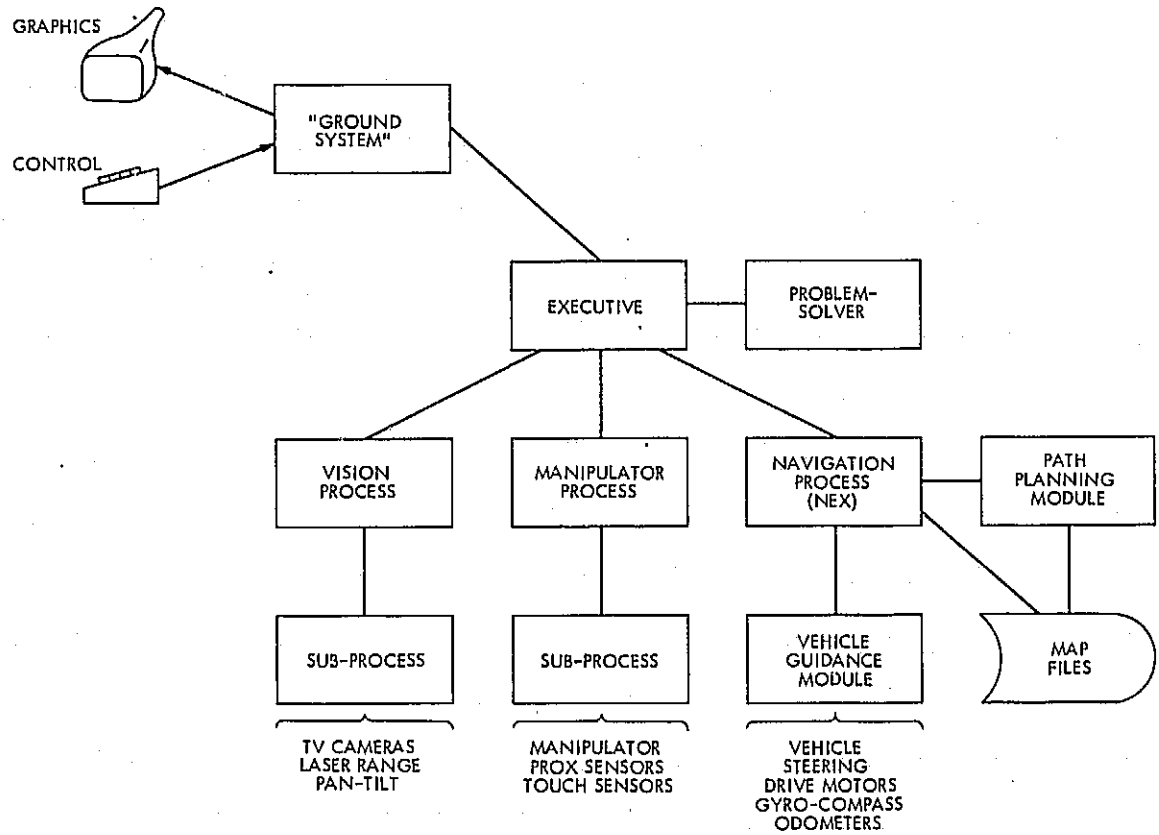



Fig. 2. Robot Control Structure

The navigation system runs as three concurrent processes: the navigation executive (NEX), the path planner module (PPM), and the vehicle guidance module (VGM). Both NEX and PPM access the terrain model files. NEX is the controlling process for all navigation functions. It contains the command interface to the robot executive which translates acceptable commands into the appropriate action. The NEX process invokes the path planner upon request and processes map update requests generated by PPM. Map requests are forwarded to the vision system, and replies from the vision system are processed by NEX procedures into the terrain model format and added to the data base. The required transformations will be discussed below. NEX also invokes the VGM either to move the vehicle along planned paths or to execute movement primitives. The format of NEX messages is discussed in Appendix 3.

## 3. The Terrain Model

In order to perform path planning, the navigation system must maintain a model of the robot's environment in which features that would affect the vehicle's movement are represented. Since the area explored by the robot may be large and many such obstructions encountered, it is desirable to have a terrain model that is partitioned into segments of a convenient size, within which the features have a compact numerical representation. The map segments should normally reside in bulk storage and should have an access structure that allows rapid loading when needed. The model used in the navigation system was designed to meet these requirements. In addition, since the testing of proposed path links is the major computational expense of path-planning, the representation is optimized for this purpose. The segments, when loaded, form a hierarchy for accessing the barrier descriptions and, as discussed below, the structure of the descriptions was chosen to facilitate the process of path search.

The territory represented in the terrain model is partitioned into map sectors by a fixed lattice of grid lines. The grid lines are drawn parallel to the axes of the robot's absolute (lab-based) coordinate system and are equally spaced, so that the sectors are square and may be numbered relative to the origin. The sector number may thus be used to compute the absolute coordinates of the sector. The map sector is defined by providing to the model a file containing the terrain description for the area covered; the resultant directory of sectors is thus analogous to a catalog of charts. At present, a three-meter grid spacing is used for within the lab.

The primary source of input to the model is the vision system. When a description of a sector is requested by NEX for use by the planner, the request is sent to the vision system, which performs the required terrain analysis. Here the perceptual problem is that of producing from stereo TV and laser rangefinder inputs a segmentation of the area coverd by the requested sector into regions described as traversible, obstructed (nontraversible) or unknown (YC1, YC2). The information available from a new viewpoint must be combined with that already in the model to produce a new sector description. This map maintenance process is subject to errors which will be discussed later.

It is of interest to note that what is described in the map is the traversibility of the terrain surface. This is adequate for the purposes of path-planning and allows a two-dimensional representation for all types of obstructions, as shown in Fig. 3. As a greater variety of terrain descriptors are added to the model, such as slope and altitude, information pertaining to the third dimension may influence the cost of a path on the goals of the robot. Should it become necessary to describe multilevel structures, new description types may easily be added to the model.

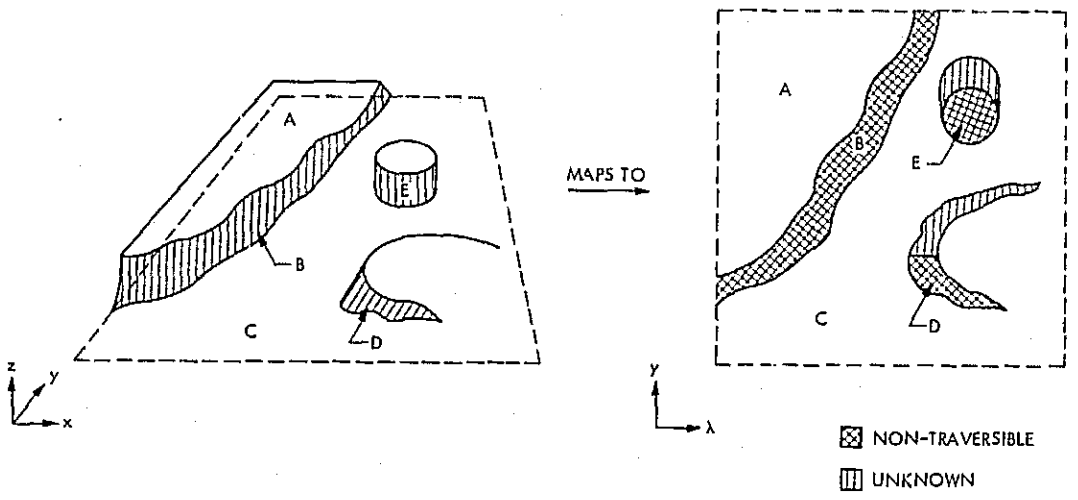



Fig. 3. A Terrain Map Sector

Within a map sector, terrain regions are described by polygonal boundaries which are represented as lists of the vertices (corners) of the polygon. At present, regions are classified either as obstacle (nontraversible) or unknown. All else is presumed clear and traversible. Some generality is lost by dividing barriers that overlap a sector boundary between the adjacent sectors, but the path planner detects this case and regards the parts as a single obstacle. There is also a capability for a region at sector level to represent a cluster of objects, in which case the description contains a list of other boundary or cluster descriptors. When loaded into memory, the descriptors for regions within a sector become the "datums" of SAIL "items," so that the associative search features of SAIL may be used in the path-planning process (VL1).

In new map sector descriptions provided by the vision system, the borders of the terrain regions are represented by lists of predefined unit vectors, in which consecutive vectors describe (15 cm) unit steps along the boundary. The navigation system must translate this representation into the polygon vertex list used in the map. The process, described in Appendix 1, locates the minimum number of corners necessary to describe the boundary to within a known tolerance. The operations include smoothing, elimination of inaccessible regions in the interior of closed boundaries and finally, an iterative polygon approximation to the boundary. The resulting description is a list of corners in order of their connection, plus a centroid and clearance radius used to simplify the path testing process (Appendix 2 and F1).

In the normal mode of operation, when the PPM is assigned a planning task, its first operation is to determine which map sectors lie along the straight line path from start to goal. Queries for the indicated sectors are sent to the vision system, which then determines if a map update is possible

for any requested sector. Each sector update is dated, so that if the vehicle has moved since the last update, additional information may be available from the new point of view. In the latter case, the vision system would provide a new sector update; otherwise it would indicate that the planner should use the existing model. As the queries are answered the corresponding sector maps are loaded into memory for use by the path-planning process. The boundary between the loaded and not-loaded sectors is represented in the map as a special obstacle. If, during the course of planning, the map border is encountered, the map must be expanded by adding additional sectors. The appropriate queries are then generated and the process is suspended until the replies are received. Thus, the sector loading mechanism forms a sort of "virtual memory" for the segmented map. The system is structured so that the map updating process of the vision system may operate freely in parallel with the navigation process, collecting terrain data available from the current viewpoint, even though the updates may not be needed immediately.

## 4. The Path-Planning Process

The task of using the terrain model to find an unobstructed path to a selected goal is performed by the path-planning module. Naturally, there may be many alternative routes to a goal, so a measure of path cost is introduced to define a selection criterion for optimal path search. At present, the cost of a path is the distance along it, but other measures, such as time or energy, may be used in the future. The cost metric could also be redefined by the system from one path search to the next to solve specific problems.

True optimal search is possible only when all obstructions in the areas encountered by the search are known. The planner, however, has access only to terrain information represented in the map at the time of planning plus

those features observable from the robot's current location. From the robot's initial point of view, much of the terrain may be obscured (by occlusion or distance), but since the obscured terrain is also represented in the model by unknown regions, the planner can detect the case where a proposed traverse intersects the unknown and may terminate the search at that point. An optimistic executive could move the robot to that point (or near enough to classify the unknown area), update the map, and plan a new path to the original goal. A pessimistic executive could regard unknown terrain the same as an obstruction and look for a (possibly) longer path. The planning algorithm is capable of functioning in the latter mode, and may be told to switch to this mode after detecting (and remembering) the first (possibly optimal) partial path.

With the selection of a cost metric and a function for defining nodes in the search space, traditional methods of optimal path search may be used (HNR1). The decision to use the energy required for the traverse (which in a zero slope lab environment translates to distance) as a metric was motivated by the need to demonstrate a system suitable for application in an actual robot planetary exploration vehicle, as compared to, for example, SRI's robot, SHAKEY, which simply used search depth as the cost, resulting in a path with the fewest number of links (R1). Although, as one would expect, the strategy of avoiding an obstruction by generating candidate paths to either side of it is a feature common to previous path planners, the minimum distance requirement, as shown below, demands a more complex node generation scheme than that required for simpler cost measures. Also, proper choice of successor nodes, combined with pruning, keeps the problem one of tree search rather than graph search.

Path-Planning Defined

For the purposes of this discussion, path planning will be defined for a point vehicle and then elaborated for the finite case. The map will be defined in terms of a set of vertices, $\underline{V}$, and a set of nontraversible walls, $\underline{W}$, where

$$\underline{W} = \{A_1B_1, \ldots, A_mB_m;\ A_i, B_i \in \underline{V}, \text{ and the segment } A_iB_i \text{ is part of some closed polygon}\}.$$

The set of vertices, with the addition of two points defining the start and the goal, define $\underline{P}$, the set of all points in the map. Then for each point $P_i$ in $\underline{P}$ we define a set $\underline{L}_i$, where

$$\underline{L}_i = \{P_iP_j;\ j \neq i,\ P_iP_j \in \underline{W} \text{ or } P_j \in \underline{V} \text{ and } P_iP_j \text{ cuts no polygon in the map}\}.$$

$\underline{L}_i$ is called the link set of $P_i$ and is composed of the walls adjacent to $P_i$ (if $P_i$ is a vertex) plus all line-of-sight links from $P_i$ to other members of $\underline{P}$. A path from the start to some point $Q \in \underline{P}$ is a list of links $(P_{k_1}P_{k_2}, P_{k_2}P_{k_3}, \ldots P_{k_{n-1}}P_{k_n})$, where $P_{k_1} = \text{start}$, $P_{k_n} = Q$, and $P_{k_i}P_{k_{i+1}} \in \underline{L}_{k_i}$, etc.

Path search may be defined in these terms. We define a successful node in the search as a point (in $\underline{P}$) to which there is a known path. The link of a node is the path segment from the parent node to the given node. Similarly, successor links go from a node to successor nodes. A goal link is the link from a node to the goal. For each node $P_{k_i}$ in the search, we select successor links from $\underline{L}_{k_i}$ until the (optimal) path is found. In our map definition, however, the link sets must be derived, since only the walls are represented. By deriving the link set as needed, we avoid the combinational explosion that would result from representing the link set of each point in the map. The link set from a node is found by proposing candidate links to be tested for membership

in the set. Normally, the first candidate is the goal link from the node. A candidate link $P_kP_{k'}$ is tested by examining the set $\underline{W}$ for intersections. If $P_kP_{k'} \in \underline{W}$, the membership assertion is true. If $P_kP_{k'}$, cuts no wall, the assertion is true, and $P_{k'}$ becomes a node in the search with $P_k$ as its parent and (typically) the goal link from $P_{k'}$ as its successor candidate. However, if there is an intersection with some $A_iB_i$ in $\underline{W}$ the assertion is false, and lines $P_kA_i$ and $P_kB_i$ then become candidate links for future testing. Note also that lines $P_jA_i$ and $P_jB_i$, j=1,2,...k-1, may also be candidates for their own link sets, but, in an optimal path search, all but one of the successor candidates to $A_i$ or to $B_i$ may be pruned as discussed below.

With the goal of finding the minimum cost path from start to finish, the A* algorithm (HNR1) may be applied, with modifications, to perform optimal path search in the space defined above. Given a node generating function, $\Gamma$, and an admissible node cost criterion, the A* method is guaranteed to find the lowest cost path through the space defined by $\Gamma$, if it exists. In PATH*, the algorithm used in the JPL robot, a node in the search space is actually what is described here as a path link, since it is the link record that is tested for success or failure. Also, the notions of parent and successor are different from those of A*. After a path link is tested, PATH* may select one or more points in the map as destinations for candidate links, but, by a procedure discussed below, the search tree is traced back to find the optimal parent for each chosen destination. The destination then becomes a candidate for the link set of that parent. New candidate links generated as a consequence of the failure of a link are said to be "engendered" by the failed link and are associated with it for possible use in subgoal generation. The algorithm uses these and other relations

between links to form an associative data base describing the search context for use in node generation and pruning. These techniques will be illustrated in the examples below.

The cost function for a link is the actual distance along the (unique) path from the start to its endpoint plus the straight line distance remaining to the goal. Since the straight-line distance from the endpoint to the goal is the lower bound on the actual path cost of reaching the goal, this heuristic estimate satisfies the admissibility requirement of A*. In certain maze-like configurations, this heuristic estimate may be increased to improve search efficiency. As in A*, untested candidate links are kept in a list ordered by this cost estimate, so that the main loop may always select the least-cost link for the next test.

The question remains, of course, whether the successor generation procedure described above is capable of covering the link set from a node or, in the case of optimal search, of generating that member of the link set that lies on the optimal path from the given node. This turns out to be the heart of the planning problem, because there are maze-like configurations of concave barriers where the search must move away from the goal in order to reach it. Barriers with concave boundaries must be avoided on the edge-by-edge (wall following) basis discussed above, rather than by generating links to the extreme tangent points of the barrier (which is adequate for convex barriers). Also, as will be shown, it is often necessary to choose as the successor candidate link from a new node some link other than the goal link. This alternative to the goal link is called a subgoal link, whose end node (the subgoal) is the end node of the link whose failure (obstruction) led to the generation of the given (successful) link.

To illustrate these principles, consider the search space shown in Fig. 4a. The straight path from start to goal is obstructed by wall CD. Candidates SC and SD are generated. In the search tree notation shown in Fig. 4b, the line cutting the tree link from parent to node implies an obstruction of the physical link, as well as indicating that the successor nodes have the same parent as the failed link. The parent node is mentioned in the box for the successor for convenience, and also illustrates the interchangeability of the notions of node and physical link.

The successors would be tested in order by cost, but for the purpose of discussion we will consider a more depth-first approach. Link SD is unobstructed, so the goal link is generated. This link is in turn obstructed by wall FH. Note that the successor H is to the right of the parent link SD, and that SD is avoiding wall CD on the right. This implies that H may be reached from the parent of D (in this case the start point S) and is guaranteed to avoid CD on the right, so the link SH is generated instead of DH. In practice the successor generator function will trace back perhaps several generations to find the oldest ancestor that does not satisfy the "parent-backup" condition, thus selecting the parent with the shortest path to the successor that will avoid on the same side those same walls avoided by the intervening links. Of course, the new link from the backed-up parent is not guaranteed to be unobstructed, just that it will not hit those walls avoided by the intervening links. If the shortest path to the successor lies on the opposite side of one of the intervening walls, it would be found by the normal search process proceeding from the nodes on the opposite ends of the walls.

Continuing with the example, the new link SH is obstructed by wall IJ. The links SJ and SI are generated as usual. However, note that link SI hits the wall near D again. This repetition is detected by an associative

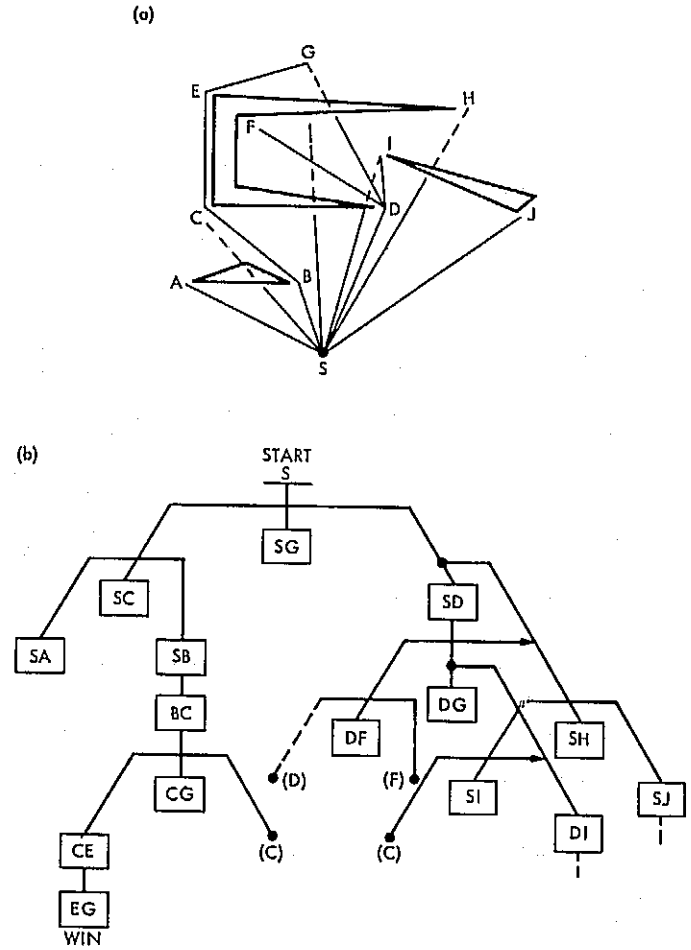



Fig. 4. Path Search Examples (Sheet 1 of 2)

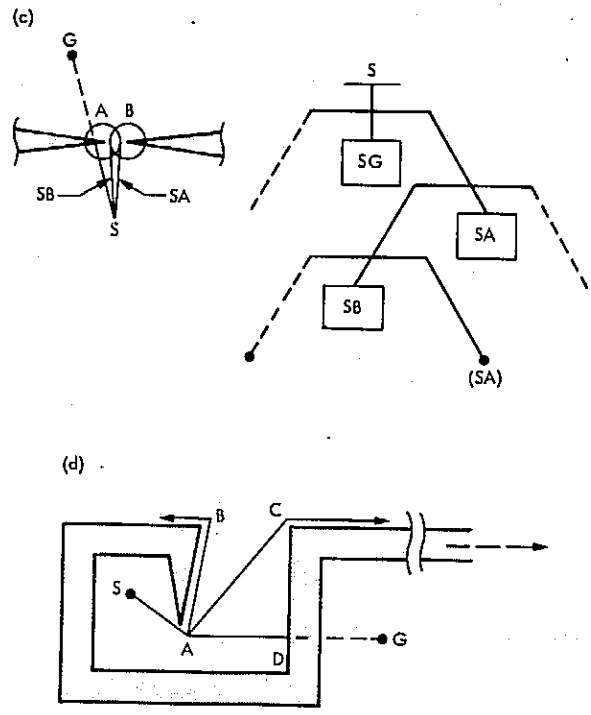



Fig. 4. Path Search Examples (Sheet 2 of 2)

mechanism (discussed below), and since SD was previously found to be successful, the link DI may be generated at once.

Returning to consider link SC, other features of the algorithm may be shown. SC fails, suggesting SB as a candidate avoiding B on the right. SB succeeds, but note that now the goal is on the right of the line containing SB. This state would normally indicate parent backup, but since the goal link from any parent would have already been considered, the destination of the failed link that engendered SB is proposed, in this case C, so BC is generated to avoid CD on the left. This remembering of subgoals is accomplished by associating with the successor links the link whose failure led to their generation. In this case, the failed link SC is associated with both links SA and SB. Note that SB could be obstructed as well, and new links from S would be engendered with B as the subgoal. Such an occurrence would represent the "pushing" of a new level of subgoal onto an implied "stack". In general, once a node is successfully reached and if the successor link is generated to a subgoal (instead of the goal), the subgoal of the link associated with the successful link is then passed along (associated) to the successor, i.e., if the successor is a subgoal, it inherits the subgoal of that subgoal. This represents a "pop" of the implied subgoal stack.

Pruning the Search Space

One of the advantages of performing optimal cost-directed search is that the first path found to a node is the optimum (c.f. HNR1). This allows a node marked as having been successfully visited to be used for pruning the search. No different path to that node need be considered later in the search. This eliminates the need for a graph search process in which a lower cost to a node may be discovered later in the search, requiring updating of all successor

node costs. Thus, for example, when the goal link from A hits wall CD in the figure, neither AC nor AD should be generated. Pruning is indicated in the tree of Fig. 4b by a dot in place of the successor. The requirement for barriers to be closed polygons is dictated by this pruning consideration, since if the same vertex could be reached from both sides of a barrier it would be necessary, when testing a candidate for pruning, to determine if the candidate is on the same side of the wall as the successful link. That would not always be a simple test, so considerable time is saved by the requirement that barriers have "thickness".

The other category of pruning deals with the detection of duplicate links with the same parent, which can occur as a consequence of repetitive failure configurations, or due to parent backup (as shown above), or in cases where the search originates within a concave barrier. Whenever a new link is to be proposed, the destination of the link is compared with that of every other link proposed from the parent. The parent-successor associations are used to derive this set. If no match is found, the proposed link may be generated. However, if the link had been previously generated (i.e., a match is found), the link could be either unobstructed, obstructed, or not yet tested. For each of these cases, action is taken that results in the generation of the appropriate link required to guarantee continuation of the optimal search. Required subgoals may be associated with untested nodes, or, as in the example, the tree below the parent may be examined for the proper node from which to generate a link to the subgoal. Also, the repetition detection will recognize those barrier configurations in which a gap is too narrow for a finite-sized vehicle. In Fig. 4c the circles around the vertices A and B indicate the radius by which the (finite-sized) vehicle must avoid the corner. When link SA attempts to avoid A on the right it encounters the wall at B. Then when SB

attempts to avoid B on the left, A is encountered again, but this is detected, and since SA engendered SB, the repetition is suppressed, effectively treating the gap as closed.

Maze-Solving

Another case that requires special treatment is that of maze-solving, where wall-following is needed if the shortest (or even the only) path is to be found. In Fig. 4d the starting point is contained within a concave polygon. From the vertex B, the goal link is generated, but is obstructed by wall CD. Since there is no path to the right of D, the search would proceed to the left of C, perhaps indefinitely, if there were no other rule. However, as mentioned in the definition of the search space, the walls connected to vertex B are contained in the link set of B, and in this case the shortest path is along the wall AB. It should be noted that unless an adjacent wall is encountered by the normal search that always proceeds toward the goal, wall-following is needed only if it becomes necessary to circle back around the starting point of the search. This allows the normal heuristic estimate of the remaining distance to the goal (used in computing the node cost that determines the order of testing) to be increased by the straight-line distance from the start to the node in question, since that is a lower bound on the path length back around the enclosing obstruction. This increase in total node cost results in fewer unnecessary tests. Thus, when a new successful node is found, its total cost is increased by the defined amount and then reinserted in the list of untested nodes as a candidate for wall-following which would not be tested unless the observed search cost reached its new cost estimate.

Using these rules, the search will continue until a terminating state is reached. If a successful goal link is found, or if an obstructing wall is the border of an unknown region, normal termination occurs. If the goal is enclosed within a barrier, or if the list of untested nodes is exhausted due to repetition pruning, the goal is declared inaccessible. Also, the search could run out of memory, in which case, the path to the successful node nearest the goal is returned.

Real-World Considerations

It is useful for the path planner to conduct the path search in accordance with the actual vehicle size and maneuvering constraints. The size and shape of the vehicle must of course be considered in detecting collisions, and modelling the vehicle turning capability during path search eliminates the need for adjusting and retesting the planned path. Modelling the vehicle's turning geometry is easily done in the path link generator by storing in each link record the center of a turning circle at (or near) the subgoal and the straight line path that is the tangent between that circle and the turning circle at the parent link endpoint (Fig. 5a). A link is then defined as a turn from the parent's endpoint to the link heading followed by a straight traverse ending at the tangent point of the subgoal turning circle. The sign of a turn, indicating left or right, is dictated by which side of an obstructing edge the link is avoiding, i.e., if the parent link ends on the left of an edge, its successors will begin with a right turn about the vertex, etc. The turning center at a vertex need not be located on the vertex. As shown in Fig. 5b, the tangent is found between the turning circle at the parent node and the avoidance circle (of radius $r_a$) centered on the vertex. The solution is obtained by solving the geometrically equivalent problem for the right triangle shown, where

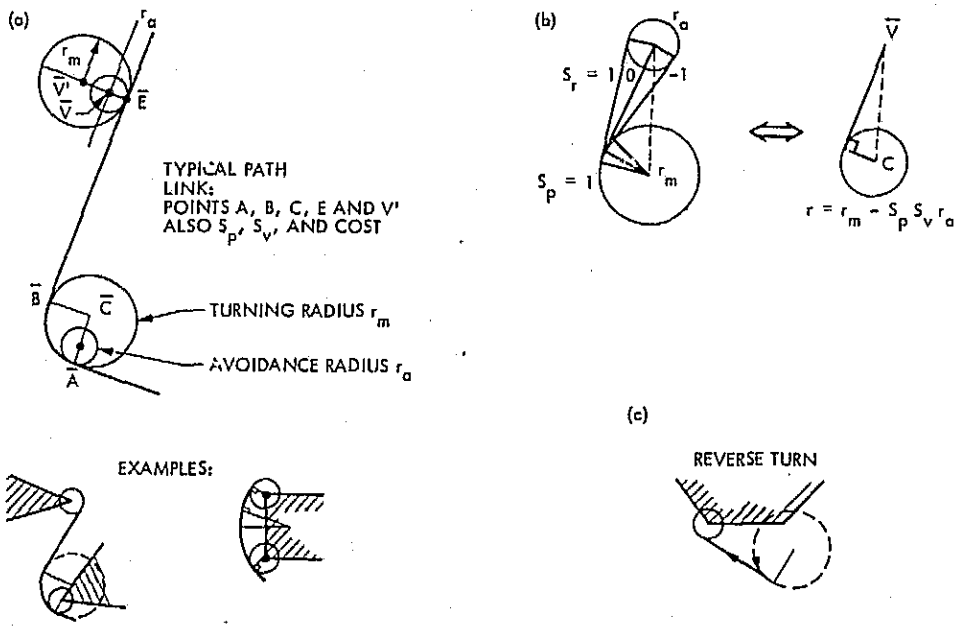



Fig. 5. Path Link Examples

$S_p$ is the sign of the turn at the parent node and $S_v$ is the desired side of the destination vertex. ($S_v$=0 indicates $r_a$=0.) The endpoint and direction of the link then determine the location of the new turning center near the vertex. Turns in reverse may be represented for those situations where a shorter path may be obtained by backing up (initial or terminal heading constraints) or where normal forward movement is restricted (Fig. 5c). The vehicle's length and width must also be considered by the link-testing procedure which searches the map for the first obstructing wall (if any) encountered by either the turn toward the link heading or the straight part of the link. The testing procedure is discussed in Appendix 2.

PATH* also has several special purpose move generators for such cases as confined spaces requiring complicated maneuvers or for special goal categories. A goal may be specified as a requirement that the robot's manipulator be positioned near enough to a selected object to reach it, etc. Also, goals near an edge may impose heading constraints on the vehicle at the goal. The use of tree search methods and state-space representation by node records is an improvement over recursive reduction in this domain. In fact, examples may be constructed in which the success of the search depends upon the ability of PATH* to abandon attempts to reach an inaccessible subgoal in favor of proceeding toward the goal directly from some intermediate node. A recursive algorithm that required reaching the subgoal would fail. In PATH*, the search tree and the other associations become a data base for a variety of operators with all levels accessible through the parent-successor and other relations.

## 5. Planned Path Execution and Error Recovery

Upon command to execute a planned path, NEX invokes the VGM and sends it the path links in succession. The VGM has a system of feedback control

loops for translating the movement commands sent by NEX into vehicle steering and drive signals. Vehicle odometer and gyro-compass heading feedback is used to maintain an estimate of the vehicle's current location, which is then used in the guidance loops to keep the vehicle on the planned path. Front and rear wheels steer in opposite directions, placing the turning radius through the vehicle center. It is desirable to make heading changes without stopping the vehicle, and since the rolling turn is not a circle due to the finite steering rate, this creates a systematic tracking error. The path planner requires a clearance along the planned path that is actually larger than the vehicle width, so that this error is tolerable.

The vehicle will be equipped with proximity and tilt sensors and already possesses a scanning laser rangefinder to aid in the detection of unexpected obstacles. Limited evasive maneuvers by the VGM are allowed, but if avoidance of the obstructing region requires substantial deviation from the planned route, the path is aborted, and error recovery procedures are invoked.

There are numerous error sources having direct impact on the robot's performance. The uncertainty in vehicle position as determined by dead reckoning grows with distance from a known location and may be reduced only by external references such as landmarks. The sensory limitations of the vision system result in uncertainty in the relative position of terrain features which at present are added to the map by using the vehicle's location as an absolute reference point, thus increasing error. Terrain classifications are themselves probabilistic in nature, and in an unstructured environment, mistakes will be made. The end result is that the robot will eventually encounter a rock it never saw, and update the map by remembering the rock in the wrong location relative to a lost robot! After an intervening sojourn it

may even repeat the process on the same rock. Of course, laser and proximity sensors should prevent actual collision, but both the position and map errors remain.

Landmark navigation, when perfected, will allow the system to reduce the robot's positional uncertainty below some upper bound. The map updating process provides another opportunity for error reduction. Knowing the sensory uncertainties, the position of perceived terrain features, and given the locations of previously detected features, the new perspective may be matched against the old by varying the estimated vehicle position (within the error bounds) until the best fit is found. Data structures have been proposed (M1) that record a robot's perceptual history and associated uncertainties, so as to facilitate such a process.

Until such features are implemented, error recovery will consist of a simple map update from the current estimated position, followed by execution of a replanned path.

## 6. Future Work

It is expected that the navigation capabilities of the robot will be expanded in the following areas, more or less in order:

1. In confined quarters it is necessary for the path planner to generate moves that simultaneously avoid several obstacles and that make heading changes by a combination of forward and reverse turns or movements. In some situations reverse search is useful. Such features will be provided either as an improvement to the current algorithm or else be integrated with the general problem solver.

2. A landmark location function, to be provided as part of the vision system, will be used to reduce position uncertainty either when error estimates in the dead-reckoning position exceed a maximum or else continuously by landmark tracking feedback during vehicle motion. Similarly, real-time visual feedback would be used to assist obstacle avoidance.
3. The terrain model will be expanded to categorize areas by slope, texture, etc. Objects may be given functional properties such as "pushable," or "fuel-source," etc. to be used in conjunction with "high-level" planning. Previously executed paths may be remembered, forming a sort of "road map."

## Appendix 1. Map Update Processing

When a map sector update is transmitted to NEX by the vision system, the terrain area boundaries within the sector are described by lists of unit vectors. The unit vectors used are along the basis vectors of the terrain coordinate system, i.e., $\pm \underline{e}_x$, $\pm \underline{e}_y$. The transformation from the unit vector list to the linked vertex list is done in 3 steps (Fig. 6):

1. Consecutive identical unit vectors are replaced by their sum.
2. Inaccessible interior "bubbles" are closed off.
3. The actual polygon approximation is performed.

Interior bubbles are detected by forming partial sums of the chain from one vertex to another. If the length of the resultant vector is less than the vehicle width, the gap is closed off, providing that the intervening path along the chain proves to be contained by the rest of the chain.

The polygon approximation is a successive approximation loop in which the initial fit of two vertices is expanded to improve the fit (F1). The point on the actual boundary with maximum variance from the edge between two vertices in the approximation is inserted in the list, defining two new edges. The test is repeated on each new edge until the observed maximum variance reduces below a threshold.

The resulting list of vectors is then summed successively to produce the absolute coordinates of the vertices. The list is composed as a record and filed.

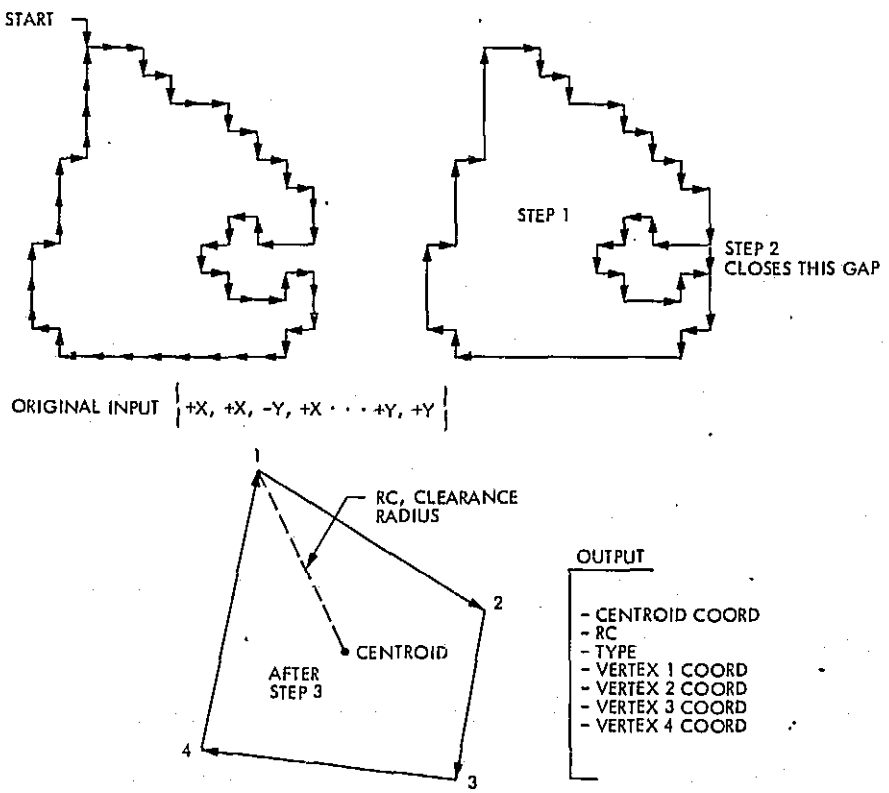



Fig. 6. Map Input Processing

## Appendix 2. Collision Testing in the Terrain Map

The vehicle is approximated as a rectangle, with variables VL2 and RS representing half the length and width respectively. The value RS defines a "safety radius" on either side of the path. The actual coordinates of path endpoints, etc. contained in a path link record refer to the position of the vehicle center point, so that actual collision testing is performed relative to that point. The test consists of detecting whether any edge of an obstacle intersects the area swept out by the vehicle along the path (Fig. 7). Turns are tested by testing successive .5 radian chord lines until the turn is completed.

When a path line is to be tested, the endpoint is extended VL2 units along the path and used to locate the vehicle front edge line as shown. Collision with an obstacle is detected if either of the following tests is true:

a) Any vertex is $\leq$ RS units from the extended path-line.

b) Any edge intersects either the path line or the vehicle front edge.

The nearest collision to the start of the path is found, and the left and right (of path) vertices of the obstructing edge are offered as possible subgoals.

It is useful to limit the search to only those obstacles that lie near the path line to avoid performing the detailed search of every obstacle in the map. Included in the obstacle record is a centroid and clearance radius (RC, the radius of the superscribed circle about the centroid). Barriers within the map sectors that contain the path are tested to see if the distance from the centroid to the path-line is $\leq$ RC + VL2. If so, then the detailed test is performed.

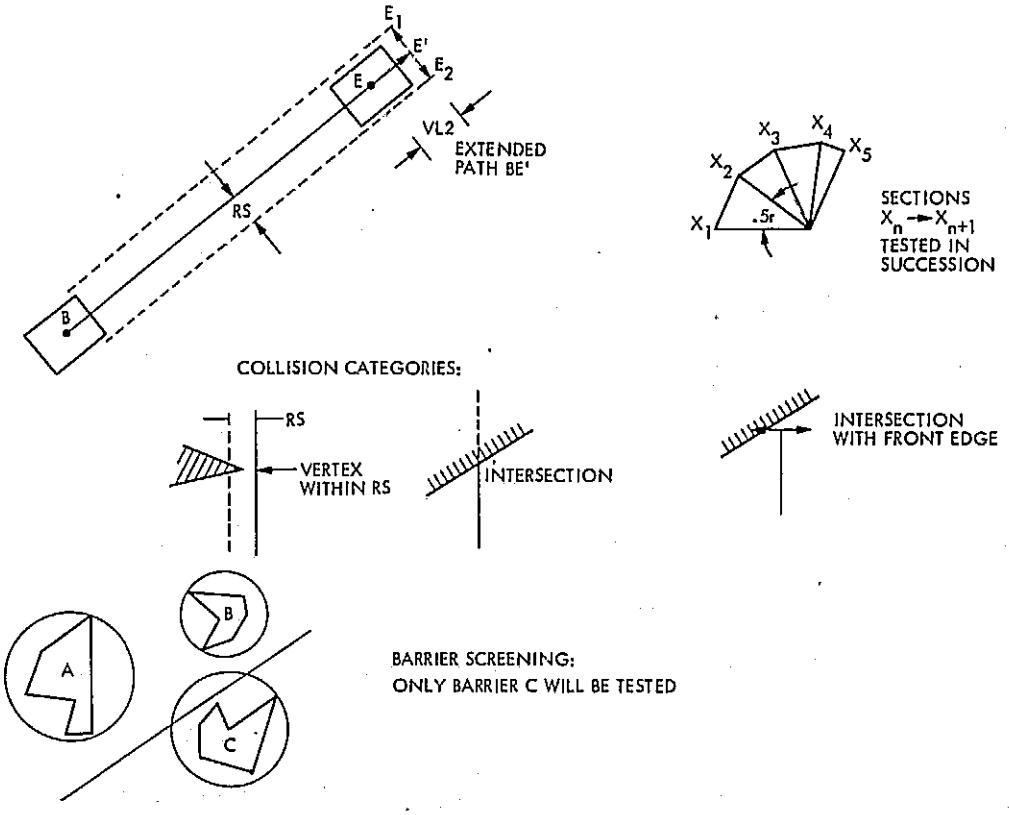



Fig. 7. Collision Testing

# Appendix 3. Program Structure

## I. NEX Structure

With the exception of the polygon approximation process (Appendix 1), NEX is primarily a command decoder and sequencer that accepts inputs from REX, EYE, PPM, and VGM, translating them into either commands to another subsystem or to status information. The NEX process is in a suspended (idle) state until input is ready, and it returns to this state after the message is processed. Inputs may be either commands or responses from the other modules. Commands are strings of ASCII characters, and responses may be either strings or binary array messages. Commands to other systems that require a response are assigned an integer message number and are placed in a queue of active messages until the corresponding response is received. NEX commands may generate multiple subsystem commands, as in GOTO or DOPATH.

NEX Commands:

ABORT - terminate current processes and return to idle.

ASKTV(msgn,mapx,mapy,time-last-seen)
- initiates a map sector request for (mapx,mapy); may come from either REX or PPM.

DIE - performs ABORT, then logs out PPM and NEX.

DOPATH(name) - retrieves command sequence for named (previously planned) path and sends execution commands to VGM.

GOAL(<destination specifier*>)
- response to data-base query generated by "ASK" option.

---

*<destination specifier> may be X,Y; X,Y,<options>; or ASK (ask database for goal). <options> may be 0 (on the point, same as X,Y), P (pickup:position the arm work area at X,Y), or θ (degrees, required heading at destination X,Y).

GOTO(<dest. spec.>)

- performs successive PATHTO and DOPATH commands until goal is reached or failure is detected.

NEWMAP(msgn, mapname, time-seen [,mapx,mapy])

- adds map sector to PPM's map directory, replacing any previous map; may occur in response to ASKTV or may be unsolicited. (Also PPM command.)

NEW!LOC(X,Y,$\alpha$)

- initializes vehicle position to X,Y and defines X-axis as $\alpha$ degrees from North; - requires response from VGM telling vehicle heading in the new frame.

NOMAP(msgn), OLDMAP(msgn)

- response to ASKTV, informing PPM to use no map or the old map for the specified sector.

PATH(name, $X_o$,$Y_o$,$\theta_o$,<DEST. SPEC.>)

PATHTO(name, <dest. spec.>)

- plan a path from $X_o$,$Y_o$,$\theta_o$ (or present location, PATHO) ending at specified destination, store under the name given.

RESUME - resume interrupted path

SPEED!LIMIT(V)

- Set vehicle speed limit

STATUS - Report status of NEX, PPM, and vehicle.

STOP!VEH - interrupt the execution of a path; may be resumed.

WAITON(process)

- NEX waits until specified process is finished. Process may be SPC, PPM, or query.

NEX Inputs from PPM

In response to a PATH command, PPM sends ASCII messages of the form <code>; <message> to NEX. The codes are:

1. success
2. inaccessible goal
3. encountered unknown
4. search capacity exceeded
5. warning messages
6. map sector request; message is an ASKTV command to NEX.

NEX Inputs from VGM

In response to commands from NEX, if a completion response is requested, (by the presence of a nonzero message number) an ASCII message of the form: msgn, completion status, vehicle position $(x,y,\theta)$ is returned.

NEX Inputs from EYE

In response to map sector requests, EYE will return binary array messages containing either chain code describing boundaries within the sector or a request-completion message. The messages are of variable length, with a minimum of two words, the message number of the original sector request, and the type of message. Type 0 messages indicate the end of chain code record transmission for the sector, and type <0 indicate that the sector contains no boundaries, but is entirely of type |type|. If type >0 the remaining words of the message describe a boundary of the specified type, giving in successive words: $X_o$, $Y_o$ (the start of the boundary in abs. coords.), the number of steps of chain code followed by that many two-bit chain code cells packed right-justified, eight per word. Implemented boundary-type codes are:

1 - nontraversible, 2 - unknown, 6 - field of view. Chain codes 0-3 indicate steps in the -y, +x, -x, +y directions respectively, of length determined by the system parameter USCALE (normally 15 cm). After smoothing by the POLYGON algorithm (Appendix 1), boundary records are stored on disk for subsequent use by the PPM.

NEX Outputs to PPM

PATH and PATHTO commands to NEX cause a PATH command to be sent to PPM. NEWMAP, OLDMAP, and NOMAP commands are forwarded to PPM (with the msgn assigned by PPM to the original sector request substituted), or are generated upon completion of a sector request by EYE.*

NEX Outputs to VGM

VGM commands are ASCII messages of the form msgn, command-code [, arguments]. The codes (and arguments) are:

VGM idle = 0

STOP!VEH = 1

RESUME (from stop) = 2

NEW!LOC = 3, X, Y, $\alpha$

SPEED!LIMIT = 4, v (cm/sec)

execute path link = 10, $\pm$ radius, $X_a$, $Y_a$, $X_b$, $Y_b$, $X_c$, $Y_c$, w

- minus <0 indicates reverse turn; subscripts a, b, and c are for start of turn, end of turn, and end of link, respectively.

*The decision to send queries for map sectors to the operator or to EYE is determined by the value of an internal variable MAPOPT, set by the OPTIONS command to NEX. Default is EYE.

NEX Outputs to EYE

During path planning, NEX sends ASCII commands to EYE: (MAPOPT = 0)

MAPSTART (vlocx, vlocy, θ)

- start-up message

MAPSEC (msgn, mapx,lmapy)

- sector request generated by ASKTV command.

MAPOUT - end of planning

II. PPM and VGM

A block diagram of PPM structure is provided in Fig. 8. For a discussion, see main text; VGM structure is described briefly in the text.

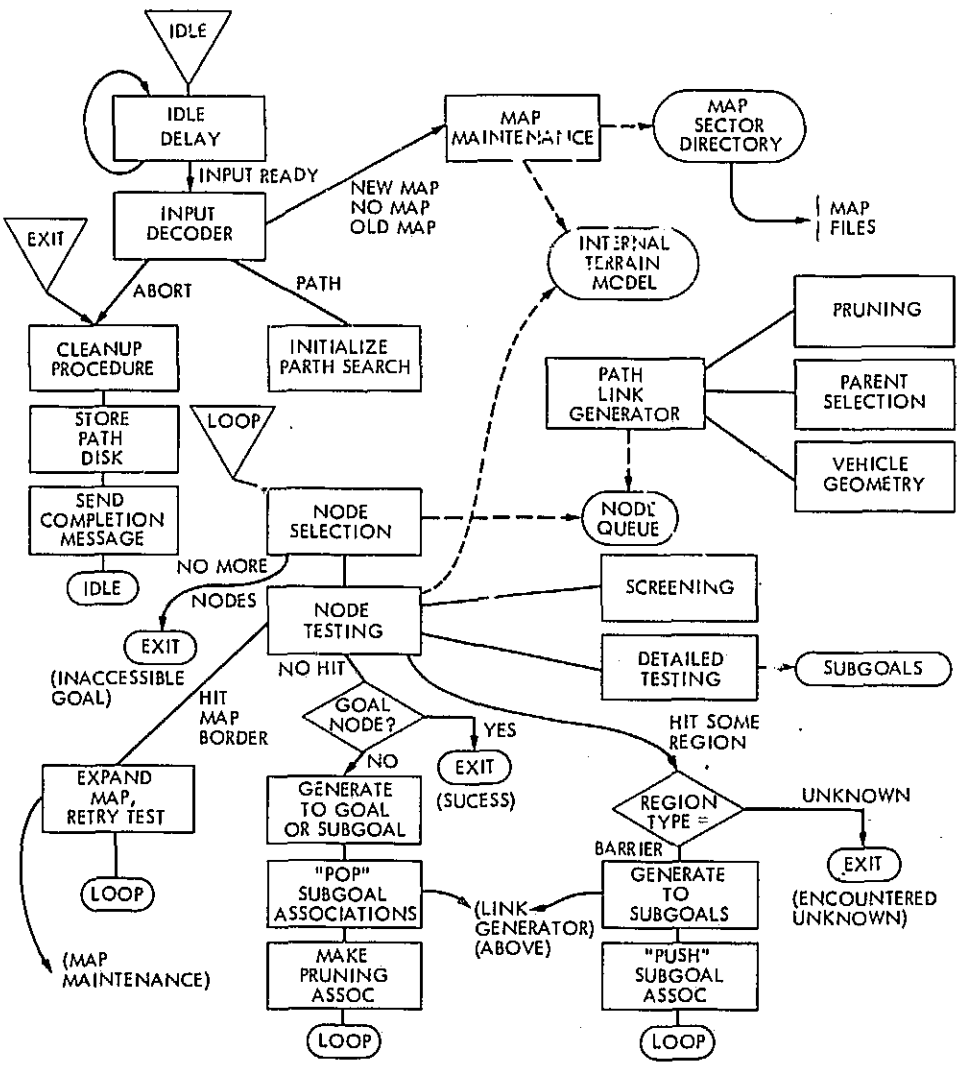



Fig. 8. PPM Structure

REFERENCES

F1. Freeman, H., "Computer Processing of Line-Drawing Images," Computing Surveys, Vol. 6, No. 1, March 1974.

FS1. Feldman, J.A., and Sproull, R.F., "System Support for the Stanford Hand-Eye System." Proceedings of the 2nd International Joint Conference on Artificial Intelligence, London, England, 1971.

HNR1. Hart, P.E., Nilsson, N.J., and Raphael, R., "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," IEEE Transactions on Systems Science and Cybernetics, July 1968.

M1. Merriam, E.W., Robot Computer Problem Solving System. Bolt Beranek and Newman, Inc., Cambridge, Mass. (Progress Report in publication).

R1. Raphael, et al., Research and Applications - Artificial Intelligence, Report on SRI Project 8973. Stanford Research Institute, Menlo Park, Calif., Dec. 1971.

VL1. Van Lehn, K.A. (editor), SAIL User Manual, Stanford Artificial Intelligence Laboratory Memo AIM-204. Stanford University, Stanford, Calif., July 1973.

YC1. Yakimovsky, Y., and Cunningham, R., DABI - Data Base for Image Analysis with Nondeterministic Inference Capability, Technical Memorandum 33-773. Jet Propulsion Laboratory, Pasadena, Calif., Feb. 1976. (Also printed in Pattern Recognition and Artificial Intelligence, C.H. Chen, editor Academic Press, New York, 1976.

YC2. Yakimovsky, Y., and Cunningham, R., A System for Extracting 3-D Measurements from a Stereo Pair of TV Cameras, Technical Memorandum 33-769. Jet Propulsion Laboratory, Pasadena, Calif., March 1976.



NASA—JPL—Coml., L.A., Calif.