Correspondence

Path Planning for a Mobile Robot

Christos Alexopoulos and Paul M. Griffin

Abstract— **Two problems for path planning of a mobile robot are considered. The first problem is to find a shortest-time, collision-free path for the robot in the presence of stationary obstacles in two dimensions. The second problem is to determine a collision-free path (greedy in time) for a mobile robot in an environment of moving obstacles. The environment is modeled in space-time and the collision-free path is found by a variation of the A^* algorithm.**

I. Introduction

Recently, a great deal of research has been done in the area of motion planning for a mobile robot. The motion planning problem is the determination of a path for a mobile robot from its current position to a goal position through an environment of obstacles. The obstacles may either be stationary or moving. The desired path is a shortest-time path where there are no collisions between the robot and the obstacles. If the robot has constant speed, and instantaneous acceleration and deceleration, the path planning problem reduces to finding a shortest collision-free path.

Much of the prior work in robot path planning has been for stationary obstacles [1]–[3], [5], [6], [9], [11], [12], [15], [16]. A necessary condition in this case is that the path be made up of straight line segments connecting a subset of vertices V of the obstacles [10], [12]. An algorithm based on this necessary condition has been developed and is called the visibility graph algorithm, or VGRAPH [12]. In this algorithm, a visibility graph G composed of the starting position, s, the goal position, t, the vertex set V, and the edge set E that contains an edge for each pair of visible vertices is constructed. A shortest path from s to t is determined by Dijkstra's method [7]. Other methods of motion planning partition the space into some type of smaller space by triangulation, convex splitting or generalized cones [1], [3], [5]. Traversals are then determined between these smaller regions from s to t.

In the case where the obstacles can move, the problem becomes much more difficult. Canny and Reif [4] showed that motion planning for a point in the plane with velocity limits in the presence of moving obstacles is NP-hard, making the existence of polynomial algorithms unlikely. Kant and Zucker [10] developed a heuristic that decomposes the problem into two subproblems. The first subproblem determines a shortest path among the stationary obstacles, ignoring the moving obstacles. The second subproblem determines a velocity for the robot along the path to prevent collisions with the moving obstacles. Reif and Sharir [14] developed an algorithm that determines the time intervals for which the robot can make feasible movements, and used these intervals to find a collision-free path. The complexity of the algorithm, however, is exponential in the number of obstacles. Fujimura and Samet [8] included time as a dimension in the model, and then discretized the space using a quadtree-type hierarchical structure. A collision-free path was then determined in space-time. The run time of the algorithm, however, is exponential in the number of nodes in the input.

The purpose of this paper is threefold: 1) to present an algorithm for the determination of a shortest, collision-free path of a mobile robot in the presence of stationary obstacles, 2) to extend this algorithm to find a collision-free path (greedy in time) for the case where the obstacles move along linear paths with constant velocities, and 3) to illustrate the proposed procedures by examples. Section II presents the algorithm for motion planning in a stationary environment. Section III extends the algorithm in Section II to the case of moving obstacles. The environment of moving obstacles is described by a space-time representation, and a collision-free path is determined by a variation of the A^* algorithm [13]. Concluding remarks are discussed in Section IV.

The following assumptions are made in this paper:

1) Obstacles are simple polygons with vertices in the set N. Let $V = N \cup \{s,t\}$.
2) The obstacles are expanded by the size of the robot. Therefore, the robot can be considered to be a point moving among the expanded obstacles [8], [11], [14], [16].
3) The nonstationary obstacles move along linear paths with constant velocities.
4) The mobile robot has constant speed, and instantaneous acceleration and deceleration.

II. Stationary Obstacle Environment

In this section, we assume that the obstacles are stationary. As mentioned previously, one method for solving this problem is by VGRAPH [12]. The worst-case complexity of VGRAPH presented in [12] is $O(n^2 \cdot \log\ n)$. Asano *et al.* [2] and Welzl [17] independently developed an algorithm that finds the visibility graph in $O(n^2)$ time. The shortest collision-free path problem can be solved in this case in $O(n^2)$ time using Dijkstra's method [7]. This section presents a variation of VGRAPH that we call V^*GRAPH. V^*GRAPH can greatly reduce the number of vertices of the visibility graph that are visited and forms the basis for the algorithm presented in the Section III for the case of moving obstacles.

The idea behind V^*GRAPH is to only expand a subset of the visible nodes expanded by VGRAPH. The path from s to t is found by using the A^* algorithm in Nilsson [13]. This algorithm is an admissible search strategy guided by an evaluation function f. The value $f(n)$ at any node n is an estimate of the shortest-path-length from s to t through n. Specifically,

$$f(n) = g(n) + h(n)$$

where $g(n)$ is the shortest-path-length from s to n and $h(n)$ is an estimate of the shortest-path-length from n to t. If $h(n)$ is a lower bound on the actual shortest-path-length from n to t for all n, then the A^* algorithm is guaranteed to terminate with an optimal path from s to t, if such a path exists. This is called the admissibility property of the A^* algorithm.

The heuristic function $h(n)$ used in V^*GRAPH is the Euclidean distance from n to t. Obviously this is a lower bound on the actual distance along any path from n to t, since it assumes that there are no obstacles between these two points. For this reason, the admissibility property is satisfied.

Manuscript received September 24, 1989; revised February 24, 1991 and July 28, 1991.

The authors are with the School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA 30332-0205.

IEEE Log Number 9104542.

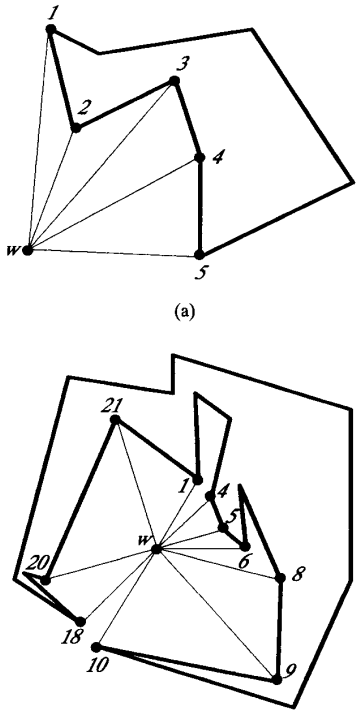

Fig. 1. Two examples of w—visible sequences on an obstacle.

Let w be a point in $\Re^2$. The set of vertices in V that are visible from w are called the *w-visible vertices*. A set of consecutive w-visible vertices on a single obstacle is called a *w-visible sequence*. An obstacle vertex is called an *acute vertex* if it is the vertex of an acute interior polygon angle, otherwise it is called an *obtuse vertex*. For a pair of points $(x, y) \in \Re^2$, $d(x,y)$ denotes the Euclidean distance between x and y.

The advantage of V^*GRAPH over VGRAPH is that, on the average, less nodes need to be considered. This is shown in the following two theorems. Theorem 1 states that from a current position w, only a subset of the w-visible vertices must be considered. This subset contains the endpoints of the w-visible sequences. Theorem 2 states that only acute obstacle vertices need to be considered.

Theorem 1: From a current position w, only the extreme vertices on a w-visible sequence need to be considered.

Proof: Let S denote a w-visible sequence on an obstacle O. If S contains one or two vertices, then those vertices are extreme vertices of S. Alternatively, suppose that S contains at least three vertices and let x and z be the extreme vertices of S. If the line segment $\overline{wt}$ intersects O, then every shortest collision-free path that goes through w must contain either $\overline{wx}$ or $\overline{wz}$. Otherwise, it contains a polygonal line with the same endpoints as one of these line segments and therefore is not shortest. □

Fig. 1 shows two examples of w-visible sequences. Note that in Fig. 1(a), all visible vertices 1, 2, 3, 4, and 5 form a w-visible sequence and so from Theorem 1, the only vertices that need to be considered are the extreme vertices 1 and 5. In Fig. 1(b), there are four visible sequences, namely $\{4,5,6\}, \{8,9,10\}, \{18\}$ and $\{20,21,1\}$. Therefore the only vertices that need to be considered are the endpoints of these sequences 4, 6, 8, 10, 18, 20, and 1.

Determining the extreme points of the visible sequences can be simplified if we do some prepossessing on the obstacles. Specifically, if the vertices are numbered as they occur around a tour of a polygon, then sorting on the vertex numbers can be used to determine the visible sequences and their extreme vertices. We assume that the obstacle vertices are numbered in this fashion. This requires $O(n)$ steps, and only has to be done one time at the original obstacle input. The following procedure may be used for finding the extreme points of the w-visible sequences on an obstacle O.

Procedure to Find Extreme Points of w-visible Sequences on an Obstacle O:

```
S = set of w-visible vertices on O;
T = sorted set S;
j = 1;
b_j = first element in T;
For = v_i = (second element in T) to (last element
in T) do
  if = (v_i = v_{i-1} + 1) then
    e_j = v_i;
  else
j = j + 1;
b_j = v_i;
if = ((b_1 = 1) and (e_j = n)) then
  b_1 = b_j;
  j = j - 1.
```

The output from the procedure listed above is the extreme vertices of the j w-visible sequences. Each sequence j has a pair of extreme vertices (b_j, e_j) unless the sequence contains a single vertex. In the latter case, sequence j has only one extreme vertex b_j.

Theorem 2: A shortest, collision-free path from s to t cannot contain obtuse obstacle vertices.

Proof: Suppose that a shortest path P contains an obtuse vertex y on an obstacle O. Let x and z denote the predecessor and successor nodes of y on P. Since O is simple, then there is a point w on the line segment $\overline{yz}$, other than y and z, which is visible from x. The triangle inequality

$$d(x,w) < d(x,y) + d(y,w)$$

yields

$$d(x,w)+d(w,z) < d(x,y)+d(y,w)+d(w,z) = d(x,y)+d(y,z).$$

This implies that the path obtained from P by replacing the vertex y by the point w is shorter than P, a contradiction. □

Fig. 2 shows a polygon made up of 10 vertices. From Theorem 2, only the vertices 1, 2, 6, 7, 8, 10, and 11 need to be considered. Of course Theorem 1 and Theorem 2 may be used together. For instance, if we first apply Theorem 1 to Fig. 1(b) the set of vertices to be considered is {4, 6, 8, 10, 18, 20, 1}. Applying Theorem 2 reduces this set to {4, 6, 10, 18, 20, 1}.

As in the case with Theorem 1, we assume that some prepossessing is done to the obstacles given as input. Specifically, we assume that each obstacle is listed as being convex or not. This is done by checking the position of the two adjacent sides of each vertex. To prepossess the entire input requires $O(n)$ steps, and only has to be done for the original obstacle input.

V^*GRAPH makes use of Theorems 1 and 2 to prune the search graph. As vertices are added to the graph, the evaluation function from the A^* algorithm is used to evaluate each node. The V^*GRAPH algorithm is listed below.

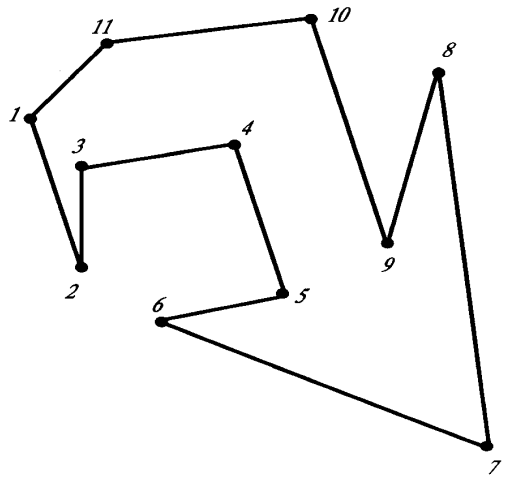

Fig. 2. Example obstacle in which from Theorem 2, only vertices 1, 2, 6, 7, 8, 10, and 11 need to be considered.

V^*GRAPH Algorithm

INPUT: Initial robot position, s, goal position, t, and obstacles
OUTPUT: Shortest collision-free path P from s to t

```
begin
  P := {s};
  Open := {s};
  g(s) := 0;
  f(s) := 0;
  repeat
      w := {i ∈ Open : f(i) ≤ f(j), ∀j ∈ Open};
      remove w from Open;
      put w in P;
      VV := {w-visible vertices not in Open};
      EV := {extreme vertices of the w-visible paths in VV};
      AV := EV− {obtuse vertices in EV};
      for = each vertex i in AV do
        h(i) := Euclidean distance d(i,t) from i to t;
        g(i) := g(w) + d(w,i);
        f(i) := g(i) + h(i);
        put i in Open;
  until ((Open = ∅) or (w = t));
if = (Open = ∅) then
    exit with failure;
  if = (w = t) then
    exit with P;
end.
```

Theorem 3: V^*GRAPH finds a shortest collision-free path from s to t in $O(n^2 \cdot \log\ n)$ time and space, where n is the total number of obstacle vertices.

Proof: First note that in the worst case, V^*GRAPH generates the same search graph as VGRAPH. Suppose that there exists a collision-free path P' that is shorter than the path P found by V^*GRAPH. Then there must be some vertex w on P', which is not on P. Otherwise, by the admissibility property of the A^* algorithm, path P would be at least as short as P', a contradiction. Further, w must not have been a visible vertex from any vertex in P. Since w is not visible from any vertex v in P, then the line segment $\overline{vw}$ must pass through an obstacle for each v. Any path from v to w then would involve passing through some vertex that V^*GRAPH discarded either by Theorem 1, Theorem 2 or by the evaluation function of the A^* algorithm. Therefore, any path that included w must be shorter than P, a contradiction.

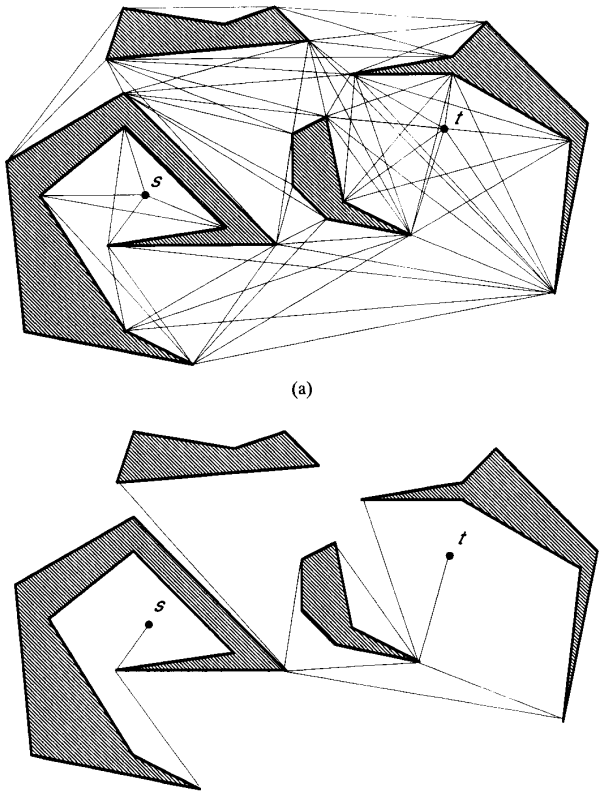

Fig. 3. Stationary obstacle example showing the number of vertices considered in VGRAPH (a) and in V^*GRAPH (b).

The worst-case complexity of V^*GRAPH is determined as follows. There are four steps for each iteration of the algorithm. The first step finds a point w in *Open* with minimum $f(w)$. Since there are at most n points in *Open*, this takes $O(n)$ time. The second step finds the set of visible vertices from the current position. This requires $O(n \cdot \log\ n)$ time [12]. The third step determines the extreme points of the visible paths and, as mentioned earlier, requires $O(n)$ time. The final step removes the obtuse visible vertices in $O(n)$ time. Then each iteration requires $O(n \cdot \log\ n)$ time. Since there will be at most n iterations, the overall worst-case complexity of V^*GRAPH is $O(n^2 \cdot \log\ n)$. □

As mentioned earlier, Asano *et al.* [2] and Welzl [17] independently developed an algorithm to determine the visibility graph in $O(n^2)$ time, and hence by using Dijkstra's method on the visibility graph, the shortest path may be determined in $O(n^2)$ time. V^*GRAPH can be made to run in $O(n^2)$ time by using the entire visibility graph algorithm and would guarantee that the resulting collision-free path was shortest. However, this would mean that the entire visibility graph would have to be constructed. The advantage of V^*GRAPH over algorithms that use the visibility graph is that a lot less nodes are considered on the average. An example is shown in Fig. 3. Fig. 3(a), shows the number of nodes considered by the VGRAPH algorithm while the number considered by V^*GRAPH is shown in Fig. 3(b).

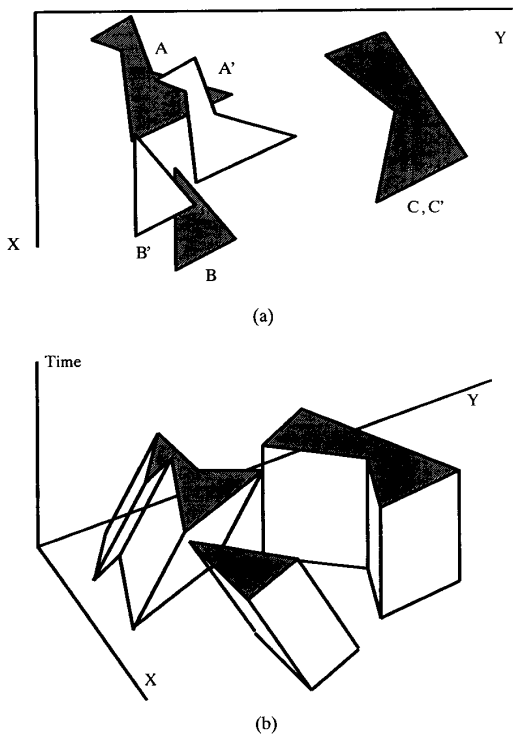

Fig. 4. 2-D environment captured at two points in time (a), and the corresponding environment represented in space-time (b).

III. Moving Obstacle Environment

In this section, we assume that the non stationary obstacles are moving along linear paths with constant velocities. As mentioned previously, the motion planning problem for a mobile robot with velocity constraints (shortest-time, collision-free path) with moving obstacles is NP-hard [4]. We show that the V^*GRAPH algorithm presented in Section II can be extended to find a collision-free path that is greedy in time, but not necessarily optimal.

Since both the obstacles and the mobile robot are assumed to move on the x-y plane, the time dimension is used to define the position of the obstacles. This three-dimensional space is called space-time. Each obstacle defined in space-time forms a prism. An example is shown in Fig. 4. In Fig. 4(a), the obstacles are shown in the x-y plane at two instances in time. The space-time representation of this scene is shown in Fig. 4(b). The stationary obstacle (C) is a prism that is orthogonal to the x-y plane, while the moving obstacles (A and B) are prisms that are sloped.

In order to find a collision-free path from s to t, the three-dimensional space must be searched. In this case, the condition that the shortest collision-free path is made up of straight line segments connecting a subset of the vertices of the obstacles is no longer true. Therefore, VGRAPH cannot be used as a heuristic since the visibility graph changes over time. However, under the assumptions previously given, a simple modification of V^*GRAPH finds a collision-free path greedy in time. This extended version of V^*GRAPH is called the E^*GRAPH algorithm. Like V^*GRAPH, E^*GRAPH also makes use of the A^* algorithm and Theorems 1 and 2. Note that since space-time is a three-dimensional space, Theorems 1 and 2 are applied on a 2-D projection onto the x-y plane.

Since the robot moves with constant velocity v, then from some position w in space-time, the possible new positions that the robot can move to are defined by the surface of a cone C emanating from w. This surface may be generated as follows. Construct a vector $\vec{a}$ from w at angle θ from the plane P through w, which is parallel to the x-y plane, where

$$\theta = \tan^{-1}(v^{-1}).$$

Sweep this vector $\vec{a}$ about w, maintaining constant angle θ from P. Since there are obstacles present (prisms in space-time), we are interested in the intersections of the cone C with the obstacles. We focus on the intersections of the prism edges with C. We call this set the *w – visible points at angle θ* and use it to construct the search graph for E^*GRAPH. These points define *w-visible sequences at angle θ* and correspond to w-visible vertices in the x-y plane.

E^*GRAPH makes use of Theorems 1 and 2 to prune the search graph. As vertices are added to the graph, the evaluation function from the A^* algorithm is used to evaluate each nodes. The E^*GRAPH algorithm is listed here.

E^*GRAPH Algorithm

```
INPUT: Initial robot position, s, goal position,
t, and initial obstacle positions with velocities
OUTPUT: Collision-free path P from s to t (greedy
in time)
begin
  θ := tan^-1(v^-1);
  P := {s};
  Open := {s};
  g(s) := 0;
  f(s) := 0;
repeat
    w := {i ∈ Open : f(i) ≤ f(j)∀j ∈ Open};
    remove w from Open;
    put w in P;
    VV := {w-visible points at angle θ not in Open};
    EV := {extreme points on the w-visible paths
    in VV at angle θ};
    AV := VV− {points in EV corresponding to
    obtuse obstacle vertices};
    for = each vertex i in AV do
      compute the Euclidean distance d(w,i) in
      time-space;
      c(w,i) := d(w,i);
      compute the Euclidean distance d(i,t) in
      time-space;
      h(i) := d(i,t);
      g(i) := g(w) + c(w,i);
      f(i) := g(i) + h(i);
      put i in Open;
  until ((Open = ∅) or (w = t));
  if = (Open = ∅) then
    exit with failure;
  if = (w = t) then
    exit with P;
end
```

The worst-case complexity of E^*GRAPH is determined as follows. There are four steps for each iteration of the algorithm. The first step finds a point w in *Open* with minimum $f(w)$ in $O(n)$ time. The second step finds the set of w-visible points at angle θ in $O(n \cdot \log\ n)$ time. The third step determines the extreme points on the visible sequences at angle θ in $O(n)$ time. The final step removes those visible points that correspond to obtuse obstacle vertices in $O(n)$ time. Then each iteration requires $O(n \cdot \log\ n)$ time. Since there

will be at most n iterations, the overall worst-case complexity of E^*GRAPH is $O(n^2 \cdot \log\ n)$.

The aforementioned algorithm has assumed that the mobile robot moves with constant velocity. This restriction may be relaxed as follows. Approximate the continuum of velocity values that the mobile robot can take on by one of m discrete velocity values in this range. Then apply the algorithm using these m values. This will increase the worst-case complexity to $O(m^2 n^2 \cdot \log\ n)$.

The algorithm has also assumed that obstacles move along linear paths. This assumption may be relaxed to the case of obstacles moving along piecewise linear paths. In this case, each time the obstacle moves in a new direction, the prism in the space-time representation also changes direction. The only change to E^*GRAPH is in the step of finding the visible points of intersection from current position w. If an obstacle has at most c path changes, then at most c intersection points from w to vertex v will need to be considered. The step of finding the set of w-visible points at angle θ requires $O(c \cdot n \cdot \log\ n)$ time. The overall worst-case complexity of E^*GRAPH in this case is $O(c \cdot n^2 \cdot \log\ n)$.

IV. Conclusion

In this paper we have presented two algorithms for mobile robot path planning. The first algorithm called V^*GRAPH finds a shortest-time, collision-free path for a mobile robot in the presence of stationary obstacles. V^*GRAPH owes its advantage over other similar algorithms in its average case performance to the fact that fewer nodes are considered. The second algorithm finds a collision-free path in the presence of moving obstacles that is greedy in time. The environment is modeled in a three dimensional space-time, and the path is found by using a variation of the V^*GRAPH algorithm. The worst-case complexity of both algorithms is $O(n^2 \cdot \log\ n)$.

Acknowledgment

The authors would like to acknowledge Lakshmi Narasimhan and the anonymous referees for their helpful comments and suggestions.

References

[1] R. C. Arkin, "Navigational path planning for a vision-based mobile robot," *Robotica*, vol. 7, pp. 49–63, 1989.

[2] T. Asano, T. Asano, L. Guibas, J. Hershberger, and H. Imai, "Visibility-polygon search and euclidean shortest paths," in *Proc. 26th IEEE Symp. Foundations of Comput. Sci.*, Portland, OR, 1985, pp. 155–164.

[3] R. A. Brooks, "Solving the find path problem by good representation of free space," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-13, pp. 190–197, 1983.

[4] J. Canny and J. Reif, "New lower bound techniques for robot motion planning problems," in *Proc. 28th IEEE Symp. Foundations of Comput. Sci.*, Los Angeles, CA, 1987, pp. 49–60.

[5] J. L. Crowley, "Navigation for an intelligent mobile robot," *IEEE J. Robotics Automat.*, vol. RA-1, pp. 31–41, 1985.

[6] S. K. Dambhampati and L. S. Davis, "Multiresolution path planning for mobile robots," *IEEE Trans. Robotics Automat.*, vol. RA-2, pp. 135–145, 1986.

[7] E. W. Dijkstra, "A note on two problems in connection with graphs," *Numer. Math.*, vol. 1, pp. 269–271, 1959.

[8] K. Fujimura and H. Samet, "A hierarchical strategy for path planning among moving obstacles," *IEEE Trans. Robotics Automat.*, vol. 5, pp. 61–69, 1989.

[9] J. Hershberger and L. J. Guibas, "An $O(n^2)$ shortest path algorithm for a non-rotating convex body," *J. Algorithms*, vol. 9, pp. 18–46, 1988.

[10] K. Kant and S. W. Zucker, "Toward efficient trajectory planning: The path-velocity decomposition," *Int. J. Robotics Res.*, vol. 5, no. 3, pp. 72–89, 1986.

[11] D. T. Lee and F. P. Preparata, "Euclidean shortest paths in the presence of rectilinear barriers," *Networks*, vol. 14, pp. 393–410, 1984.

[12] T. Lozano-Pèrez and M. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Commun. ACM*, vol. 22, no. 10, pp. 560–570, 1979.

[13] N. J. Nilsson, *Principles of Artificial Intelligence*. Palo Alto, CA: Tioga, 1980.

[14] J. Reif and M. Sharir, "Motion planning in the presence of moving obstacles," in *Proc. 26th IEEE Symp. Foundations of Comput. Sci.*, Portland, OR, 1985, pp. 144–154.

[15] M. Sharir and A. Schorr, "On shortest paths in polyhedral spaces," *SIAM J. Comput.*, vol. 15, pp. 193–215, 1986.

[16] J. S. Singh and M. D. Wagh, "Robot path planning using intersecting convex shapes: Analysis and simulation," *IEEE J. Robotics Automat.*, vol. RA-3, pp. 101–108, 1987.

[17] E. Welzl, "Constructing the visibility graph for n line segments in $o(n^2)$ time," *Inform. Process. Lett.*, vol. 20, pp. 167–171, 1985.

A Closed-Form Massively-Parallel Range-From-Image-Flow Algorithm

Daniel Raviv and James S. Albus

***Abstract*—A closed-form solution for obtaining three-dimensional (3-D) structure of a scene for a given six degree of freedom motion of camera is provided. The solution is massively parallel, i.e., the range that corresponds to each pixel is dependent on the spatial and temporal changes in intensities of that pixel, and on the motion parameters of the camera. The measurements of the intensities are done in *a priori* known directions. The solution is for the general case of camera motion. The derivation is based upon representing the image in the spherical coordinate system, although a similar approach could be taken for other image domains, e.g., the planar coordinate system. Comments are made on the amount of computations, error and singular points of the solutions. A practical way to significantly reduce and implement them is suggested.**

I. Introduction

The perception of depth is critical to survival for creatures of nature. Depth perception is necessary for locomotion without collision in a complex natural three-dimensional (3-D) world. It is also advantageous for recognizing objects. In a natural environment, image segmentation based on brightness boundaries of stationary objects is extremely difficult. It is confounded by shadows, surface texture, and markings that often produce brightness and color boundaries that are much more distinct than true object or surface boundaries. In the natural world, segmentation based on motion discontinuities is much more reliable than segmentation based on brightness discontinuities.

Many methods have been proposed for computing depth from visual input. Of these, only the methods of stereo and image flow can compute depth directly without any knowledge of, or understanding about, the contents of the image. Only image flow can compute depth from a single retina without simultaneously viewing the scene from two separate eyes. Depth from image flow is thus extremely important to prey animals, such as fish, most birds, and rabbits because their

Manuscript received July 28, 1990; revised July 9, 1991.

D. Raviv is with the Robotics Center and Electrical Engineering Department, Florida Atlantic University, Boca Raton, FL 33431.

J. S. Albus is with the Robot Systems Division, National Institute of Standards and Technology, Gaithersburg, MD 20899.

IEEE Log Number 9104541.