A Stereovision-based navigation system for a mobile robot

Michel Buffa, Olivier Faugeras, Zhengyou Zhang

Rapports de Recherche

N°1895

Programme 4

Robotique, Image et Vision

A STEREOVISION-BASED NAVIGATION SYSTEM FOR A MOBILE ROBOT

Michel BUFFA
Olivier FAUGERAS
Zhengyou ZHANG

Avril 1993

A stereovision-based navigation system for a mobile robot[1]

Un système de navigation pour robot mobile, qui utilise la stéréovision

Michel Buffa **Olivier D. Faugeras**
Zhengyou Zhang

INRIA Sophia-Antipolis BP 93
06902 Sophia-Antipolis Cedex, France

E-Mail: buffa@sophia.inria.fr, faugeras@sophia.inria.fr
zzhang@sophia.inria.fr

[1]This work was supported in part by Esprit project P940.

Abstract

This report describes the work at INRIA on obstacle avoidance and trajectory planning for a mobile robot using stereovision.

Our mobile robot is equipped with a trinocular vision system that has been put into hardware and is be capable of delivering 3D maps of the environment at rates between 1 and 5 Hz. Those 3D maps contain line segments extracted from the images and reconstructed in three dimensions. They are used for a variety of tasks including obstacle avoidance and trajectory planning.

For those two tasks, we project on the ground floor the 3D line segments to obtain a two-dimensional map, we simplify the map according to some simple geometric criteria, and use the remaining 2D segments to construct a tessellation, more precisely a triangulation, of the ground floor. This tessellation has several advantages:

- It is adapted to the structure of the environment since all stereo segments are edges of triangles in the tessellation,
- It can be efficiently computed (the algorithm we use for the triangulation has a complexity of $O(\log n)$ per update, if n is the number of points used),
- It is dynamic, in the sense that segments can be added or subtracted from an existing triangulation efficiently,

We use this triangulation as a support for further processing. We first determine free space, simply by marking those triangles which are empty, again a very simple processing, and then use the graph formed by those triangles to generate collision free trajectories. when new sensory data are acquired the ground floor map is easily updated using the nice computational properties of the Delaunay triangulation and the process is iterated.

We show examples in which our robot navigates freely in a real indoors environment using this system.

Résumé

Ce rapport décrit le travail effectué par le laboratoire Robotvis de l'INRIA sur la planification de trajectoire et l'évitement d'obstacles pour un robot mobile, à l'aide de la stéréovision. Notre robot mobile est équipé d'un système de vision trinoculaire hardware capable de délivrer des cartes 3D de l'environnement à une fréquence allant de 1 à 5Hz. Ces cartes 3D se composent de segments de droites qui sont calculés à partir des images et reconstruits en trois dimensions. Nous les utilisons pour accomplir diverses tâches, comme planifier une trajectoire et éviter les obstacles. Pour ces deux dernières tâches, nous procèdons de la manière suivante: les segments 3D sont projetés sur le sol afin d'obtenir une carte bidimensionnelle; cette carte est ensuite simplifiée à l'aide de critères géomètriques simples, puis une triangulation de Delaunay des segments 2D est alors effectuée. Cette triangulation a plusieurs avantages:

- Elle est adaptée à la structure de l'environnement puisque tous les segments sont des arrêtes des triangles.
- Elle peut être calculée très efficacement (l'algorithme que nous utilisons a une complexite $O(\log n)$ par mise à jour, si n est le nombre de points à trianguler.
- Elle est dynamique dans la mesure où chaque segment peut être ajouté ou retiré d'une triangulation existante rapidement.

Cette triangulation est utilisée comme support pour d'autres traitements. Nous déterminons tout d'abord l'espace libre simplement en marquant les triangles qui sont vides, puis utilisons le graphe formé par ces triangles pour générer une trajectoire sans collision.

Chaque fois que des nouvelles données provenant des capteurs vision sont disponibles, la carte 2D est rapidement mise à jour grace aux propriétés intéressantes de la triangulation de Delaunay.

Nous montrons à la fin de ce rapport des exemples réels de notre robot explorant une pièce encombrée, avec à chaque étape la reconstruction de l'espace libre et la localisation des obstacles autours du robot.

Contents

List of Figures

1 Introduction

Our mobile robot can use a vision machine which has been designed and built within an European Esprit project, project P940, also called DMA, for Depth and Motion Analysis. Without entering into the details of this machine, it is sufficient to say here that it can process three images acquired from a trinocular stereo rig and compute the position and orientation in three dimensions of a set of line segments. These line segments are produced by polygonal approximations of edges in the three images. The stereo algorithm that matches the polygonal chains has been described in [AL87] and implemented in parallel on a board with 3 Motorolla 56000 DSP's. This board, part of the DMA machine, has been designed and built jointly by INRIA and MATRA.

At the time of this writing the final version of the machine is just available in our laboratory, and we will start to use it intensively. The throughput time is between 1 and 5 Hz, depending upon the exact hardware configuration. This means that, in the best case, we will reconstruct a three-dimensional wireframe representation of the environment five times a second.

Having put things in perspective, what we want to describe in this article is a piece of work that is built on top of the DMA machine and plans to use its real time capabilities.

The task is to build local representations of the robot environment that can be used to dynamically map free space, plan and update trajectories. We use the word trajectory and not itinerary to mark the difference in time scale and in planning complexity. The representation we are going to describe is local and does not carry any semantics. It is a pure volumetric representation of the free space around the robot, as measured by the stereo system over the last few seconds. It is therefore a combination of, let us say, no more than 25 three-dimensional wireframes, which is used to support tracking of an itinerary which has been set up as a goal by another, slower process. The representation is thus local, both in space and in time. The use of this local trajectory planning is to allow this process to gather information about the actual state of the environment and in particular about events that could not be foreseen, such as the approach of a moving obstacle. This local representation is, on one hand passed over to the higher-level, slower process and, on the other hand used immediately, in a reactive fashion, to cope with possible collisions and to perform the actual robot control operations that lead to a satisfying pattern of motions.

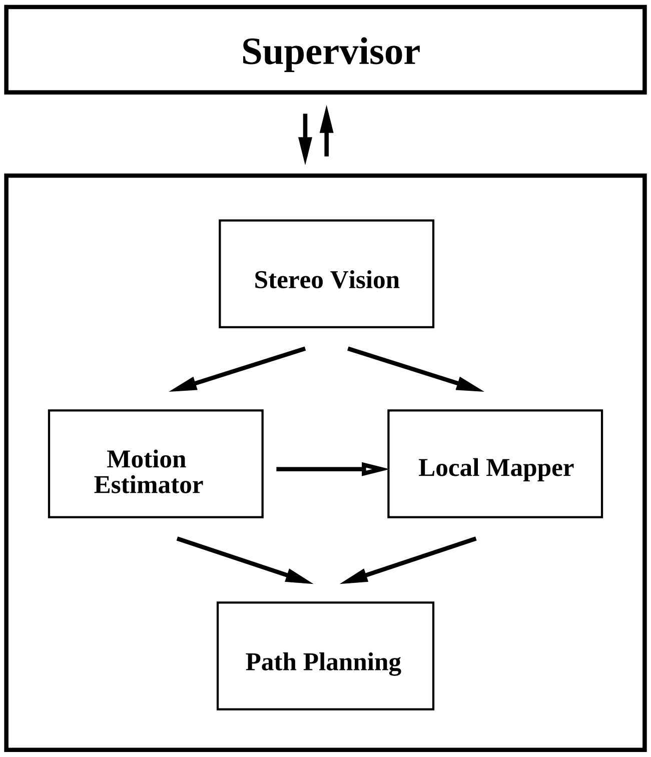

Figure 1: Architecture of our system

2 The motion estimator

When our robot moves, we must register successive local frames provided by the stereo system and estimate the motion of the robot. This is performed by the motion estimator, which estimates the movement using two kinds of informations: two consecutive 3D wire-frames and the odometric data coming from the sensors on the wheels of the robot. This estimator uses the odometry information as an initial motion estimation to match two consecutive stereo frames, and then the matching results to refine this estimation. This process is describe elsewhere [AF89, ZF90b] and provides at the same time an estimate of the motion and a measure of its uncertainty. If the number of matched segments is too small or null, only the odometer information will be used as the estimate of the movement.

3 The map making process

3.1 What do we need ?

The purpose of the map making process is to build and update a 2D map each time new data from the vision sensors are available, so it should operate incrementally. Besides, the representation used for the map making should reflect the structure of the environment the robot has seen so far, and should be suitable for navigation purposes. This means that we would like to know both the free space and the object shapes and positions in order to position the robot precisely in its environment and to plan a safe trajectory. Finally, we would like that the whole process could be achieved in a small amount of time.

In the following sections, we propose one solution which uses the Delaunay triangulation. We will explain why the Delaunay Triangulation based representation is an appropriate one. All the different steps from the arrival of a new set of 3D stereo line segments to the final result (an updated volumetric 2D map), will be described.

Now, let us see how to update the 2D map when a new set of 3D line segments becomes available to the map making process.

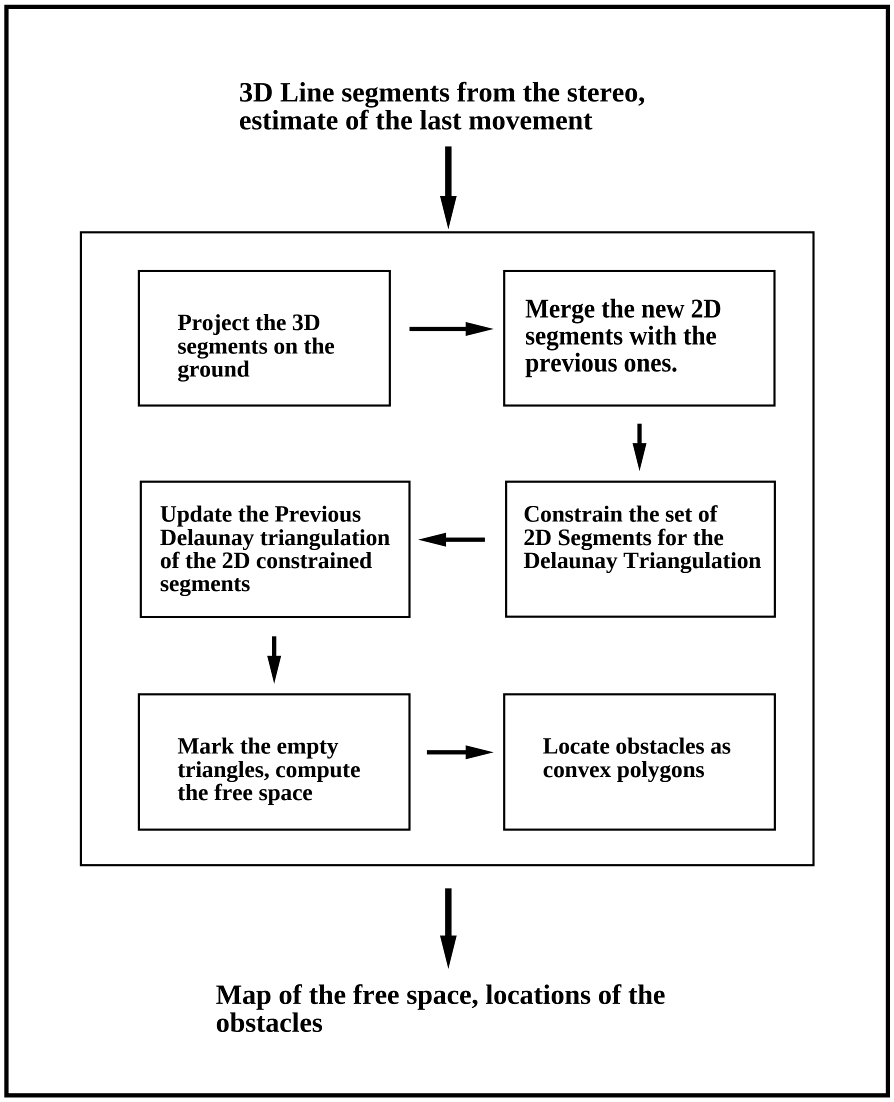
3D Line segments from the stereo, estimate of the last movement
Project the 3D segments on the ground
Merge the new 2D segments with the previous ones.
Constrain the set of 2D Segments for the Delaunay Triangulation
Update the Previous Delaunay triangulation of the 2D constrained segments
Mark the empty triangles, compute the free space
Locate obstacles as convex polygons
Map of the free space, locations of the obstacles

Figure 2: The map making process architecture

3.2 Projecting the 3D segments on the ground

We assume that for the moment, the ground on which the robot is moving can be considered as flat. This local plane is known through calibration and used as a support for our representation.

We project on the ground all 3D line segments obtained from stereo which lie between the ground and the plane parallel to it at a height equal to that of the robot. The 3D segments whose endpoints are either on or under the ground plane are not projected because they are reflexions or markings on the ground and don't match any real obstacle.

We have thus reduced a three-dimensional wireframe representation to a two-dimensional one. To help guide the reader's understanding, we show in figure 3 an image of a part of a typical scene that the robot has to cope with, and the projection on the ground floor of the 3D line segments reconstructed from stereo. We notice that this representation is probably not immediately useful, being still quite complex geometrically.

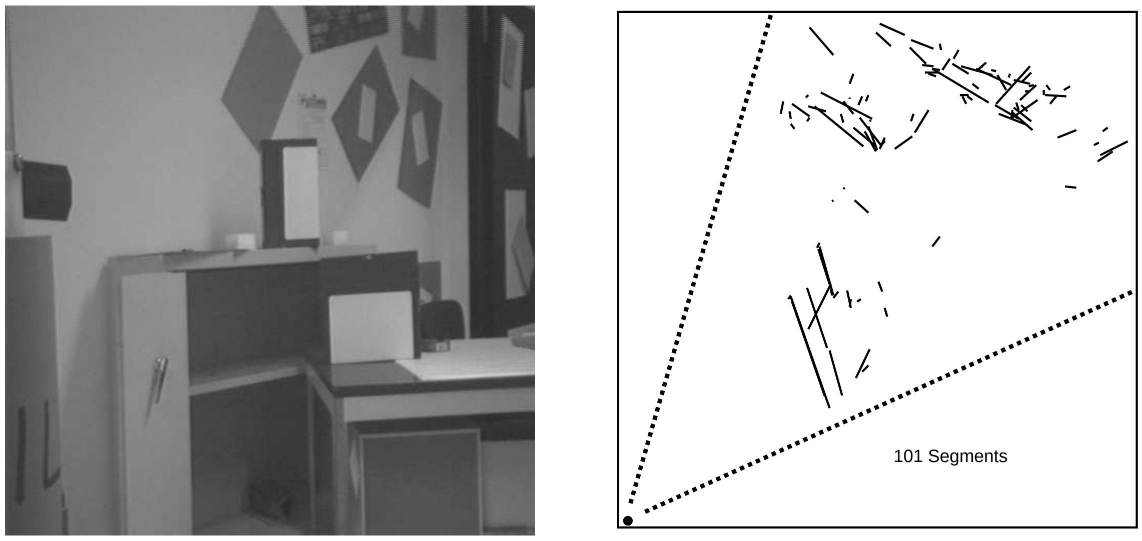

Figure 3: A typical indoor scene and the projection of the 3D segments produced from a stereo view of this scene

A further item of interest is that in the process of reconstructing 3D line segments, we also compute a measure of their uncertainty. The exact measure depends on what kind of representation we are using for the line segments [AF89, ZF90a] but it always represents the amount of confidence we have in a specific segment. This confidence is a function of the reliability of the various elements in the processing chain such as, edge detection, polygonal approximation, calibration of the trinocular stereo rig, as well as of the geometry of the scene (segments which are far are less reliable than segments which are close). It is represented by a symmetric weight matrix, also called sometimes a covariance matrix if a probabilistic interpretation is required, whose size is that of the representation,

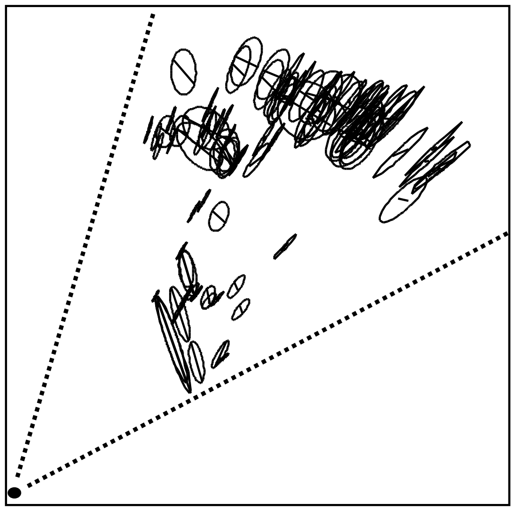

Figure 4: Ellipses of uncertainty for the midpoints of the 2D segments of figure 3

and whose diagonal elements are big if the corresponding parameters in the representation is uncertain, and small otherwise.

From the 3D representation of the line segments and the corresponding measure of uncertainty it is possible to compute a representation for the projected segments and the corresponding uncertainty (see section 3.3.2). Uncertainties are conveniently represented by ellipses in the plane where they can be thought of as geometric upper bounds. For example, figure 4 shows the ellipses representing the uncertainties of the midpoints of the projected segments of figure 3. We note that segments which are far from the robot are more uncertain than those which are close.

Having such a measure is important because it allows processes that operate on the representations to give more weight to reliable measurements than to others. An example of such a process is one that estimates the robot egomotion [AF89, ZFA88, ZF90a] and the obstacles motion [ZF90c, ZF90b]. In this article we use this information to help producing a simplified version of the 2D map of figure 3. The main reason for producing such a map is that of conciseness. Since our goal is to produce a map of free space, we are uninterested in redundant information such as too many, geometrically similar, segments in the same area that possibly arise from a highly textured pattern. Suppressing redundant information will allow us to keep only the information really relevant to the task at hand and will facilitate and speed up further processes.

3.3 Updating the 2D map with integrating the new set of projected segments

At each new cycle, the 2D segments coming from the last wireframe are merged with the set of 2D segments which represent what the robot has seen in the previous cycles. The map of the 2D segments is updated.

Before studying the merging process, we describe how line segments are represented.

3.3.1 Representation of line segments

A (2D or 3D) line segment is usually represented by its endpoints M_1 and M_2 and their covariance matrices Λ_1 and Λ_2. Since the endpoints of a segment are not reliable, we cannot directly use them in most cases. Considering the longitudinal (along the segment) position of a line segment is much less precise than the transverse one, one may use instead its supporting line. However, after this abstraction we lose completely the longitudinal information, which is useful in many cases such as matching and fusion. Further, the uncertainty of the supporting line does not reflect that of the segment, *i.e.*, the supporting line of a very uncertain segment may be less uncertain than the supporting line of a less uncertain segment. For those reasons, we have proposed a new representation for 3D line segments which is a trade-off between lines and segments [Zha90, ZF90a].

In our representation, the spherical coordinates ϕ and θ are used to represent the direction of a line segment, and the midpoint $\mathbf{m}$ to locate the segment. The length of the segment is denoted by l. If we denote $M_2 - M_1$ by $\mathbf{v} = [x,\, y,\, z]^t$ (the non-normalized direction vector), the unit direction vector by $\mathbf{u} = \mathbf{v}/\|\mathbf{v}\|$, $[\phi,\, \theta]^t$ by $\boldsymbol{\phi}$, we get

$$\begin{aligned} \phi &= \begin{cases} \arccos \frac{x}{\sqrt{x^2+y^2}} & \text{if } y \geq 0 \\ 2\pi - \arccos \frac{x}{\sqrt{x^2+y^2}} & \text{otherwise} \end{cases} \\ \theta &= \arccos \frac{z}{\sqrt{x^2+y^2+z^2}}. \end{aligned} \tag{1}$$

The covariance matrix $\Lambda_{\boldsymbol{\phi}}$ of $\boldsymbol{\phi}$ is given, to the first order approximation, by

$$\Lambda_{\boldsymbol{\phi}} = \frac{\partial \boldsymbol{\phi}}{\partial \mathbf{v}} \Lambda_{\mathbf{v}} \frac{\partial \boldsymbol{\phi}}{\partial \mathbf{v}}^t, \tag{2}$$

where $\frac{\partial \boldsymbol{\phi}}{\partial \mathbf{v}}$ is the Jacobian matrix of $\boldsymbol{\phi}$ with respect to $\mathbf{v}$, and $\Lambda_{\mathbf{v}} = \Lambda_1 + \Lambda_2$. Similarly, we can compute the covariance matrix $\Lambda_{\mathbf{u}}$ of the unit direction vector $\mathbf{u}$.

The most important part of our representation is the modelization of the uncertainty of the midpoint. The midpoint of a line segment is modeled as

$$\mathbf{m} = (M_1 + M_2)/2 + n\mathbf{u}, \tag{3}$$

where n is a random variable. In fact, equation 3 says that the midpoint of a segment may vary randomly around its position $(M_1 + M_2)/2$ along the direction of the segment.

The random variable n is modeled as Gaussian, zero mean and whose standard deviation σ_n is some positive scalar. In our implementation, σ_n is related to the length l of the segment, $\sigma_n = \kappa l$ (we choose $\kappa = 0.2$). This says that a long segment is much likely to be broken into smaller ones in other views. The covariance matrix of $\mathbf{m}$ is given by (see [ZF90a, Zha90])

$$\Lambda_{\mathbf{m}} = (\Lambda_1 + \Lambda_2)/4 + \sigma_n^2(\Lambda_{\mathbf{u}} + \mathbf{u}\mathbf{u}^t). \tag{4}$$

The uncertainty in the length of a segment is not modeled because it is not required in our algorithm.

A 2D line segment can be viewed as the orthogonal projection of a 3D line segment on a plane. A 2×3 matrix T can be used to describe this projection, *i.e.*, we can relate a 3D point $\mathbf{m}_3$ with its projection $\mathbf{m}_2$ (here subscripts are used to denote the dimension, which will be omitted if no ambiguity) by the following equation

$$\mathbf{m}_2 = T\mathbf{m}_3. \tag{5}$$

For example, if we want to project a 3D point on the plane $y = 0$, then $T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$. We represent 2D line segments similarly to 3D line segments, that is, a 2D line segment is described by its angle θ with the axis x, its midpoint $\mathbf{m}$, and its length l. Those parameters can be easily computed based on equation 5. We compute also the variance σ_θ for θ and the covariance matrix $\Lambda_{\mathbf{m}}$ for $\mathbf{m}$. For example, $\Lambda_{\mathbf{m}_2} = T\Lambda_{\mathbf{m}_3}T^t$. The uncertainty on l is not modeled.

3.3.2 Merging the 2D segments, simplification of the 2D map

At this point, we suppose that we have a 2D map representing what the robot has seen previously, and a set of 2D segments representing the last informations the robot has seen.

Using the estimation of the movement given by the motion estimator, the previous set of segments of the old map can be transformed in the current coordinate system, and the merging process can be attempted with the set of newly measured segments.

The figure 3 shows clearly that the projected segments contain redundant information: some colinear segments are very close, some segments seem to represent the same physical object but exploded in several parts, in some areas the segments are very dense...

In the following, we describe how to intelligently merge geometrically similar segments in the same area, which are, for example, projections of some textures on a wall.

The simplification is performed as follows. All segments are sorted by a bucketing technique which allows to easily access the neighbors of a segment. The segments are also sorted according to their orientation. Only a few computations are requiered to update the list of neighbors when the new 2D segments are added to the old ones.

Now for a segment not yet processed S, we know the list of segments which are neighbors to it and another list of segments which have similar orientations to it. The intersection of the two lists are the first candidates. If a segment S' among those candidates is similar enough to S, we then merge S and S' yielding a new segment $\hat{S}$. We then compare $\hat{S}$ with the rest of the candidates: if a new similar segment S'' is found, $\hat{S}$ will be updated. The above procedure is applied to every candidate and every unprocessed segment.

The merging technique for 2D line segments is the adaptation of that for 3D line segments [ZF90a, Zha90]. Let θ, $\mathbf{m}$ and l be the parameters of a segment S with uncertainty measurements σ_θ^2 and $\Lambda_{\mathbf{m}}$, and θ', $\mathbf{m}'$ and l' that of another segment S' with uncertainty measurements $\sigma_{\theta'}^2$ and $\Lambda_{\mathbf{m}'}$. Those two segments can be merged if and only if they satisfy the following relations based on the Mahalanobis distance

$$(\theta - \theta')^2/(\sigma_\theta^2 + \sigma_{\theta'}^2) \leq \kappa_\theta, \tag{6}$$

$$(\mathbf{m} - \mathbf{m}')^t(\Lambda_{\mathbf{m}} + \Lambda_{\mathbf{m}'})^{-1}(\mathbf{m} - \mathbf{m}') \leq \kappa_{\mathbf{m}}, \tag{7}$$

where κ_θ and $\kappa_{\mathbf{m}}$ are thresholds. Looking up the χ^2 distribution table, we can choose $\kappa_\theta = 3.84$ for a probability of 95% with 1 degree of freedom and $\kappa_{\mathbf{m}} = 5.99$ for a probability of 95% with 2 degrees of freedom. The above conditions say that we merge only segments which are the same (in the probabilistic sense).

Based on a minimum-variance estimator, we get the following parameters for the merged segment $\hat{S}$:

$$\begin{aligned}
\hat{\theta} &= (\sigma_{\theta'}^2\theta + \sigma_\theta^2\theta')/(\sigma_\theta^2 + \sigma_{\theta'}^2), && (8)\\
\sigma_{\hat{\theta}}^2 &= \sigma_\theta^2\sigma_{\theta'}^2/(\sigma_\theta^2 + \sigma_{\theta'}^2), && (9)\\
\hat{\mathbf{m}} &= \Lambda_{\mathbf{m}'}(\Lambda_{\mathbf{m}} + \Lambda_{\mathbf{m}'})^{-1}\mathbf{m} + \Lambda_{\mathbf{m}}(\Lambda_{\mathbf{m}} + \Lambda_{\mathbf{m}'})^{-1}\mathbf{m}', && (10)\\
\Lambda_{\hat{\mathbf{m}}} &= \Lambda_{\mathbf{m}}(\Lambda_{\mathbf{m}} + \Lambda_{\mathbf{m}'})^{-1}\Lambda_{\mathbf{m}'}. && (11)
\end{aligned}$$

They give the orientation and the position of the merged segment. Since the two segments are considered as two instances of a single segment, the union of them should be its better estimate. By projecting the endpoints of the two segments on the merged segment, we choose the farthest projections as the endpoints of the merged segment and the length can also be computed. The *real* midpoint M of the merged segment can be determined and can always be expressed as

$$M = \hat{\mathbf{m}} + s\hat{\mathbf{u}}, \tag{12}$$

where s is a scalar and $\hat{\mathbf{u}}$ is the unit direction vector of the merged segment. The above equation can be interpreted as the addition of a *biased* noise on $\hat{\mathbf{m}}$, thus the covariance matrix of M is given by

$$\Lambda_M = \Lambda_{\hat{\mathbf{m}}} + s^2(\Lambda_{\hat{\mathbf{u}}} + \hat{\mathbf{u}}\hat{\mathbf{u}}^t). \tag{13}$$

For more details, the reader is referred to [ZF90a, Zha90].

An example of real data fusion is shown in the figures 5 and 6. During the first iteration, the segments of figure 3 are fused with themselves. The results can be examined in figure 5. We notice that the number of segments is considerably reduced. Then the robot moves and get a new set of 3D data, which are used to estimate its displacement (see section 2). The previous set of 2D segments of the figure 5 is fused with the new data after projection on the ground plane. The figure 6 shows the results of this fusion. The new updated map is to be compared with the previous map of figure 5.

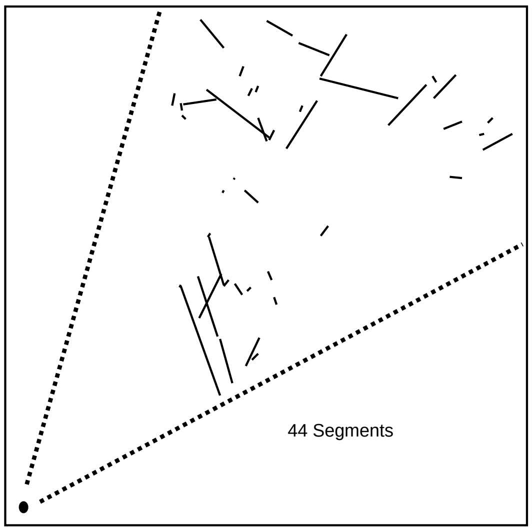

Figure 5: First step: simplification of the set of segments of figure 3

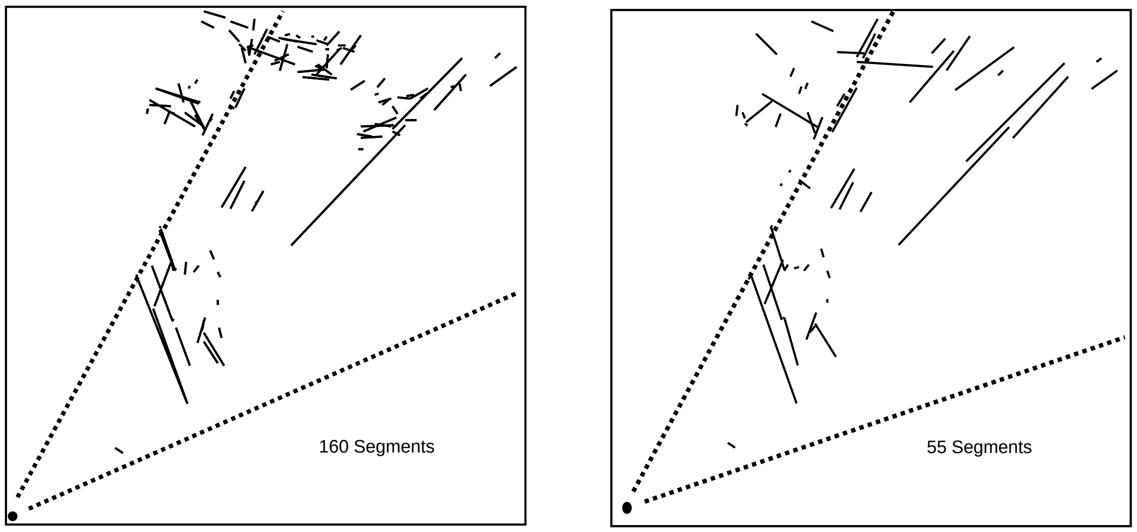

Figure 6: Updating the 2D map. Before and after the fusion

3.3.3 Preserving the visibility attribute

When the segments are merged, very often two or more segments are fused into a big one, especially if they are parts of a same physical segment seen partially from different viewpoints. Imagine that our robot turns its head clockwise and looks at an edge of a wall. The edge will be rebuilt by the merging process and if everything goes fine, should become one long segment. The problem is that the long resulting segment has been seen in reality from different viewpoints, different locations in space. We need to know which part has been seen from which robot position. This information is requiered to compute correctly the free space, as to be described in section 4.4.

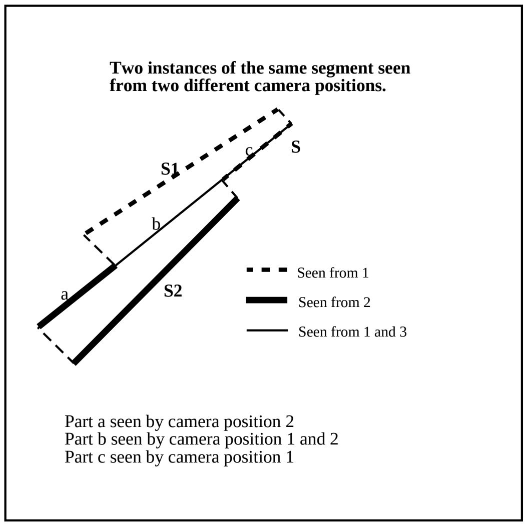
Two instances of the same segment seen from two different camera positions.
S1
c
S
b
a
S2
Seen from 1
Seen from 2
Seen from 1 and 3
Part a seen by camera position 2
Part b seen by camera position 1 and 2
Part c seen by camera position 1

Figure 7: When merging two segments, the visibility of each part of the resulting segment is computed

4 Constructing a volume representation of the free space

4.1 Constructing a Delaunay triangulation of a 2D map

The idea is to compute, as an intermediate representation, a triangulation of the endpoints of the 2D segments that has the characteristic of containing the segments as edges of the triangulation. A set of planar points is said to be triangulated if its points are joined by nonintersecting straight line segments so that every region internal to the convex hull is a triangle.

There exist many different triangulations of a set of points but we have chosen the Delaunay one because of two reasons:

- It has some nice properties related to shape, and
- It can be implemented efficiently.

Both reasons are well described in [FLMB90].

The shape property can be summarized as follows in two dimensions but it is true also in three dimensions: suppose we measure a number of points on the boundary of an object and assume that this boundary is piecewise smooth. Then, under some fairly weak assumptions, the Delaunay triangulation of the set of measured points contains a polygon that approximates the boundary shape well. If that polygon can be easily recovered from the triangulation, then we have a nice way of representing the shape of the object. How to do this is described in the section 4.4.

The Delaunay Triangulation and its dual, the Voronoï diagram, are subjects of major interest in Computational Geometry. With respect to the implementation, there exist many algorithms for computing the Delaunay triangulation of a set of n 2D points. Many compute it with an optimal time complexity of $O(n \log n)$. The basic idea is to use the divide and conquer strategy [PS85, LS80, For87]. These algorithms are rather complex and difficult to implement effectively and though they are optimal in complexity, they have one main drawback, namely that they are not incremental: we have to wait until all data points have been collected before we can start triangulating them. This is in contrast with what is needed for our application where data are collected sequentially and must be processed on the fly.

Fortunately, there exist several incremental algorithms which are suboptimal [GS78] but match our needs: they do not impose to compute again the whole triangulation at each insertion. The basic idea of these incremental algorithms is quite simple: suppose we have triangulated the first n points, and that we have just measured the $n+1$st. We exploit the *fundamental property* of the Delaunay triangulation which is the following. If ABC is a Delaunay triangle, consider its circumscribing disk. That disk does not contain any other data point. This property of the disks is necessary and sufficient for a triangulation to be a Delaunay one. Coming back to our new data point, we only have to determine those disks of the existing triangulation it falls into (the corresponding triangles are edge connected) and retriangulate their data points together with the new point. The triangles whose circumscribing disk contains a point are *in conflict* with the point.

In the last few years incremental algorithms which are non optimal in the worst case but with a good complexity have been proposed. One of them [GS78] whose worst case time complexity is $O(n^2)$ but whose average complexity is $O(n\sqrt{n})$ has been used to represent stereo data. This is described in details in [FLMB90]. Some recent algorithms use a data structure like the Delaunay Tree [BT86, BT] to update rapidly the triangulation: as said in the previous paragraph, each time a new point is inserted, the incremental algorithms must locate the triangles in conflict with this point. The principle of the Delaunay Tree is the following: instead of eliminating triangles during the different steps of the construction, all the triangles are stored as nodes of the Delaunay Tree, and at each step relationships between each triangles of the successive Delaunay Triangulation are maintained. When a point is inserted, all the Delaunay triangles in conflict are declared dead (killed by the point), but they still exist in the Tree. The new triangles become sons of these dead triangles. The Delaunay triangulation is formed by the union of all the alive triangles. The structure of the Delaunay Tree speeds up the localisation of the triangles in conflict with a point to be inserted. Recently, an algorithm based on the Delaunay Tree has been proposed by [DMT] whose randomized complexity is $O(\log n)$ per update. This algorithm allows both the insertion and the deletion of points, which is exactly what we need to update a volume representation of the free space from a 2D map made of segments, as explained in section 4.3

A further property of the Delaunay triangulation which is very relevant to our application is related to the skeleton. Indeed, if we consider the centers of the Delaunay disks and join those centers whose triangles share an edge but do not cross the boundary of free space, we obtain a set of polygonal lines that are a subset of the Voronoï diagram of the set of measured points [FLMB90]. Voronoï diagrams have been heavily used in robotics as a support for trajectory planning and the fact that the Voronoï diagram of free space is present in our representation comes as a strong support for our approach.

4.2 Constrained Delaunay triangulation

In our case we have more than points, we have segments. If we just triangulate their endpoints as shown in figure 9 we obtain the so-called unconstrained triangulation of the set of segments of figure 8. It can be seen that some segments are not Delaunay edges. This is a problem because the next step described in the following section may produce a less accurate representation of free space than if all stereo segments are Delaunay edges. The problem can be eliminated by adding some more points on some of the segments: the results on the same scene are shown in figure 9 which is the so-called constrained Delaunay triangulation of the set of stereo segments. In [FLMB90] the authors developped an algorithm that constrain the segments. This algorithm performs in $O(n^2)$ in the worst case, but our implementation uses *bucketing techniques* so the average running time is quite less than that. The basic idea is to modify the input data by adding more points to the initial segments. This can be done using the fundamental property of the Delaunay Triangulation described earlier in this section. This property involves that if the circle defined by a segment (as its diameter) does not intersect any other segment, then this segment will be a delaunay edge of the resulting triangulation. It is a sufficient condition. Let C be such a circle attached to the segment s, and $s_1, s_2, \ldots, s_n$ be the segments intersecting C. It is possible to split s into a finite number of subsegments such that none of the circles attached to those segments intersects any of the s_i's. Here we omit the details of this preprocessing step and just present the algorithm for computing the constrained Delaunay Triangulation:

- For any edge in $\mathcal{S} = (s_i)_{i=1}^n$, if it does not satisfy the fundamental property, preprocess it and add the corresponding new points in a list.
- Compute the Delaunay Triangulation of the new set of points.

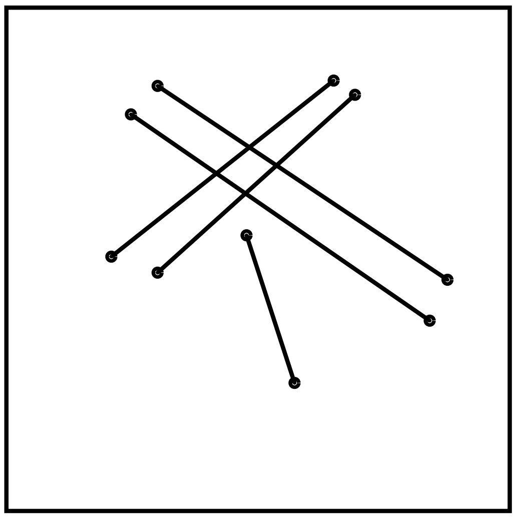

Figure 8: A 2D map of a scene

We saw in section 3.3.3 that when two or more segments are merged, we know from where each part of the resulting segment has been seen (see figure 7). When points are added on segments during the constraining phase, they inherit the visibility attribute of the part of the segment they have been put on, as shown in figure 10. This information on the points is very important for the process which compute the free space from the triangulated map of the constrained segments, and will be described in details in section 4.4.

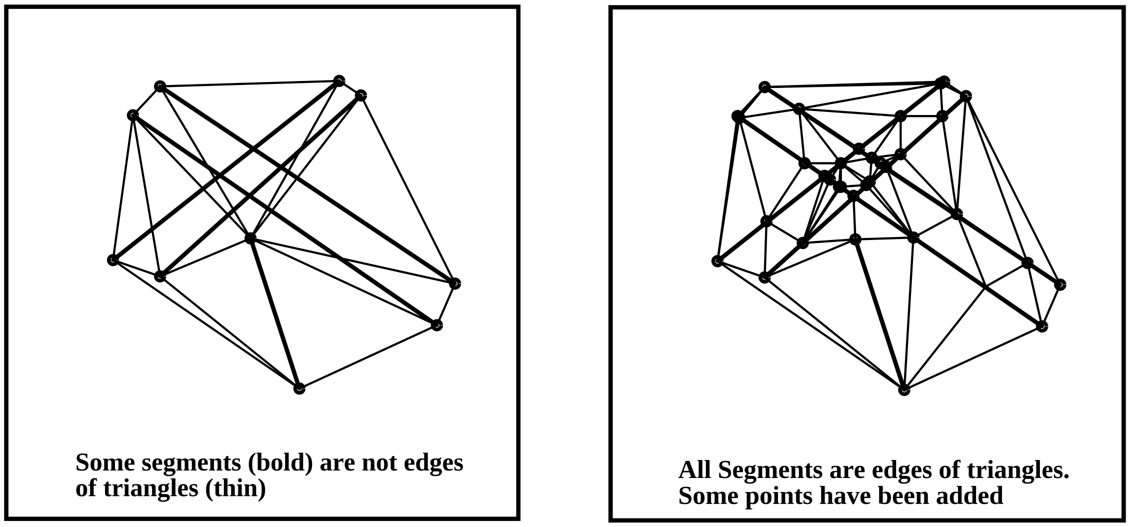
Some segments (bold) are not edges of triangles (thin)
All Segments are edges of triangles. Some points have been added

Figure 9: Unconstrained and constrained Delaunay triangulation of the set of segments of figure 8

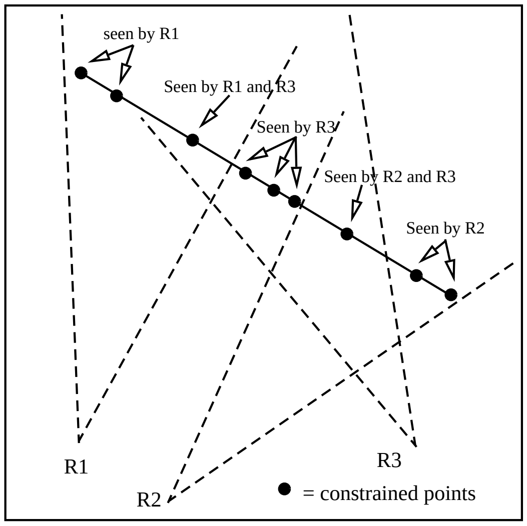

Figure 10: When a segment is constrained, the visibility attribute is inherited by the added points

4.3 Updating dynamically an existing triangulation

When a new segment is fused with an old one, the old segment is erased from the triangulation, and the new one must be inserted.

New segments that have not been seen before must be inserted too.

The segments of the updated map must be constrained before being inserted in the triangulation. The algorithm which constrain a set of segment is not incremental (see section 4.1). Fortunately, we noticed that when we move some segments from a set of already constrained segments, only the segments in the neighborhood need to be constrained again.

To sum up, the segments that must be removed from the triangulation are those which are fused with some recently observed segments, and those which need to be constrained differently due to modification of their neighborhood (whose endpoints remained the same, though).

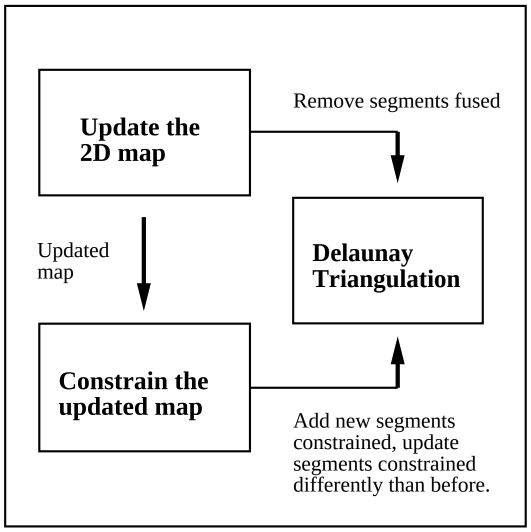
Update the 2D map
Remove segments fused
Updated map
Delaunay Triangulation
Constrain the updated map
Add new segments constrained, update segments constrained differently than before.

Figure 11: Updating the delaunay triangulation

4.4 Marking empty triangles, computing the free space

Having constructed the Delaunay triangulation of our map, we now wish to identify those triangles which are parts of the free space. In order to do this, we exploit a very simple visibility property which is best explained by looking at figure 12. In this figure, the physical segments, which represent real obstacles are the thick lines, and the other Delaunay edges are thin lines. We notice that some segments which can be seen by the robot in 3D space are now behind some other segments which represent obstacles between the robot and those segments behind them. In the figure, the point P has been seen by the robot, but the optical ray is intersected first by the segment S and the segment S_1. Those triangles crossed by the optical ray before it meets the first segment S are marked as free space.

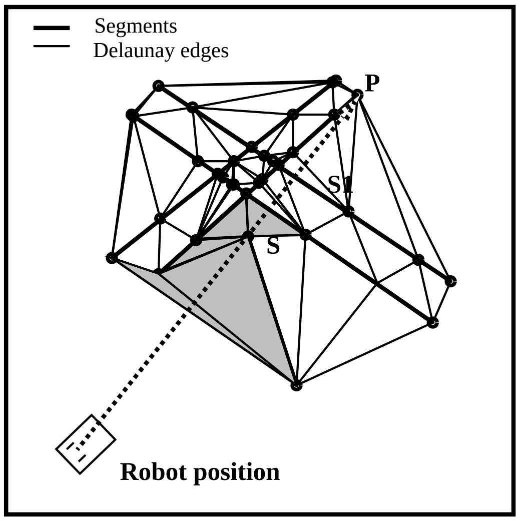

Figure 12: Basic principle of using the visibility property to mark empty triangles.

We saw in section 3.3.3 that the segments of our 2D map may have been seen partially from several viewpoints, so the points on these constrained segments may have been seen from several viewpoints too. Suppose that the example map of the figure 13 has been updated several times by merging 2D segments seen from several viewpoints and that the part of the segment which contains the point P has been seen from two different robot positions R_1 and R_2. It's necessary to "launch" two optical rays, R_1P and R_2P to mark all the empty triangles.

To sum up, the algorithm for marking the empty triangles is the following:

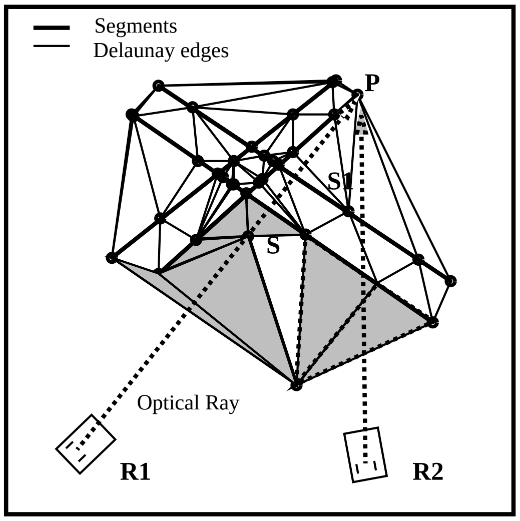

Figure 13: Using the viewpoints to mark empty triangles.

Algorithm: Marking Empty Triangles

```
• For each point $A$ of the triangulation do :
    – For each robot position $R_i$ from where the
      point $A$ has been seen
        ∗ For each triangle $\mathcal{T}$ crossed by optical ray
          $R_iA$ do (in the order from $R_i$ to $A$) :
            · Mark $\mathcal{T}$ as free space,
            · If the edge of $\mathcal{T}$ through which the
              ray $R_iA$ comes out is a segment then
              stop, else continue.
```

Once the triangles belonging to the free space are marked, it is very easy to compute a hull of each obstacle by connecting all delaunay edges which belong to both an empty triangle and a triangle which has not been marked. At this point, we know the free space, the voronoi" diagram (see section 4.1), the position and shape (as convex polygons) of the obstacles. The robot can now plan to move safely to a new position.

5 The trajectory generation module

5.1 Introduction

Actually, we have implemented a demo in which our robot first integrates five views by rotating only the triplet of cameras and builds a panoramic map of the free space in front of him, then plans a safe trajectory, moves and takes again a panoramic, and so on... This behavior is close to the human one: we look around before moving in a hostile environment.

The algorithm:

1. Take a panoramic view of what is in front of the robot, 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 14). 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. Of course there may exist several possible passages, some may not be reachable if on the way to them there are narrow gaps made by obstacles or if the robot should go out of the free space (see figure 17). So what we do is that we first determine the possible passages, then compute the shortest ways to reach them (and sometimes there is no way to go to a given passage as explained earlier), then choose among the different ways the one that make the robot go closer to the goal.