MVA '92 IAPR Workshop on Machine Vision Applications Dec. 7-9,1992, Tokyo

A Complete Navigation System for a Mobile Robot, Using Real-time Stereovision and the Delaunay Triangulation

Michel Buffa Olivier D. Faugeras Zhengyou Zhang

INRIA Sophia-Antipolis
BP 109 - 06561 Valbonne Cedex - FRANCE

Abstract

Our robot is equipped with a trinocular vision system that has been put into hardware and delivers 3D maps of the environment at a rate between 1 and 5Hz. Those 3D maps contain line segments extracted from the images and reconstructed in three dimensions. This vision machine is heavily used in our laboratory, and our navigation system uses the 3D segments it produces to compute the free space around our mobile robot, locate the obstacles and plan a safe trajectory.

1 Introduction

Two years ago, before the availability of that vision machine, we published a paper [BFZ90] describing our work on obstacle avoidance and trajectory planning for a mobile robot using stereo vision. Since then, we have developed a complete navigation system that uses the principles described in that paper, but we have considerably improved our previous results. We have now a running demonstration of our mobile robot exploring a room in order to reach an arbitrary goal while locating and avoiding obstacles. At each step, each time a new 3D map has been obtained, a ground floor 2D map of the explored space is updated, and a trajectory is planned either to go to the goal if a safe route exists, or to explore unknown areas of the room if no such route could be found.

To this end, we project the 3D segments coming from the vision machine on the ground, merge them with the previous ones in order to remove redundant information and update the ground floor map. We then compute a tesselation, more precisely a *Constrained Delaunay triangulation (CDT)* of this map using the endpoints of the 2D segments and then determine the free space by erasing the triangles that do not contain any obstacles (a very simple processing). These parts have been described in [BFZ90].

Then, using that triangulation as a graph, we can plan a safe trajectory, move the robot and iterate the process.

In this short paper we will just recall previous results and detail the way we use the Constrained Delaunay triangulation as a support for the trajectory generation.

One main feature of our system is that the knowledge of its originally unknown environment can be updated very quickly each time a new set of data is available. Projecting the 3D segments on the ground, merging them with the existing 2D map, updating the Delaunay triangulation, re-computing the free space and planning a safe trajectory, all this takes about 4 seconds on a Sparc station and less than 1 sec on a Sparc2. This performance is possible because of the dynamic properties of each part of our navigation system. We use a very recent algorithm for computing the Delaunay triangulation and we propose an original utilization of a well known algorithm for constraining the set of segments so that they become edges of the triangulation.

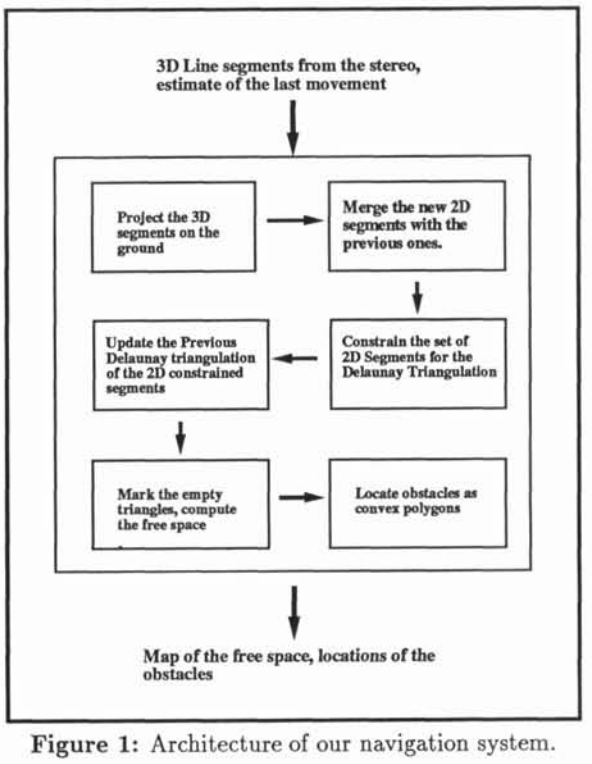


Figure 1: Architecture of our navigation system.

2 Computing the free space and locating the obstacles

2.1 What method we shall use ?

We have a 2D map made of line segments. The interpretation of this map is not an easy task, it is hard to recognize the free space and the shape of the obstacles in front of the robot. An intermediate representation is necessary.

Like [FLMB90], we propose to use a Constrained Delaunay Triangulation. The reason of our choice are detailed in [BFZ92], and can be summarized like this:

- It gives a regular tesselation of the space. It has been proved that the Delaunay Triangulation and the Constrained Delaunay Triangulation are the best triangulations for surface approximation [Llo77, SHa78, Kli80].
- From this tesselation we can obtain a polygonal approximation of the obstacles [LM89, FLMB90].
- The Delaunay triangulation is the dual of the Voronoi diagram. The latter has been heavily used in robotics as a support for trajectory planning, and we will show in this paper that the triangles can be used too.
- It can be computed dynamically in a very efficient way.

The reader is invited to look at [Aur91] for a survey of the Delaunay Triangulations and Voronoi diagrams, [Flo88] for a survey of Constrained Delaunay Triangulations.

2.2 Computing dynamically a Constrained Delaunay Triangulation (a CDT) of the set of segments

The 2D map we maintain represents what the robot has seen so far. Each time a new set of segment is available, we don't want to compute the CDT of all the segments. To update quickly the CDT we need a dynamic treatment: some segments must be added and some must be removed. Unfortunately, the first dynamic algorithms that can maintain a CDT directly have been published in August 1992 [WT92, TCK92], too late! We started our work two years ago!

Instead of designing a new dynamic algorithm from scratch, we choosed a two step method: we dynamised the static algorithm of [FLMB90] that adds points on the segments so that the standard Delaunay triangulation computed on the set of points (extremities of the original segments and added points) includes all the segments as edges of the triangulation. This dynamisation was rather easy because it appeared that when a segment is added or removed, only a local treatment is necessary, due to the following properties:

- $s = (P, Q)$ will be a Delaunay edge if the circle of diameter PQ does not intersect any other segment.
- The two segments $s_i = (P_i, Q_i)$ and $s_j = (P_i, Q_j)$ which have a point in common will be delaunay edges if the circle passing through P_i, Q_i, Q_j does not intersect any other segment.

We used this dynamized algorithm in conjunction with the algorithm of [ODT90] which computes dynamically the standard Delaunay triangulation of a set of points.

2.3 Computing the free space, locating the obstacles

We described in [BFZ90] a way to compute the free space and locate the triangles from the CDT of the set of segments, using a very simple visibility criterion.

3 The trajectory generation module

3.1 Introduction

The behavior of the robot in our demonstration is close to the human one: it look around him before moving in a hostile environment, then move, then look again and so on...

The algorithm is performed as follows:

1. Take a panoramic view of what is in front of the robot by rotating only the triplet of cameras, update the 2D map, compute the free space.
2. Look if there are *possible passages* to approach the goal. If such passages exist, compute the paths to go to them. If there are more than one possible passage, choose the best one among them.
3. If there is a passage, perform the corresponding movements computed in step 2. and go to step 1.
4. If no possible passage has been found, perform a safe movement in the free space so that the next panoramic view will reveal things that have not been seen before. Go to step 1.

Explanation of step 2:

An edge of the triangulation is a *possible passage* if:

- It is on the boundary of the free space.
- It is an edge of two triangles (one from the free space and one internal to the convex hull).
- It is not a physical segment.
- It is long enough. (The robot must be able to cross it).

This definition means that a possible passage is an edge of the convex hull that has been built by the process that removes the empty triangles, but does not represent something that has really been seen (see figure 2). In that case, the robot is attracted by such an "unknown" part of the map he built so far. He wants to see what's lying there and if there is a passage that can lead him to the goal.

3.2 How we compute a path

The possible passages are determined during the computation of the boundary of the free space. See [BFZ90] for more details.

To compute the different paths to the passages, we first associate to each passage the corresponding empty triangle.

Then, we can consider the set of triangles that belong to the free space as a graph whose vertices are the centers of gravity of each triangle and edges are links between each pair of adjacent triangles (see figure 3). The length of each edge represent the distance between the barycenters of two adjacent triangles. We can then apply a common shortest-path algorithm to determine the shortest path from any triangle to the triangle right in front of the robot. This is a standard Dijkstra algorithm that examines recursively all the triangles starting from one particular triangle (in our case the one in front of the robot) and updates precedence relationships. At the end, it is possible to know the shortest way to go from any triangle to the initial triangle just by using these

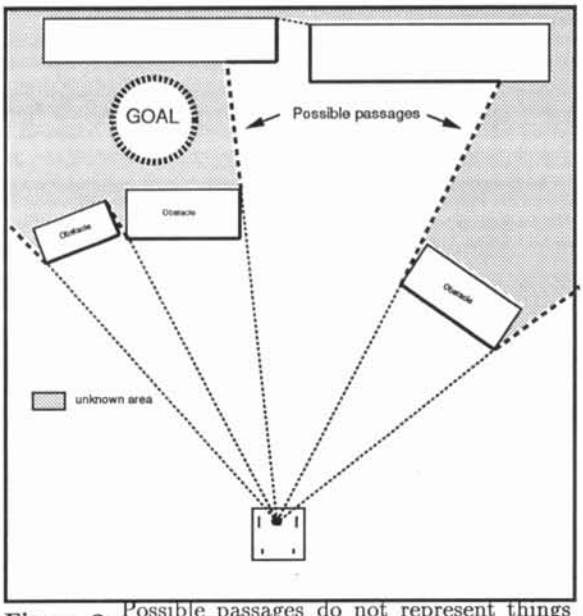


Figure 2: Possible passages do not represent things that have been seen

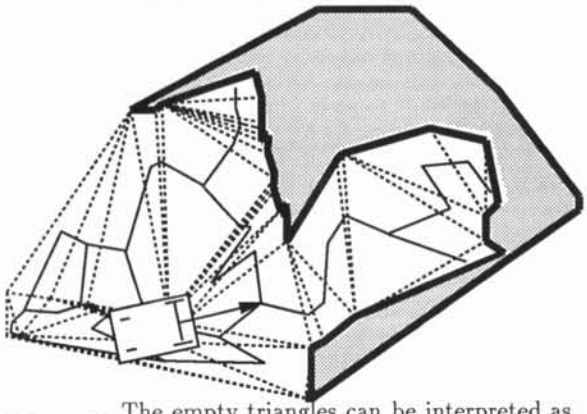


Figure 3: The empty triangles can be interpreted as a graph.

precedence relationships.

If during the examination of the triangles we don't take into account the triangles that are too close from the obstacles, then we greatly improve the execution time of the shortest path computation. Of course, discarding empty triangles can make some passages impossible to reach. For example if there is a narrow gap just on the way to a large passage, no safe way can be computed to go to that passage. See figure 5.

So what we do to determine the set of possible paths for the robot is: we examine each possible passage, we look at the corresponding empty triangle, and if there is a path from the triangle in front of the robot then we mark it. We then choose among these paths the one whose end is the closest to the goal.

3.3 Simplifying the path

Figure 4 shows clearly that these paths can't be performed without a further treatment: they are not smooth at all. Depending on the shapes of the adjoining triangles the turn angle between each edge of the graph may be very big, resulting in a very jagged trajectory. We propose a very simple method for simplifying and

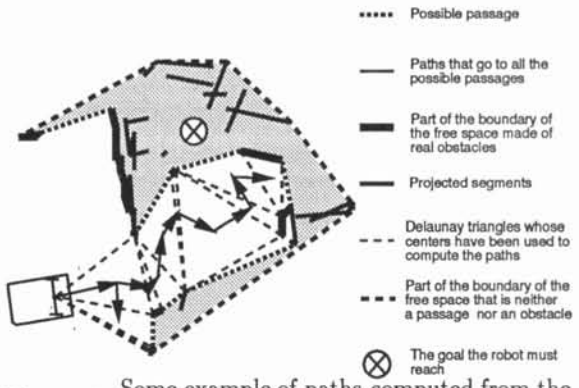


Figure 4: Some example of paths computed from the graph formed by the empty triangles

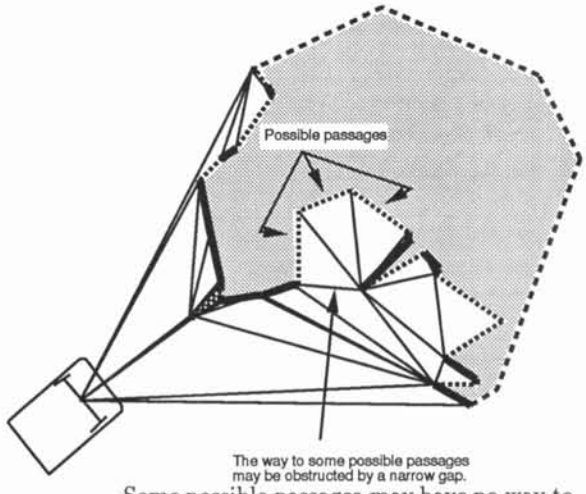
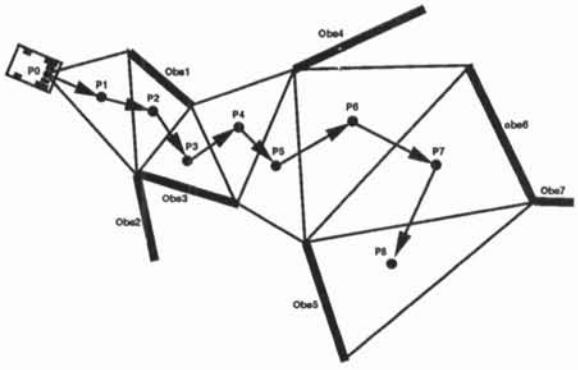


Figure 5: Some possible passages may have no way to go to them.

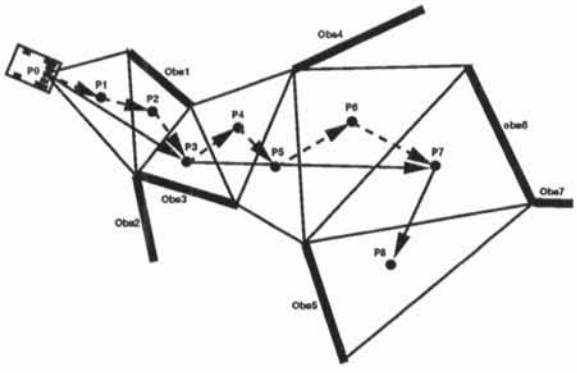
smoothing such paths:

We start recursively from $P0$. We will first try to simplify the path by going directly from $P0$ to $P2$. Is it possible? Yes, the robot won't hit any obstacle. We then try $(P0-P3)$. It works too. Let us try $(P0-P4)$! This time the simplification is impossible: the straignt line $(P0-P4)$ is too close to $Obs1$. We then try $(P0-P5)$, $(P0-P6)$... and so on until $(P0-P8)$. Starting from $P0$, the only simplification we found was to go directly from $P0$ to $P4$. We introduce that modification, and start again with the way point just after $P0$ now: $P3$. Doing the same processing, we see that it is possible to go directely from $P3$ to $P7$, but not to $P8$ as the segment $P3-P8$ is too close to $Obs5$. So, we then try to find a shortcut that starts from $P7$ and so on... The treatment is completed when the last point has been examinated. With the example of figure 6, the final simplified path is $(P0-P3)$, $(P3-P7)$ and $(P7-P8)$.

When we test if a shortcut is possible, we have to check if it is not too close to any obstacle. This test is made using two different distance functions: one that computes the distance between two line segments (for example when we test if $(P0-P4)$ is not too close to $Obs1$), and one that computes the distance between a line segment and a point (when we test if $(P3-P7)$ is not too close to the extremity of $Obs4$).



The path before simplification



The simplified path

Figure 6: How we simplify the path

It is a very simple recursive treatment. Once we have got the final simplified path, we just translate it into orders we send to the robot.

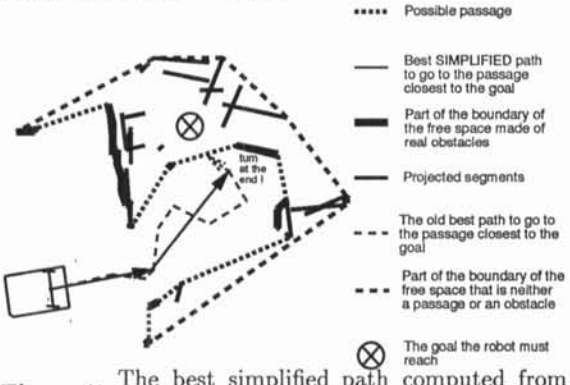


Figure 7: The best simplified path computed from the possible paths of the figure 4

If no path is found, then the robot will perform a safe movement in the free space so that it can look in the unexplored part of the room and maybe find a way to go to the goal, as shown in figure 8.

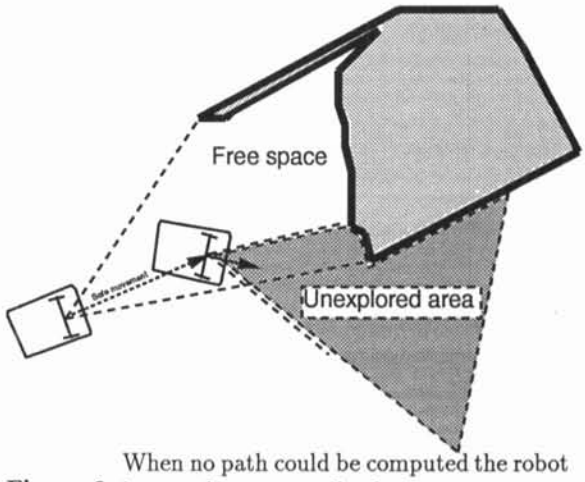


Figure 8: When no path could be computed the robot just performs a small safe movement to see a new area

References

[Aur91] F. Aurenhammer. Voronoi ddiagrams - a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3), September 1991.

[BFZ90] Michel Buffa, Olivier Faugeras, and Zhengyou Zhang. Obstacle avoidance and trajectory planning for an indoors mobile robot using stereo vision and delaunay triangulation. In *Proceedings of Roundtable Discussion on Vision-Based Vehicle Guidance '90*, pages 12.1–12.8, Science University of Tokyo, July 1990. IEEE.

[BFZ92] Michel Buffa, Olivier Faugeras, and Zhengyou Zhang. Obstacle avoidance and trajectory planning for an indoor mobile robot using stereo vision and Delaunay triangulation. In I. Masaki, editor, *Vision-based Vehicle Guidance*, chapter 13, pages 268–283. Springer, New York, 1992.

[FLMB90] Olivier D. Faugeras, Elizabeth Lebras-Mehlman, and Jean-Daniel Boissonnat. Representing stereo data with the delaunay Triangulation. *Artificial Intelligence Journal*, 44(1-2), July 1990. Also INRIA Tech. Report 788.

[Flo88] L.De Floriani. *A Survey of Constrained Delaunay Triangulation Algorithms for Surface Representation*, chapter Issues on Machine Vision. Springer-Verlag, 1988.

[GM79] A.L. Zobrist. G.K. Manacher. Neither the greedy nor the delaunay triangulation of a planar set of points approximates the optimal triangulation. *Inf Proc Lett*, 9:31–34, 1979.

[IB76] AK. Aziz I. Babuzka. On the angle condition in the finite element method. *Numerical Analysis*, 13(2):214–226, 1976.

[Kli80] GT. Klingsek. Minimal triangulations of polygonal domains. *Computer Graphics and Image Processing.*, 9:121–123, 1980.

[Llo77] EL. Lloyd. On triangulations of a set of points in the plane. In *Proc IEEE 18th Annual Symp on the foundations of Computer Science.*, pages 228–240. IEEE, 1977.

[LM89] Elisabeth Lebras-Mehlman. *Representation de l'environnement d'un robot mobile.* Phd thesis, Universite de Paris-Sud, Centre d'Orsay, 1989.

[ODT90] S. Meiser O. Devillers and M. Teillaud. Fully dynamic delaunay triangulation in logarithmic expected time per operation. *Computational Geometry Theory and Applications.*, 1990. To be published. Available as Technical Report INRIA 1349. Abstract published in LNCS 519 (WADS'91, august 1991).

[SHa78] M.I. SHamos. *Computational Geometry.* Phd thesis, Yale University, New Haven, Connecticut, 1978.

[TCK92] David M. Mount Thomas C. Kao. Incremental construction and dynamic maintenance of constrained delaunay triangulations. In *Proceedings of the Fourth Canadian Conference on Computational Geometry.*, August 10-14 1992.

[WT92] Cao An Wang and Y. H. Tsin. Efficiently updating constrained delaunay triangulations. In *Proceedings of the fourth Canadian Conference on Computational Geometry*, August 10-14 1992.