1. Introduction

The objective of a path planner for a mobile robot operating in environments with unknown obstacles (dynamic or not), is to calculate collision-free trajectories.

The research carried out in this paper has mainly been inspired by the project Robauco [1], whose main objective is the development of technologies for obtaining mobile robots capable of complex tasks that demand a high degree of autonomy and capacity of collaboration in the presence of human beings. One of the main applications of our work is the use of mobile robots in dangerous missions where the environment can be risky for humans (e.g., rescue missions, etc.). In this case, an unmanned autonomous vehicle (UAV) is sent in advance to obtain the images of the outdoor environment. Using the motion planning method proposed, the best trajectory to reach the goal is obtained over these images, and this trajectory is sent to the mobile robot. Since the UAV takes images consecutively, in case of changes in the environment, the path can be recalculated using the same method.

Fast Marching (FM) techniques have been used directly in Path Planning methods to calculate trajectories from one point to another. Nevertheless, following this strategy, the generated trajectories are not guaranteed to be safe and smooth. As explained in [2], the shortest geometrical path obtained can be unsafe, since it can touch the corners, walls, and obstacles, as shown in Figure 1.

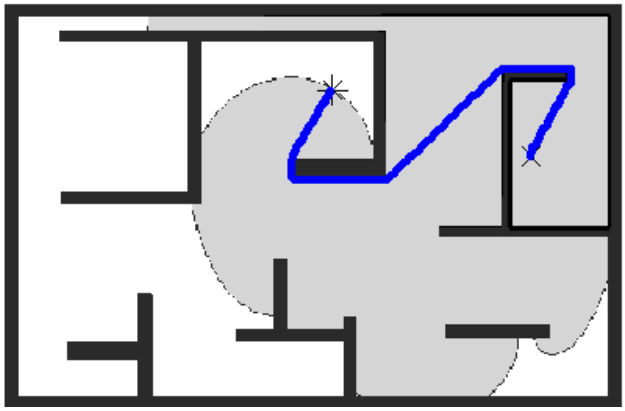

Figure 1: Trajectory calculated by the Fast Marching method directly.

Moreover, in most works, these methods are used in $2D$ environments

(indoor environments), whilst the method presented here adds another dimension to the problem (outdoor environments).

The new method proposed in this paper consists of several phases. First, as previously introduced, a $2\frac{1}{2}$ or 3D image of the surface is needed. Then, a triangular mesh is constructed over the image, which allows us to generate realistic surfaces due to the capabilities of the triangles to fit the characteristics of the map better. Once the grid is constructed, the method extracts some information from the environment to obtain the height and to calculate the gradient and the spherical variance, which gives information about the roughness of the surface. Then, it combines these data with the robot limitations to generate a weight matrix W. This matrix can be view as a difficulty or viscosity map which is situated on the 3D surface. Once the matrix is ready, the method applies the FM algorithm over this modified surface (the grid + matrix W) to generate the path.

If matrix W is not used, the trajectory obtained will be just the geodesic distance, i.e., the length of the shortest path between the two points. Applying matrix W, the proposed method gives a path which considers the features of the surface and the limitations of the robot. Moreover, it also gives us information about the speed of the robot based on the FM wave propagation speed [2, 3].

The results presented are carried out in the context of the project Robauco, that is, it is assumed that the images used are taken in advance by a UAV. First, the algorithm is used to calculate different paths, for one robot on earth, by varying the ponder factors of the W matrix. The fast execution of this method allows the updating of the given path as the images of the environment are updated. Therefore, this could be considered as an on-line path planning method. Secondly, this algorithm is also applied for avoidance collision in case of two robots approaching. In both approaches, we prove that, using the proposed method, it is possible to generate smooth and safe plans in outdoor environments.

The remainder of the paper is organized as follows. In section 2, a brief summary of other works related to grid-based methods and path planning methods on outdoor environments is presented. Next, section 3 introduces an explanation about the FM method and how this algorithm can be implemented on orthogonal and triangulated meshes. The following section, section 4, introduces the viscosity matrix W and how it is formed. Section 5 presents some results obtained by simulation. Finally, the main conclusions of this paper are summarized in section 6.

2. Related work

To solve the motion planning problem it is necessary to make use of an appropriate representation of the environment on which to implement effective planning algorithms. This section presents a brief review of grid-based planning methods, since the approach presented in this paper uses that type of environment representation.

2.1. Grid-Based Methods

In the navigation of mobile robots, many methods are based on a grid-based representation of the environment. In these methods each cell has a binary representation (occupied or free) or an associated weight that represents the difficulty of traversing that area.

This grid is usually approximated by a graph, with the nodes situated in the center of each cell. Many algorithms have been developed to find a path in the graph.

Grid- or graph-based methods, such as Dijkstra's or $A*$, calculate a navigation function constrained to movement choices along graph edges that are neither optimal for execution nor an exact solution. The well-known Dijkstra's algorithm computes the optimal path between a single source point to any other point in the graph. This algorithm is indeed efficient, but suffers from metrication errors [4]. $A*$ uses a heuristic to focus the search from a particular start location towards the goal and thus produces a path from a single location to the goal very efficiently [5]. $D*$, Incremental $A*$, and $D*$ Lite are extensions of $A*$ that incrementally repair solution paths when changes occur in the underlying graph [6]. These incremental algorithms have been used extensively in robotics for mobile robot navigation in unknown or dynamic environments. To reduce the computation time [7] suboptimal variants of the search algorithms have been used, updating existing solutions efficiently when new information is received [6], [8] to use this type of approach in dynamic environments.

The graph-based algorithms consider the image as an oriented graph in which a pixel is a node, and the 4 (or 8) connections to the neighboring pixels are the vertices of the graph. These methods can not converge to the solution of a continuous problem. Even for very fine grids, the paths restrict the agent's heading to increments of $\pi/2$ or $\pi/4$. This results in paths that are suboptimal in length and difficult to traverse in practice. Frequently, a smoothing step is performed after planning to alleviate this problem, which

yields to unsatisfactory results because it only locally solves the discrete nature of the problem. The different metrics used lead to very different results. Various efforts have been made to overcome these shortcomings, including increasing the available headings or approximating continuous paths (see [9] and [10]).

Our approach addresses this problem using the FM method. The FM method uses the L2 metric, instead of the L1 (Manhattan) of Dijkstra similar methods, and it makes an intrinsic interpolation that allows us to solve efficiently the continuous light propagation equation (the Eikonal equation) to calculate the shortest path, as shown in Figure 2. In other words, although the FM method uses data obtained from a discrete grid, it just solves the continuous light propagation equation and, therefore, the solution is also a continuous function.

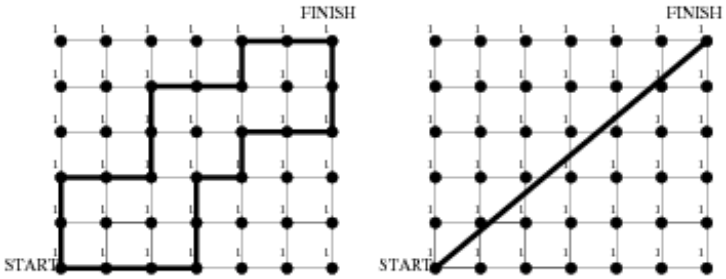

Figure 2: Dijkstra's method gives multiple short paths (left image), whilst the Fast Marching method gives the optimal diagonal path (right image).

2.2. Navigation in outdoor environments

The aim of the research presented in this paper focuses on outdoor environments. This section summarizes motion planning algorithms that can be applied outdoors.

In 1991, Mitchell and Papadimitriou [11] proposed an algorithm for determining the shortest path between a source and a destination through a planar subdivision in which each region had an associated weight. Distances were measured according to a weighted Euclidean metric: each region of the subdivision had a weight associated with it, and the weighted distance between two points in a convex region was the product of the corresponding weight and the Euclidean distance between them. In that paper, the total length of a path is defined to be the weighted sum of (Euclidean) lengths of

the subpaths within each region, and the continuous Dijkstra algorithm is used for path planning. The paper gives interesting proofs, but it is mainly theoretical and no experimental results are presented.

Instead of considering a continuous environment, some works propose the use of a discretization algorithm before the path planning. Guo et al. [12], for example, use a regular grid and the A^* algorithm in simulations. Another approach, see [13], uses a framed quadtree and the D^* algorithm for path planning. In addition to decomposing the environment, the path planning needs a cost metric for each region.

For indoor navigation, other vector field approaches that are able to compute continuous fields on a discrete environment have been proposed (Conner et al. [14] in 2003 and Lindemann and LaValle [15] in 2005). All of them could, at first, be adapted to allow outdoor navigation as proposed in this paper. However, the main motivation for seeking a new methodology, that in the end became simpler and computationally more efficient than the previous ones, was the necessity to directly and automatically incorporate some properties of the terrain into the computation, thus generating bounded speed profiles for the robot.

In our paper, we are particularly interested in using the terrain roughness, the gradient, and the height as a metric. Similarly to other outdoor navigation works, our approach decomposes the environment into a triangular grid well adapted to the characteristics of the terrain. The environment is decomposed using the Constrained Delaunay Triangulation (CDT) and then the number of faces is reduced while attempting to preserve the overall shape of the original surface. An important advantage of CDT over regular grid and quadtree representations is that it usually generates a much smaller number of cells when representing complex structures, typical of outdoor environments.

Among all the previous works, our work is closely related to Guo et al. [12], in 2003, Yahja et al. [13], in 2000, and Mitchell [11], in 1991, but with important differences. First, similarly to Guo et al. [12], our method computes cost values for each cell of a decomposed map, but our method uses a continuous version of Dijkstra's method and solves the partial differential equation of the light (Eikonal equation). Second, we work with continuous maps like in Mitchell [11], and our path is obtained on them. Only the initial data are given on discrete maps. However, instead of regular or quadtree decomposition, we decompose the environment using CDT [16]. Similar to quadtree, CDT is a non-uniform decomposition that yields in high resolution

cells in the complex regions of the environment. An important advantage of CDT over the quadtree representation is the smaller number of cells when representing non-poligonal, high complex structures, common in outdoors environments.

3. The Eikonal Equation and the Fast Marching Planning Method

In robotics, the path planner of the mobile robot must drive it in a smooth and safe way to the goal point. In nature, there are phenomena with the same way of working: electromagnetic waves. If at the goal point there is an antenna that emits an electromagnetic wave, then the robot could drive himself to the destination following the waves to the source. The concept of the electromagnetic wave is especially interesting because the potential have all the good properties desired for the trajectory, such as smoothness (that is, C^{∞}) and the absence of local minima.

The moving boundary of a disturbance is called a wave front, and it can be described by the Eikonal equation [17]. The Eikonal (from the Greek 'eikon', which means 'image') is the phase function in a situation for which the phase and amplitude are slowly varying functions of the position. Constant values of the Eikonal represent surfaces of constant phase, or wave fronts. The normals to these surfaces are rays (the paths of energy flux); thus, the Eikonal equation provides a method for "ray tracing" in a medium of slowly varying index of refraction (or the equivalent for other kinds of waves).

One way to characterize the position of a front in expansion is to compute the time of arrival T, in which the front reaches each point of the underlying mathematical space of the interface. It is evident that, for one dimension, we can obtain the equation of the arrival function T in an easy way, simply considering the fact that the distance θ is the product of the speed F and the time T.

$$\theta = F \cdot T \tag{1}$$

The spatial derivative of the solution function becomes the gradient

$$1 = F\frac{dT}{d\theta} \tag{2}$$

and therefore, the magnitude of the gradient of the arrival function $T(\theta)$ is inversely proportional to the speed:

$$\frac{1}{F} = |\nabla T| \tag{3}$$

For multiple dimensions, the same concept is valid because the gradient is orthogonal to the level sets of the arrival function $T(\theta)$. In this way, we can characterize the movement of the front as the solution of a boundary conditions problem. If speed F depends only on the position, then equation (3) can be reformulated as the Eikonal equation:

$$|\nabla T|\, F = 1. \tag{4}$$

The FM method is a numerical algorithm for solving the Eikonal equation, originally on a rectangular orthogonal mesh introduced by Sethian in 1996 [18]. The FM method is an $O(n)$ algorithm, as demonstrated in [19], where n is the total number of grid points. The scheme relies on an upwind finite difference approximation to the gradient and a resulting causality relationship that lends itself to a Dijkstra-like programming approach.

The FM methods are designed for problems in which the speed function never changes sign, and so the front is always moving forward or backward (there are no reflections, interferences, or diffractions). This allows us to transform the problem into a stationary formulation, because the front crosses each grid point only once. This conversion to a stationary formulation, in addition to a whole set of numerical tricks, gives it its tremendous speed.

Since its introduction, the FM approach has been successfully applied to a wide variety of problems that arise in geometry, mechanics, computer vision, and manufacturing processes (see [20] for details). Numerous advances have been made to the original technique, including the adaptive narrow band methodology [21] and the FM method for solving the static Eikonal equation [18]. See [20] for further details and summaries of level set and FM techniques for numerical purposes.

3.1. Algorithm Implementation on an orthogonal mesh

The FM method applies to phenomena that can be described as a wave front propagating normal to itself with a speed function $F = F(i, j)$. The main idea is to methodically construct the solution using only upwind values (the so called entropy condition). Let T(i,j) be the solution surface $T(i, j)$ at which the curve crosses the point (i, j); then, it satisfies $|\nabla T|F = 1$, the Eikonal equation.

This equation is applied to grid points, which are classified into three different types: alive, trial, and far, see Figure 3.

- Alive Points (blue points) are points where values of T are known.
- Trial Points (gray points) are points around the curve (alive points) where the propagation must be computed. The set of trial points is called narrow band. To compute the propagation, points in the narrow band are updated to alive points, while the narrow band advances.
- Far Away Points (white points) are points where the propagation was not computed yet. During the propagation, far away points are converted to trial points.

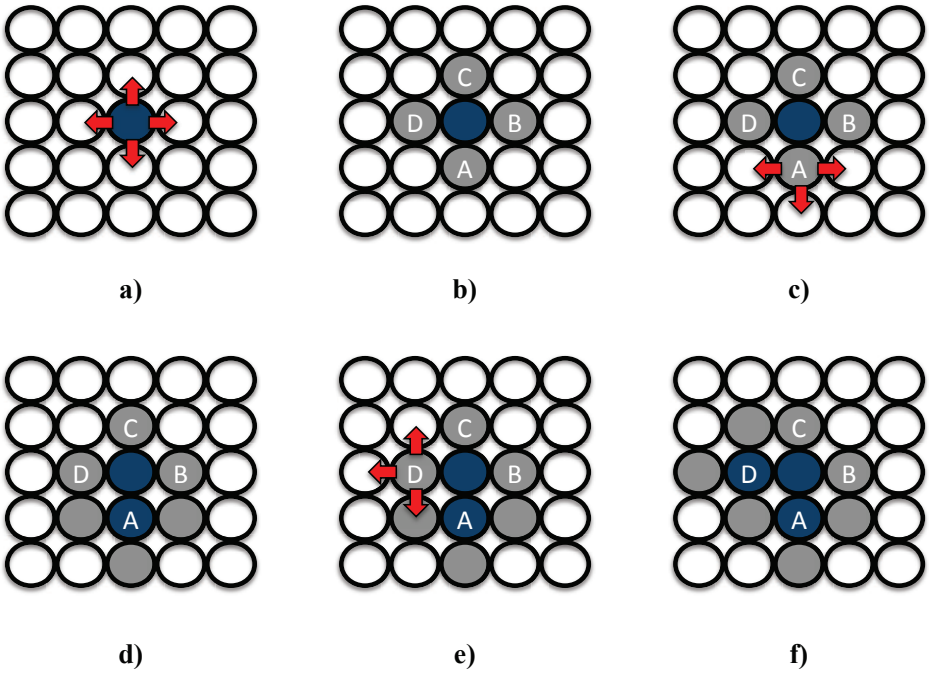

Figure 3: Scheme of Fast Marching propagation.

It is important to observe that the propagation occurs from smaller to greater values of T. Figure 3 explain this idea: the blue point (alive) represents the initial curve; in figure 3b, the value of T is computed in the neighborhood of the blue point; this neighborhood is converted from far away (white) to trial points (gray); in figure 3c, the trial point with the smallest value of T is chosen (for example, "A"); in figure 3d, the values of T are computed in the neighbors of point A, converting them from far away to trial

points. In figure 3e, the trial point with the smallest value of T is chosen (for example, "D"); in figure 3f, the neighbors of D are converted from far away to trial points. And so on.

3.2. Algorithm implementation on a triangulated mesh

As previously stated, since the work presented in this paper focuses on outdoor environments, the FM method is going to be applied on a triangulated mesh instead of on an orthogonal one. Sethian in several works [20] [22] extended the method to triangulated domains with the same computational complexity.

4. Matrix W : the viscosity map

As stated in section 1, the direct application of the FM method still has some problems. The most important one that typically arises in mobile robotics is that optimal motion plans may bring robots too close to obstacles, which is not safe.

In our approach, this problem has been solved in the same way as nature does: the electromagnetic waves, as light, have a propagation speed that depends on the media, that is the slowness index of the front wave propagation of a medium. In our case, the refraction index is defined by the viscosity map.

The proposed technique is based on the FM method, changing the speed of the wave front using a potential surface generated from the 3D environment characteristics and the robot limitations. By doing so, the method changes the time when the front reaches each point and when the generated trajectory is calculated. This trajectory is not going to be the simple geodesic, but it is going to be modified according to the robot and task needs. To be able to modify this speed, the proposed method creates a weight matrix W, which is currently built based on three main characteristics of the 3D surface: the *spherical variance*, the *saturated gradient*, and the *height*. Some other characteristics can be added to the method and it will build a different potential surface.

4.1. Spherical variance

The spherical variance [23] consists of finding the roughness of a surface to determine whether it is crossable or not. In [24] a method to calculate the roughness degree is presented. This method is based on the normal vector

deviation in each point of the surface. The spherical variance is obtained from the orientation variation of the normal vector in each point. The study uses the following reasoning:

- In a uniform terrain (low roughness), the normal vectors in a surface will be approximately parallel and, for this reason, they will present a low dispersion (see Figure 4, left).
- On the other hand, in an uneven terrain (high roughness) the normal vectors will present great dispersion due to changes in their orientation (see Figure 4, right).

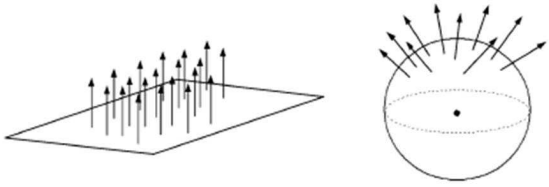

Figure 4: Spherical variance analysis. Left: A uniform surface shows parallel normal vectors. Right: An uneven surface shows normal vectors with different orientations.

The spherical variance is obtained as follows:

1. Given a set of n normal vectors to a surface, defined by their three components $\overrightarrow{N_i} = (x_i, y_i, z_i)$, the module of the sum vector R is calculated by:

$$R = \sqrt{\left(\sum_{i=0}^{n} x_i\right)^2 + \left(\sum_{i=0}^{n} y_i\right)^2 + \left(\sum_{i=0}^{n} z_i\right)^2} \tag{5}$$

2. Next, the mean value is normalized by dividing the module R between the number of data n, so the value of the result is within $[0, 1]$.

$$\frac{R}{n} \in [0, 1] \tag{6}$$

3. Finally, the spherical variance ω is defined as the complementary to the normalized mean vectors module.

$$\omega = 1 - \frac{R}{n} \tag{7}$$

When $\omega = 1$, there exists a maximum dispersion that can be considered as the maximum roughness degree, and when $\omega = 0$, a full alignment exists and the terrain will be completely flat.

4.2. Saturated Gradient

In vector calculus, the gradient of a scalar field is a vector field which points in the direction of the greatest rate of increase of the scalar field, and whose magnitude is the greatest rate of change.

Consider a surface whose height above sea level at a point (x, y) is $H(x, y)$. The gradient of H at a point is a vector pointing in the direction of the steepest slope or grade at that point. The steepness of the slope at that point is given by the magnitude of the gradient vector.

The gradient of a scalar function $f(x_1, x_2, ..., x_n)$ is denoted by ∇f or $\overrightarrow{\nabla} f$, where ∇ (the nabla symbol) denotes the vector differential operator. The gradient of f is defined to be the vector field whose components are the partial derivatives of f. That is:

$$\nabla f = (\frac{\partial f}{\partial x_1}, ..., \frac{\partial f}{\partial x_n}) \tag{8}$$

The saturated gradient consists of giving a limit value to the gradient of each point over the 3D surface. It means that, if the gradient value exceeds that limit, the point will not be included in the list of accessible points determined by the robot limitations. The gradient depends on the robot capabilities; the maximum inclination that the robot is able to cross will be the limit value for the saturated gradient.

4.3. Construction of Matrix W

As previously explained, with this matrix the algorithm can modify the path that the robot is going to follow across the 3D surface. The way the matrix modifies the path is by giving a viscosity value for each point on the surface. It means that the propagation speed of the front end of the FM wave is modified. Hence, the time when the wave reaches each point will

depend on that viscosity. We can add as many characteristics as we need to get different paths. These characteristics will modify the viscosity at each point.

The saturated gradient, the spherical variance, and the height are three matrices G, Sv, and H with the same length as the vertex matrix (the 3D mesh). The value of each vertex of the 3D grid will be determined by the calculated gradient, spherical variance, and the height of each point.

To build matrix W we give a weight or ponder factor to each surface characteristic and we can determine which one is the most important depending on the task requirements.

The values of the three matrices vary from 0 to 255, so the values of matrix W are also within this range. The components of matrix W with a value of 0 will be points in the *vertex* matrix with maximum speed. Hence, these are points which the robot can cross without any problem and at its maximum speed. The components of W with a value of 255 will be points in the *vertex* with a minimum speed. This means that the robot will not be able to pass across them.

$$W = a_1 \cdot G + a_2 \cdot Sv + a_3 \cdot H \tag{9}$$

where:

$$\sum_i a_i = 1 \tag{10}$$

The values of these ponder factors a_i are selected considering the features of the surface that want to be penalized. Those features could be penalized depending on the task requirements or the robot limitations. For example, a high value of a_1 implies that the points where the inclination of the surface (the gradient) is high will be avoided by the path.

After matrix W is generated, the method runs the FM algorithm over the modified mesh (3D mesh + matrix W) to calculate the best trajectory. With the FM method the path found will be the less time path. In the normal FM evolution, this path will be the shortest because all the points in the surface will have the same "speed" for the front propagation. With matrix W, the proposed method changes that "speed", since this matrix gives information about the difficulty to pass through each point of the surface. The trajectory will be modified depending on the surface conditions and characteristics and according to the robot limitations. Since the method modifies the "speed" of

the Fast Marching wave, it gives not only the best trajectory, but also the speed to control the robot.

5. Algorithm Simulations

5.1. Path planning on surface images

As previously stated, in the proposed method an image of the environment is needed. In relation to the outdoor environment reconstruction, there are many ways to build an environment and represent it as a $2\frac{1}{2}$ or 3D surface. This surface can be build from sensor data, elevation maps, bitmap images, etc.

The proposed method bases the surface reconstruction on bitmap images. The image contains the coordinate of each pixel and a number which indicates the value on a scale from 0 to 255. This value gives us the height of each point from the map. Since the method is working in 3D, there are 3 matrices, one for each coordinate X, Y, and Z. X and Y are the coordinates for a plain surface and Z is the height of each point. In order to create a triangular mesh, the algorithm reads the data from the bitmap file to create these three matrices X, Y, and Z and then, it builds a plain mesh based on X and Y coordinates. The algorithm generates a Delaunay triangulation [25] that is equivalent to the nerve of the cells in a Voronoi Diagram [26]. Once the triangulation is built, the next step of the algorithm is to modify the plain surface by adding the third coordinate. With this step the height of each point is changed for the real value acquired from the bitmap.

After the mesh is created, the algorithm is able to extract the needed data, the vertices and the faces of the triangles, from the matrices.

From this point, the algorithm proposed works as follows:

1. The G, Sv, and the H matrices are calculated as described in the previous section.

2. The difficulty matrix W is obtained.

3. A goal point is selected.

4. The Fast Marching potential field is calculated as a wave expansion from the initial point.

5. The path is obtained by following the direction defined by the descent gradient on the Fast Marching potential field, until the goal point is reached.

For this experiment, the image shown in Figure 5 is used. This surface corresponds to an area in the White Mountains of New Hampshire, USA. This image is a Digital Elevation Model (DEM) for the Mt. Washington quadrangle obtained from the United States Geological Survey (USGS) [27]. Following the process previously described, the algorithm is able to model the 3D surface, as can be observed in Figure 6.

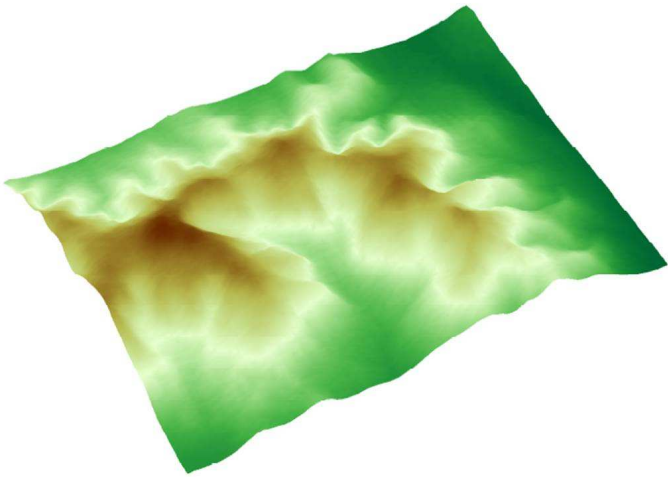

Figure 5: 3D surface built from a bitmap file

Next, several paths over the surface already presented will be obtained between the same initial and final points.

In these images, the initial and the final points have been situated over the top of the two sides of the mountain range. Those paths are obtained by varying the values of the weight factors a_i of matrix W.

Let us first show the path obtained without using the W matrix. This path, as shown in Figure 7, is the shortest path between the initial and final points since every point in the mesh has the same viscosity or difficulty value.

The fact that every point has the same viscosity value can be explained by observing the wave front propagation in this case. In Figure 8, the wave front propagation of the FM method without using matrix W is shown. The wide of each color band determines the speed of the wave front propagation.

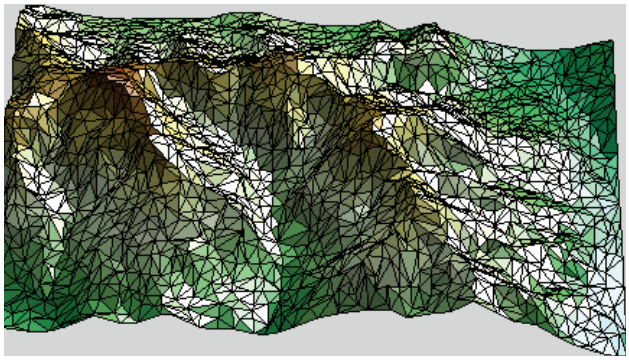

Figure 6: 3D surface with triangular mesh

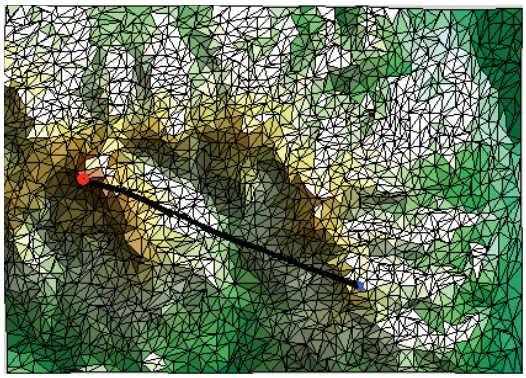

Figure 7: Geodesic calculated without using matrix W

In the point where the speed is high, the wave front expands faster and, therefore, the color band is wider. On the other hand, if the speed is limited, the wave front is slower and, as a consequence, the color band is thinner. In Figure 8, it can be observed that every color band has the same size, therefore, the wave speed is equal for every point of the surface.

In the case that $W = A$, this implies that the difficulty of the path will be determined by the height of every point of the mesh. In Figure 9, the path obtained when the height is penalized, without considering the roughness of the surface or its inclination, is presented. As can be observed, the calculated path will try to reach the final point passing trough the deepest part of the valley.

On the other hand, if we decide to calculate the path penalizing just the inclination of the surface, then the viscosity matrix is defined as $W = G$. In

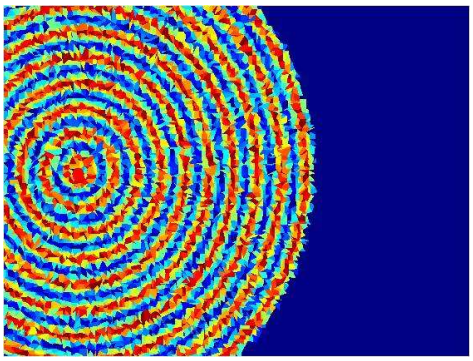

Figure 8: Wave front propagation without using matrix W

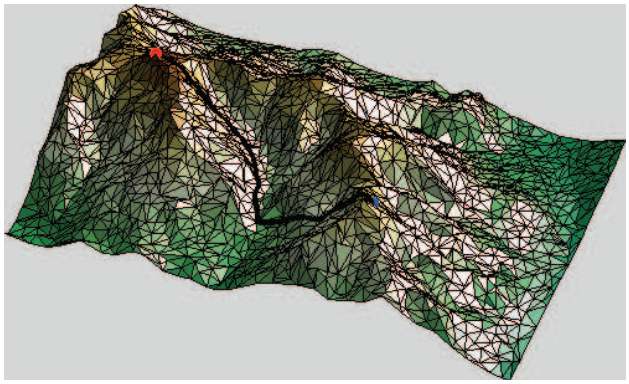

Figure 9: Path calculated when $W = A$

this case, as shown in Figure 10, the path will follow the top of the mountain chain, since, although the terrain seems very uneven, its slope is lower than the hillside one.

Another path is obtained when $W = Sv$, that is, when just the roughness of the surface is considered. In this case, the path obtained will try to follow a "soft" trajectory. In fact, as shown in Figure 11, the path is similar to the one obtained when the height is penalized. This is because, as observed in the figures, the roughness on the top of the mountain chain is much bigger than the one of the slopes of the hillsides of the valley. Therefore, the path calculated is the one expected.

Finally, the general idea proposed in this paper is the possibility of combining the three matrices in order to obtain a path that considers the height A, the roughness Sv, and the inclination G of the surface. In the previous

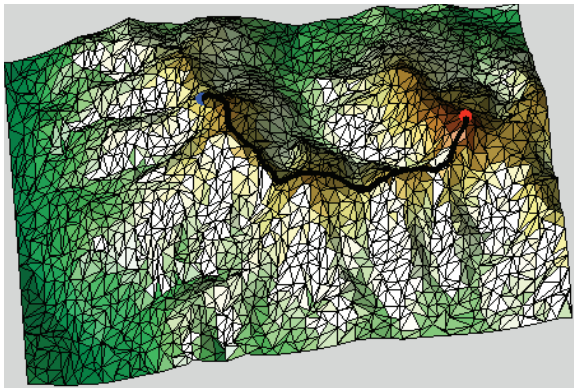

Figure 10: Path calculated when $W = G$

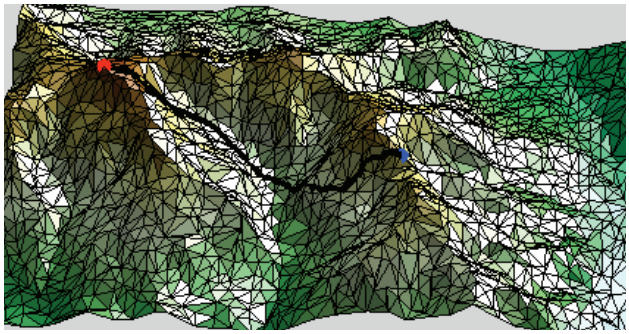

Figure 11: Path calculated when $W = Sv$

figures, it can be observed that, for the selected initial and final points, the height and the spherical variance matrices favor that the path goes down the valley. On the other hand, the gradient matrix favors the path through the top of the mountain chain. Therefore, we can select the values of each weight factor a_i in order to consider the limitations or features of the robot used.

Figure 12 and Figure 13 show two views of the same path obtained when $W = 0.15 * A + 0.05 * Sv + 0.85 * G$.

Moreover, the values of the weight factors a_i can be changed if the robot to be used is different or modified. It is also important to note that the trajectories calculated are a tentative path for the robot. The path can be modified on line by modeling the environment with the robot sensors and recalculating the trajectory in a local area.

In relation to the time consumption of the process, in table 1 the computational costs in seconds are shown. The values shown correspond to the

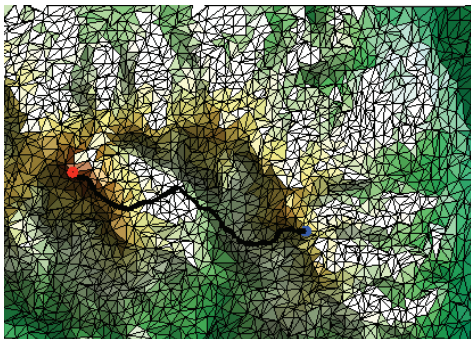

Figure 12: Path calculated using $W = 0.15 * A + 0.05 * Sv + 0.85 * G$

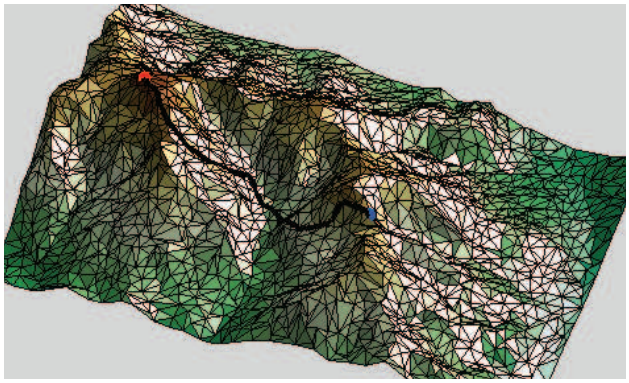

Figure 13: Path calculated using $W = 0.15 * A + 0.05 * Sv + 0.85 * G$

obtained paths between same initial and final points (over the same surface), and for different meshes (different number of vertices). As observed, the total cost of the process, which includes the time taken by the "wave expansion" phase and the path extraction, is very low (less than 1 sec), even when the number of vertices of the mesh is high. Therefore, we can consider that the algorithm proposed in this paper can be used as an on-line method.

Table 1: Computational cost (seconds)

Number of vertices	3125	5190	9300
Fast Marching time	0.05	0.09	0.18
Path Extraction	0.25	0.40	0.74
Total time	0.30	0.49	0.92

5.1.1. Test on 3D images

As shown in the previous section, the main application of the algorithm is the calculation of paths for robot navigation in outdoor environments. Another application could be its use for calculating the path for climbing robots in more complex environments. In this case, since the robot could be required to climb vertical walls, the saturated gradient is not considered in the difficulty matrix W. In order to prove the usefulness of the algorithm in this kind of surfaces, the following simulations have been made. The proposed algorithm has been used to calculate the path between two selected points situated on the 3D objects shown in Figure 14 and Figure 15.

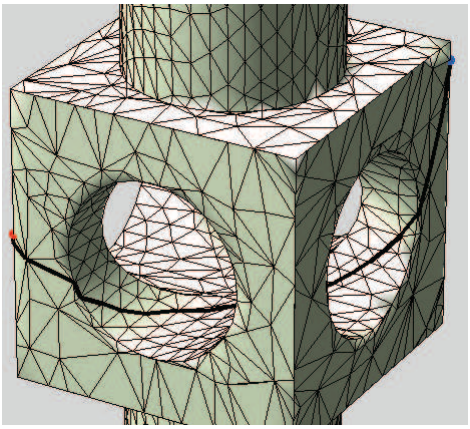

Figure 14: Path calculated on a 3D object using $W = 0.9 * A + 0.1 * Sv$

In Figure 14, the path has been calculated considering $W = 0.9 * A + 0.1 * Sv$. This implies that the difficulty matrix W gives more importance to matrix A corresponding to the height of every point. Therefore, as can be observed, the calculated path goes under the structure in order to reach the final point.

Another object is shown in Figure 15, where the calculated path also tries to go through the lower part of the object since $W = 0.8 * A + 0.2 * Sv$. In both figures, it is shown how the proposed algorithm is also useful in $3D$ objects.

5.2. Dynamic evolution of the paths of two robots approaching

Another interesting application of the proposed method is the obtaining of the dynamic trajectories of two robots which navigate approaching

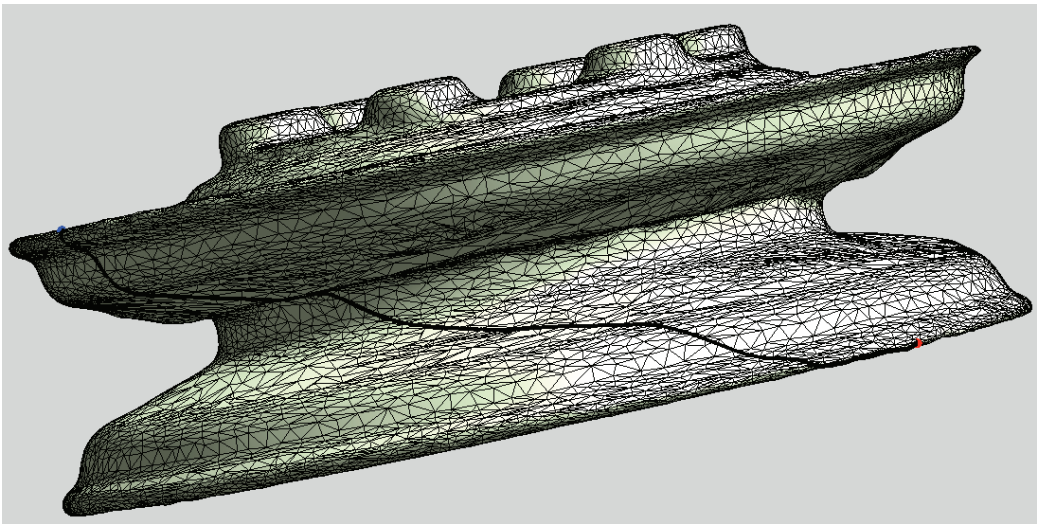

Figure 15: Path calculated on a 3D object using $W = 0.8 * A + 0.2 * Sv$

each other. In this case, another situation within the context of the project Robauco is considered. The idea is that a UAV takes an image of the surface and sends it to both robots. Then, each robot calculates the right path to get to their goals using the proposed algorithm. In order to get these trajectories, each robot maintains a map with the matrix of difficulty of the terrain. In this map, it is necessary to add a small Gaussian to the position of each robot in order to avoid possible collisions. The position of each robot can be obtained by their owns using GPS signals and, when they are separated by a reasonable distance, they communicate their respective positions by radio.

This bivariable Gaussian distribution is centered at the observed position of each robot with a standard deviation of 10 in the direction of the trajectory (longer vector), and of 3 in the second direction (shorter vector, orthogonal to the longer vector). This Gaussian has been rescaled with a factor of 10 so that it can have the necessary importance in the difficulty map.

In the initialization phase, the algorithm calculates the general difficulty matrix or viscosity map W.

Then, at each iteration, a Gaussian $G_i(k)$ is added to the observed position of each robot i. Afterwards, at each iteration, the trajectory of the robot i is calculated using the described Fast Method on the map $W + G_i(k)$, until the goal point is reached.

In the following figures the dynamic evolution of the trajectories of two

robots is shown. The goal of each robot is represented by two red dots (one of them situated in the upper-right corner and the other one in the bottom-left corner), while the current position is represented by two blue ones. The starting point of each robot, as shown in Figure 16, is almost the same goal point of the other one. Therefore, the first planned paths cross each other in several points, provoking potential collisions. As the robots move towards their goals, they approach each other. In Figure 17 we observe the situation where both robots are approaching but they are not aware of the position of the other. In this case, a collision of both robots is expected. Later, when the robots are closer, they start sensing each other and, as can be seen in Figure 18, their paths run in parallel with no collision. Finally, in Figure 19 it is observed that each robot moves away from the other till they reach their goal points. As has just been explained, this robot avoidance is made thanks to the modification of the difficulty matrix W by adding a Gaussian to the given position of the robot.

6. Conclusions

The algorithm we have presented here is a new way to calculate trajectories for moving a robot over a 3D outdoor surface. One main point about the proposed method is that it can be used not only as a Path Planning method, but also to control the robot speed to keep it within a range given by the limit speed allowed over the 3D surface, taking into account environmental characteristics and task requirements (shortest path, lower consumption, etc) reflected in matrix W. There are two values that can be attained from the results of the algorithm: the robot speed and the robot orientation. The speed is taken from the potential surface and the orientation can be taken from the next point in the trajectory that is going to be occupied by the robot. If the robot orientation and the next point where the robot is going to be are known, we can calculate the control law that has to be given to the robot in order to make it reach that next point. The most important thing about this algorithm is that it works in real time. It is really fast and gives us the possibility to use it on-line to make decisions in order to avoid fixed or moving obstacles.

7. Acknowledgment

The research leading to these results has received funding from the RoboCity2030-II-CM project (S2009/DPI-1559), funded by Programas de Actividades I+D

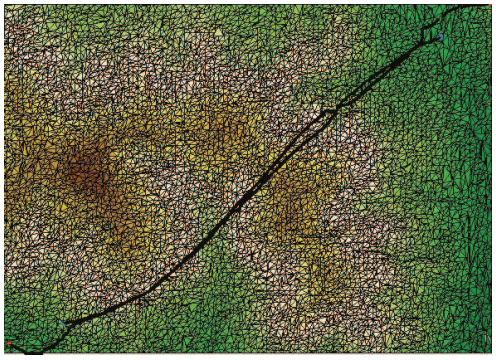

Figure 16: Starting point. Both planned paths cross.

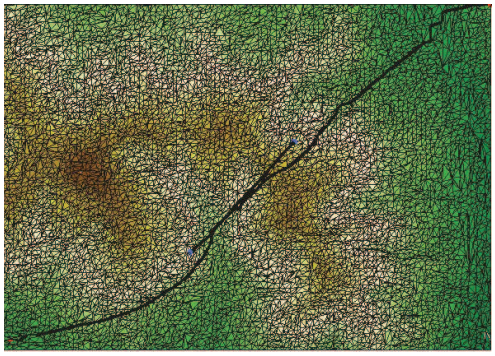

Figure 17: Robots approaching with no view of each other.

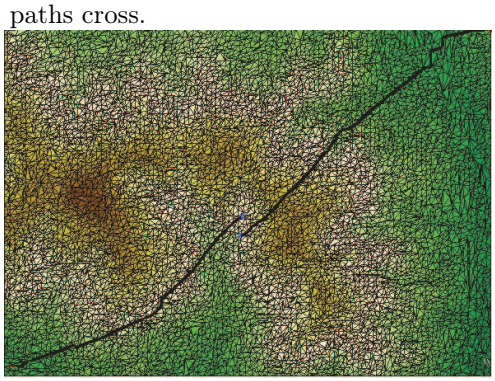

Figure 18: Robots approaching sensing their mutual position.

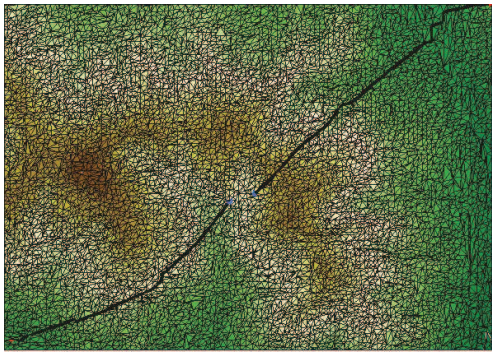

Figure 19: Robots moving away from each other.

en la Comunidad de Madrid and cofunded by Structural Funds of the EU.

References

[1] Robauco, Robauco project. <http://www.robauco.es/> (2007).

[2] S. Garrido, L. Moreno, D. Blanco, Exploration of a cluttered environment using voronoi transform and fast marching method, Robotics and Autonomous Systems 56(12) (2008) 1069–1081.

[3] S. Garrido, L. Moreno, M. Abderrahim, D. Blanco, Fm2: A real-time sensor-based feedback controller for mobile robots, International Journal of Robotics and Automation 24(1) (2009) 3169–3192.

[4] E. Dijkstra, A note on two problems in conexion with graphs, Numerische Mathematik 1 (1959) 269–271.

[5] N. Nilsson, Principles of Artificial Intelligence, Palo Alto, CA: Tioga Publishing Company, 1980.

[6] A. Stentz, The focused D* algorithm for real-time replanning, in: Proceedings of the Internatinal Joint Conference on Artificial Intelligence (IJCAI), 1995.

[7] M. Likhachev, G. Gordon, S. Thrun, Advances in Neural Information Processing Systems, MIT Press, 2003, Ch. ARA*: Anytime A* with provable bounds on sub-optimality.

[8] S. Koenig, M. Likhachev, Improved fast replanning for robot navigation in unknown terrain, in: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2002.

[9] A. Nash, K. Danie, S. Koenig, A. Felner, Theta*: Any-angle path planning on grids, in: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), p. 1177-1183, 2007.

[10] D. Ferguson, A. Stentz, Advances in Telerobotics, Springer Berlin, 2007, Ch. Field D*: An Interpolation-Based Path Planner and Replanner, pp. 239–253.

[11] J. S. B. Mitchell, C. H. . Papadimitriou, The weighted region problem: finding shortest paths through a weighted planar subdivision., Journal of the Association for Computing Machinery 38(1) (1991) 18–73.

[12] Y. Guo, L. E. Parker, D. Jung, Z. Dong, Performance-based rough terrain navigation for nonholonomic mobile robots., in: Proceedings of the IEEE Industrial Electronics Society, pp., 2003, pp. 2811–2816.

[13] A. Yahja, S. Singh, A. Stentz, An efficient online path planner for outdoor mobile robots., Robotics and Autonomous Systems 32 (2000) 129–143.

[14] D. Conner, A. Rizzi, H. Choset, Composition of local potential functions for global robot control and navigation, in: Proc. of 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003), 3546–3551, 2003.

[15] S. Lindemann, S. LaValle, Smoothly blending vector fields for global robot navigation, in: Proc. of the 44th IEEE Conference on Decision and Control, 3553–3559., 2005.

[16] J. R. Shewchuk, Applied Computational Geometry: Towards Geometric Engineering„ 1996, Ch. Triangle: engineering a 2D quality mesh generator and Delaunay triangulator.

[17] J. L. Davis, Wave propagation in solids and fluids, Springer, 1988.

[18] J. A. Sethian, Theory, algorithms, and aplications of level set methods for propagating interfaces, Acta numerica (1996) 309–395Cambridge Univ. Press.

[19] L. Yatziv, A. Bartesaghi, G. Sapiro, A fast O(n) implementation of the fast marching algorithm, Journal of Computational Physics 212 (2005) 393–399.

[20] J. Sethian, Level set methods, Cambridge University Press, 1996.

[21] D. Adalsteinsson, J. Sethian, A fast level set method for propagating interfaces, J. Comput. Phys. 118 (2) (1995) 269–277.

[22] R. Kimmel, J. A. Sethian, Computing geodesic paths on manifolds, in: The National Academy of Sciences, Vol. 95, 1998.

[23] K. Mardia, P. Jupp, Directional Statistics, Wiley Series in Probability and Statistics, 1999.

[24] Traversable region modeling for outdoor navigation, Journal of Intelligent and Robotic Systems 43 (2-4).

[25] D. T. Lee, B. J. Schachter, Two algorithms for constructing a Delaunay triangulation, Int. J. Computer Information Sci. 9 (1980) 219–242.

[26] A. Okabe, B. Boots, K. Sugihara, Spatial Tessellations: Concepts and Aplications of Voronoi Diagrams. , 1992., Chichester, UK, 1992.

[27] United states geological survey <http://eros.usgs.gov> [online] (2012) [cited 2012-09-25].