Applied Soft Computing 43 (2016) 498–509

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

# Path planning using a Multiclass Support Vector Machine

Néstor Morales*, Jonay Toledo, Leopoldo Acosta

*Departamento de Ingeniería Informática, Universidad de La Laguna, Spain*

ARTICLE INFO

*Article history:*
Received 31 January 2012
Received in revised form 14 January 2016
Accepted 24 February 2016
Available online 2 March 2016

*Keywords:*
Multiclass Support Vector Machines
Trajectory generation
Mobile robots
Path planning

ABSTRACT

In this paper, a new path planning algorithm for unstructured environments based on a Multiclass Support Vector Machine (MSVM) is presented. Our method uses as its input an aerial image or an unfiltered auto-generated map of the area in which the robot will be moving. Given this, the algorithm is able to generate a graph showing all of the safe paths that a robot can follow. To do so, our algorithm takes advantage of the training stage of a MSVM, making it possible to obtain the set of paths that maximize the distance to the obstacles while minimizing the effect of measurement errors, yielding paths even when the input data are not sufficiently clear. The method also ensures that it is able to find a path, if it exists, and it is fully adaptable to map changes over time. The functionality of these features was assessed using tests, divided into simulated results and real-world tests. For the latter, four different scenarios were evaluated involving 500 tests each. From these tests, we concluded that the method presented is able to perform the tasks for which it was designed.



## 1. Introduction

From its inception, research on mobile robots has generated great expectations. To this day the field remains wide open and is constantly evolving. Mobile robots are used in applications such as industrial processes, oceanic and planetary exploration, military projects, rescue operations, and many more. One of the most interesting areas related to mobile robotics, one that has seen considerable research since the 1960s, is path planning. This area remains largely untested, however, and many methods are still under development, with some improving on previous methods or complementing existing techniques.

The problem we wish to address with the method presented in this paper is the generation of a trajectory which allows reaching any point on a map starting from the current position of a vehicle. The challenge is to be able to do this even if the map provided is noisy and unstructured. To this end, an innovative algorithm that combines techniques typically related to Artificial Intelligence and Machine Learning is used together with classical methods from graph theory. The main idea is to employ an SVM classifier in order to generate all the possible safe paths that a robot can follow between obstacles, creating a simple, weighted graph that allows the robot to easily compute the best paths towards a given destination.

The method described in this paper uses a map as its input. Obstacles in the map are represented and transformed into point features, which are tagged using a different label for each obstacle (or class). By doing so, every feature belonging to the same obstacle will have the same label, which will be unique to each obstacle. These generated features are used to train an SVM, yielding a hyperplane that divides all these classes, ensuring that the distance from each hyperplane to the nearest feature of each class is maximized. The intersections of this hyperplane with the plane of the map are extracted and joined together to form a graph (which is then cleaned, for optimization purposes). Edges in the graph will include the value that is desired to be minimized (in our tests, length of the sub-path). Once this graph is complete, it is quite easy and fast to travel along the map.

This document is divided as follows: in Section 2 a review of previous path planning related algorithms is presented. In Section 3 a Multiclass Support Vector Machine (MSVM) is briefly explained. The different steps of the algorithm are described in Section 4 and in Section 5 we show the results of several real application experiments. At the end of the document, in Section 6, we present some conclusions and discuss the advantages of our algorithm compared with similar methods.

## 2. Previous work

Path planning is a field that has been under development since the mid-1960s, but it was not until the publication of the work by Lozano-Perez in 1979 [20] that interest in this topic started to

* Corresponding author. Tel.: +34 922 316 502x6923; fax: +34 922 318 288.
*E-mail addresses:* nestor@isaatc.ull.es (N. Morales), jonay@isaatc.ull.es (J. Toledo), leo@isaatc.ull.es (L. Acosta).

http://dx.doi.org/10.1016/j.asoc.2016.02.037
1568-4946/

grow. The problem can be divided in two stages: global and local path planning. Global path planning tries to find the best possible path from a robot's current position to its target. The physical characteristics of the robot are not usually included in this step. Local path planning is designed for a specific robot to follow the global path previously calculated, this local path being dependent on the robot's structure and capabilities. Global path planning can be divided into graph-based and grid-based techniques. Refs. [23,38] and [39] provide a compilation of some of the most relevant methods developed for each of these groups.

Graph-based path planning attempts to reduce the robot's world to a graph and then apply graph search algorithms to look for a solution. One example of these techniques is a Visibility Graph, where the robot's environment is simplified to model all the topology of the environment by linking the obstacle vertices to one another and to the initial and final nodes. Thus, the visibility graph obtains the shortest possible path as the graph is explored [49]. Another example is potential field methods, which consider the robot as a particle that is repulsed from obstacles and attracted to its destination. Potential fields are a continuous function and look for the minimum potential. A graph can then be constructed to find a global path [47].

Another classical method is the Voronoi diagram [42], where obstacles are enclosed in a 2-D polygon. The segments of the Voronoi diagram are all the points in the plane that are equidistant to the two nearest sites. The Voronoi vertex nodes are those points that are equidistant to three or more sites. The Voronoi diagram is one of the most used classical algorithms because it yields a safe path that maximizes the distance to obstacles.

Grid-based path planning divides the environment into cells, reducing the problem to a C-space (Configuration Space) search. Cells can be marked as free space, occupied space and unknown. The result of a search is a connection between adjacent cells marked as free cells from the origin to the destination. This kind of algorithm, which can integrate global and local path planning, became popular due to its use in autonomous car prototypes, like the Stanford Car for the DARPA Urban challenge [45]. It can be easily applied to real problems due to the fact that no previous knowledge of the environment is necessary. Another advantage is that they can work with partial observations of the environment.

C-space allows for multiple search algorithms, like dynamic programming where path planning is solved as an optimization problem, as in [43], where optimization techniques reduce algorithm computation time, or A*, where searching algorithms include heuristic knowledge of the route. One of the most used implementations of this kind of algorithm is the *Lattice Planner*, described in [50], which is based on performing an anytime incremental search on a multi-resolution, dynamically-feasible lattice state space. The resulting plan provides real-time performance and guarantees the suboptimality of the solution. This kind of planner allows including constraints in robot movements, as in [8], where Pivtoraiko presents a set of elementary movements that connect each discrete state value to a set of its reachable neighbors via feasible motions in a discretized field.

### 2.1. Heuristic methods

Heuristic methods are the answer given by researchers to the limitations imposed by classical methods. These limitations are time consumption in high dimensions, and the possibility of getting stuck in a local minimum. The most representative methods within this classification are: Probabilistic Roadmaps (PRM) [18], where random samples of C space are taken to find feasible paths; Rapidly Exploring Random Trees (RRT) [19], where the tree is constructed incrementally from samples drawn randomly from the search space and is inherently biased to grow towards large, unsearched areas of the space; and heuristic improvements of classical methods [37], where classical techniques like potential fields are improved using local minimum avoidance techniques, like simulated annealing.

Another class of algorithm uses Artificial Intelligence techniques to optimize the path planning cost function. These include evolutionary algorithms [41], Artificial Neural Networks (ANN) [12], Genetic Algorithms (GA) [35], Particle Swarm Optimization (PSO) [52], Ant Colony (ACO) [26] and Fuzzy Logic (FL) [13]. In general these methods do not search for a global path, looking instead for a more reactive behavior to decide on the best direction to pass an obstacle.

### 2.2. SVM methods

Within the Artificial Intelligence methods we may include the methods based on Support Vector Machine (SVM). Three main trends can be highlighted:

1. The first one uses SVM to improve or smoothen previous routes generated by other techniques, like [46], where a Voronoi-based path plan is smoothed using SVM. Ref. [6] uses SVM to smoothen a set of candidate routes in topological maps. Ref. [48] uses SVM to provide the critical points and Bezier curves are applied to smoothen the generated path. Ref. [51] uses SVM for mobile robot motion control and obstacle avoidance, and path planning is generated using potential fields based on Lyapunov functions.
2. The second one uses SVM techniques to build a path. In Ref. [24] the navigation is planned in environments for which obstacles are known. In his method, Miura randomly assigns known obstacles into two classes, positive or negative. Using these two classes, an SVM is trained and a decision boundary is used as a path. In order to make this more efficient, a set of fake obstacles (guide samples) is generated on both sides of the robot's current position and destination, as well as parallel to the line that joins both points (nominal line). This is done to help SVM search for a feasible path. The method works well, but it has two important limitations: the method is highly dependent on how well fake obstacles are placed, and tentative paths are generated using different patterns (that is, some obstacles placed in the positive class are randomly labeled as negatives and vice versa) until the number of patterns tried exceeds a predetermined number and at least one feasible path is found. This strategy does not guarantee that the final path is optimal.

   Other approaches based on SVMs are those proposed by Sarkar et al. [30], who describe the use of SVMs for navigating in known and unknown environments; and by Tennety et al. [33], who developed a method for navigating in unknown environments using SVMs and a $k$-nearest neighbor algorithm. In their methods, the entire environment sensed by the robot is divided into two classes of data sets and SVM is used to determine the maximum margin hyperplane between the data sets belonging to the two classes. The data are assigned to one class or the other, depending on whether the points are to the robot's left or right. Once the initial labels are assigned, further classification is achieved using a $k$-nearest neighbor algorithm, where $k = 1$. The idea is to assign to each point the label of the closest point already labeled. As a consequence, obstacles in the robot's environment are also classified into one of the classes based on the proximity of the obstacle points to the other points in the map. With this strategy, it is impossible to ensure that the best path for avoiding an obstacle is used, especially when more than one obstacle is present (the robot will always pass to the left or right of an obstacle, but if there is a group of obstacles, it is impossible to ensure that the robot will pass them on the shorter side).
3. Ref. [25] describes a specific application of an MSVM path-planning method and applies it to the generation of a Road

Network Definition File, which is used as global path planner for Verdino, an autonomous vehicle designed to move in closed pedestrian environments. This paper describes a complete application of an MSVM planner including obstacle inflation, obstacle clustering, navigability mask generation, SVM training, boundary extraction and the Road Network Definition File generation. A costmap is generated using the Verdino's on-board sensors and the method is tuned in order to optimize the performance of the prototype.

In the approach described in this paper, multiclass classifiers are used to obtain all the possible paths that the robot can to follow in a given environment, ensuring that the distances to obstacles are maximized, regardless of where the origin and destination points are. In this case, each obstacle is a different class in the classifier. When compared against other SVM-based methods, it improves upon them in some aspects. For example, in contrast to these methods, every possible path is explored, thus ensuring that the path obtained will always be closer to optimality and not just a tentative solution, as happened in the method developed by Miura [24], or the *k*-means approach developed by Tennety et al. [33]. In these methods, the solutions obtained by using a binary SVM are not necessarily good paths, so they cannot be used as path planning algorithms, but rather as reactive algorithms that, in some circumstances, lead to unexpected behavior. Also, for a situation in which *N* obstacles are present, the Miura method requires training up to $2^N - 2$ different SVMs to ensure that the path found is optimal. In practice, this number of combinations is not reached, since it is too expensive in terms of the time required. This leads to sub-optimal paths, or even to a valid solution not being found. However, in our method, only N different SVMs (one per class) are always calculated, ensuring that the path found is close to optimal. In addition, if a new obstacle appears in the method described by Tennety et al. [33], it is assigned to one of the classes. In the algorithm introduced in this paper, when a new obstacle appears, it is considered as a new class (see Section 5.1.2), so every possible way to avoid it is computed.

Compared to other methods presented in this section, Multiclass Support Vector Machine (MSVM) is designed to find a compromise between the distance to obstacles and the length traveled. It can be applied to continuous and discrete (C-space) spaces, converting actual robot sensor information to a graph used to calculate the shortest path. One of the main advantages of the method is its ability to reduce uncertainty due to errors introduced by sensors, since it tries to maximize the distance to the points that define different obstacles. If it is not possible to define a line that clearly divides two different objects, a function that tries to minimize this effect is generated. As noted above, it outperforms the path planning of other SVMs! (SVMs!). With respect to classical methods, the behavior is similar to Voronoi in that it tries to maximize the distance to obstacles, but in general, a lower traveled distance is achieved due to the ability of Support Vector Machine (SVM) to obtain a smooth decision boundary between classes. Paths generated by Voronoi are too similar in their obstacle profiles, so path smoothing techniques are usually necessary. C-space search algorithms, such as A*, yield soft, short paths, robot constrains can be introduced in the search and anytime methods can be used, but the resulting paths are sometimes too close to obstacles. The method presented in this paper can be used in a continuous space and finds a higher average distance to obstacles, usually at the expense of increased travel distance when compared to C-space results. In conclusion, this method is an option when safety is more important than distance, yielding shorter travel lengths than other methods designed specifically to avoid obstacles, like Voronoi.

## 3. Support Vector Machine (SVM)

Support Vector Machines (SVMs) are a set of linear classifiers usually employed both for classification and regression based on supervised learning. This technique relies on the Vapnik Chervonenkis (VC) dimension from statistical learning theory and Structural Risk Minimization. SVMs are maximum margin classifiers that yield an optimal separation hyperplane between diverse data classes. In other words, SVMs transform a linear classification of vectors into a higher dimensional space. Originally, SVMs were developed to solve data acquisition problems. They were originally applied to Optical Character Recognition (OCR) by AT&T laboratories, and later extended to machine learning and prediction problems. The original algorithm was developed by Vladimir Vapnik, but the most currently used version (soft margin) was proposed jointly by Vapnik and Corinna Cortes in 1995 [7]. This technique has been extensively used in problems involving pattern recognition and regression, multi-sensory integration, robot vision, human-machine interaction, etc. More information on Support Vector Machines (SVMs) may be found at [2,53].

### 3.1. Multiclass Support Vector Machine (MSVM)

Binary classification using Support Vector Machines is currently a well-developed technique. The generation of a solution for the multiclass classification in a single step is usually avoided. Instead, a combination of several binary SVM classifiers is used. The best-known methods are:

- *One-versus all using winner-takes-all strategy* Let $\omega_i, i \in 1, \ldots, M$ be the set of classes. The method constructs *M* binary classifiers. The output function of the *i*th classifier ($\rho_i$) is trained by taking the examples in $\omega_i$ as positive and the remaining classes as negative. Given a new sample *x*, this strategy labels it with the number of the class with the largest $\rho_i$.
- *One-versus-one using max-wins voting* In this method a binary classifier is constructed for each different pair of classes, yielding $M \cdot (M-1) / 2$ different classifiers. The binary classifier $C_{ij}$ is trained using samples from $\omega_i$ as positive and from $\omega_j$ as negative. Given a new sample *x*, if the classifier predicts that *x* is in the class $\omega_i$, then a vote is added to this class, else the vote is for the class $\omega_j$. After testing all $C_{ij}$ classifiers, *x* is assigned to the class with the highest number of votes.
- *Directed Acyclic Graph SVM (DAGSVM) [29]* This method is similar to the previous one in the training step. In the testing stage, a rooted binary directed acyclic graph with $M \cdot (M-1)/2$ nodes and *M* leaves is used. Each node contains a binary classifier $C_{ij}$. Given a sample *x*, the binary decision function is evaluated beginning at the root and then at the right or left node, depending on the output. Therefore, the final leaf node will determine the class of the sample.

There are other methods based on binary classifiers, like those based on error-correcting codes [10], the methods developed by Hastie and Tibshirani [15] and the one by Platt [28]. Other methods are able to deal with every class at the same time, like [34] and Weston and Watkins [1], which employ the decomposition method proposed in [17], and Crammer and Singer [9]. More information on More information on MSVM is available at [11,16].

## 4. Method

Given a map in which every obstacle in the robot's environment is represented, every feasible path over an obstacle-free area which maximizes the distance between objects is calculated. Other

SVM-based path-planning methods [24,33] yield a single path that connects two points in the map. In this paper, all possible paths are obtained, turning an unstructured map into a graph where edges represent different path segments and nodes their intersections. Nodes are the points at which the robot will choose between path segments depending on the desired behavior. Segment shapes are given by the decision limit between classes, which is obtained from the SVM's output.

Since a graph of feasible paths is calculated, it is possible to pick the optimal path from among all the possibilities and where optimality is defined by a specific application. This is not possible in other SVM path planning approaches where the final path is selected automatically. The path selected is not necessarily the best one and it can even lead to an inconsistent path depending on the parameters of the algorithm (for example, setting a small number of tentative paths in the Miura method [24], or obtaining a bad scenario after performing the *k*-means algorithm in Tennety et al. [33]). Also, in their methods it is difficult to include higher level decision strategies, such as, for example, always passing obstacles on the right.

MSVMs allows working with data in which measurement errors are introduced by sensors or other error sources. In other words, the method is able to generate a route despite the presence of outliers in the measurements. In fact, if the data are not linearly separable, SVM will find the hyperplane that maximizes the distance to support vectors while minimizing a function that depends on the number of incorrect classifications, also known as a function penalization term. There are four main reasons that underpin the use of SVMs as a basis for a path planning application:

1. With SVM, it is possible to generate non-linear boundaries that delimit areas on the ground, allowing for smooth paths.
2. Margin maximization allows for a safe search strategy, ensuring that paths will always be far enough away from any obstacles.
3. Using SVMs reduces the uncertainty due to errors introduced by sensors, since they try to maximize the distance to the points that define different obstacles. If it is not possible to define a line that clearly divides two different objects, a function is generated that tries to minimize this effect.

In addition, we can add a final advantage involving the use of MSVMs:

4. By using MSVMs we can generate all possible routes with optimal obstacle separation, leaving the decision of which route to use to higher-level algorithms.

The different steps of the proposed method are described next.

### 4.1. Obtaining training samples

In order to generate the paths using SVMs, a map of the obstacles in the robot's environment, in which static and dynamic obstacles are represented, must be input into the method. The difference between the current and the static map reveals new obstacles, and SVMs is only computed for new obstacles in each iteration. Old decision borders corresponding to obstacles that are no longer in the map can be erased. More information on the way in which the cost map is computed can be found in [21]. For the sake of clarity, the process by which an empty graph is generated will be explained in the sections below. In subsequent iterations, only the decision borders for those obstacles that have changed in the map are re-computed, as described in Section 4.4, but the process is the same.

Given a map with obstacles represented on it, their radius should be inflated to at least that of the robot. This ensures that paths will pass through points that are far enough away from the obstacles. The input to the SVM will be the coordinates of the points at the boundaries of these inflated obstacles, labeled according to those obstacles to which they belong.

Fig. 1 shows an input map (Fig. 1a) in which each color represents a different obstacle (or a class), and a representation (Fig. 1b) of the resulting training samples.

### 4.2. Obtaining the paths using MSVMs

Once a training set is obtained, an SVM is trained in order to define a separating hyperplane. The points at which this hyperplane cuts the two-dimensional plane define a path. In order to obtain these points, the roots of $f_i$ must be obtained (where $i$ is the number of the class or obstacle).

$$f_i = \sum_{k_i \in S_i} \alpha_{k_i} y_{k_i} K(x_{k_i}, x_i) + b_i \tag{1}$$

In this equation, $S_i$ is the set containing all the Support Vectors (SVs) for obstacle $i$; $k_i$ represents each of these SVs; $y_{k_i}$ is the label $(+1, -1)$ of each SV $k_i$; $\alpha_{k_i}$ is the Lagrange multiplier for the SV $k_i$; the set of $x_i$ are the support vectors of SVM; $K(\cdot, \cdot)$ is the Kernel function that represents the mapping of the data from the original space to higher dimensions in the non-linear SVM approach; and $b_i$ determines the offset of the hyperplane. For more information on Support Vector Machines (SVMs), please see [2].

By the end of the SVM training step, the parameters $y_i$, $\alpha_i$, $b_i$ are obtained. With this, it is quite easy to find the roots of $f_i$ by computing every possible value of $x_i$ for which $f_i = 0$.

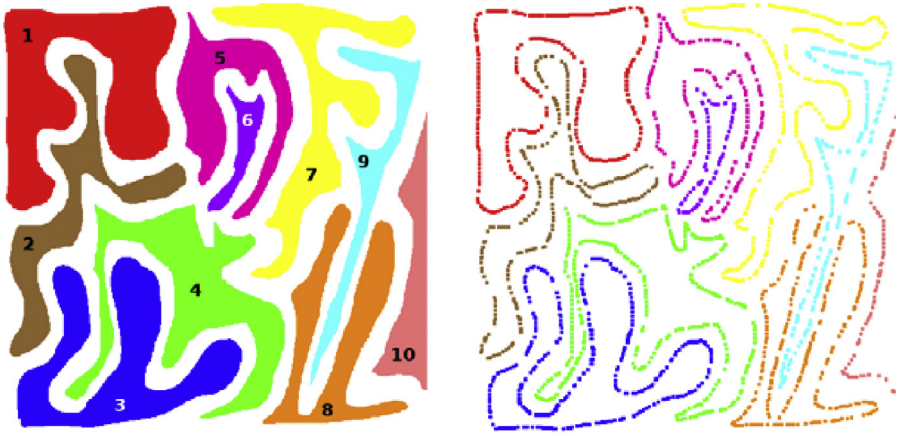
If class 4 in Fig. 1a ($i = 4$) is trained employing a *one-versus all using a winner-takes-all strategy*, in which all vectors belong to the positive class (those that belongs to class 4), or to the negative class (every other vector), a set of parameters will be obtained. If the output of Eq. (1) is represented using these parameters for every possible input value, Fig. 2a is obtained. In this figure, it is easy to verify that all the outputs from $f_4$ that are greater than 0 correspond to points that belong to class 4. Hence, the input values for which $f_4$ are 0 will define the decision limit for class 4, and it will be part of the desired path.

Of all the possible strategies, the *one-versus all using a winner-takes-all multiclass strategy* has been selected, since it allows recovering the path more easily and efficiently than other strategies, such as the *one-versus-one using max-wins voting* method, which prevents the analytical recovery of the path due to its voting-based system.

After applying this strategy, $N$ different paths will be generated, where $N$ is the number of obstacles present in the map. In other words, these are the $N$ different paths that are needed to avoid each of the $N$ obstacles. Each of these paths will maximize the distance to the points that are part of obstacles. Fig. 2c shows the calculated paths. The colors of the paths are the same as those used in Fig. 1a for the corresponding obstacles. The points shown in black are those at which the paths intersect. It is at these points where it is possible to change from one path to another.

### 4.3. Obtaining the graph

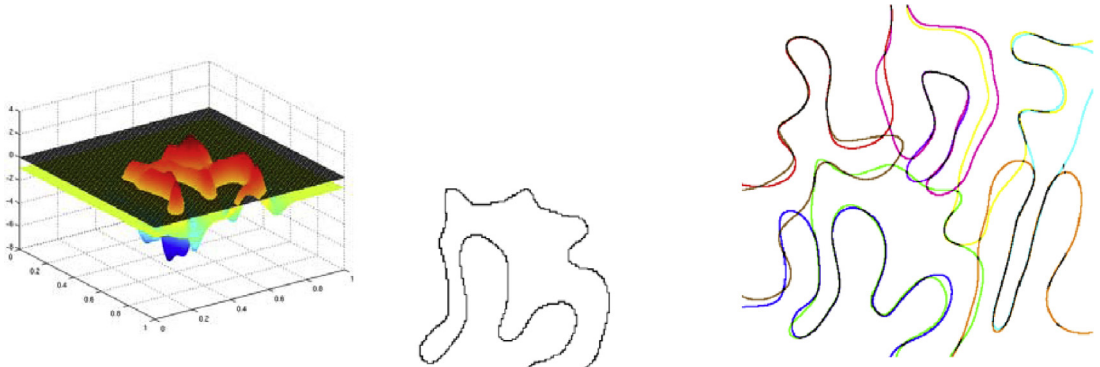
The paths calculated by SVMs allow traveling from one free point in the map to another by simply changing between paths at those points where they intersect. In order to make path planning more efficient, paths and their intersections are used to generate a non-directed graph in which path intersections are the nodes of the graph, and the path segments between intersections are its edges. These edges will have associated weights, which are initially the lengths of relative path segments, but they can also be other




(a) A map with inflated hand-generated obstacles.

(b) Points obtained from the obstacles in 1a, which will be the inputs to the training stage of the SVM.

**Fig. 1.** Input map and the resulting points.



(a) Output of every possible value from equation 1 for class 4.

(b) Decision limit for class 4.

(c) Decision limit obtained for every class.

**Fig. 2.** Representation of the decision limit retrieval process.

parameters, such as the minimal distance between a path segment and an obstacle or other values related to the robot's task.

As we can see, nodes are easily extracted, as they are the points at which paths intersect. As per Eq. (1), these are the points belonging to the set

$$\left\{x | f_i(x) = f_j(x) = 0, \quad i \in \{1, \ldots, N\}, j \in \{1, \ldots, N\}, i \neq j\right\} \tag{2}$$

Usually, a simple search of matching edges is enough; however, there are some cases, like that shown in Fig. 3, where there is more than one edge joining two nodes. In this case, different approaches can be used. The first one is to keep both edges in the graph. Another strategy is to keep only one of the edges based on some criterion, like keeping the edge with the lowest weight so as to simplify the graph. This strategy is employed in our experiments, where the edge with the shortest associated path is used.

Finally, since there are many adjacent nodes providing no useful information to the graph, the second-degree nodes are removed by joining several edges into one, thus making the graph clearer and more efficient. Fig. 4 shows a graph before and after this cleaning process. In this figure the arcs with shorter lengths are removed to simplify the graph.

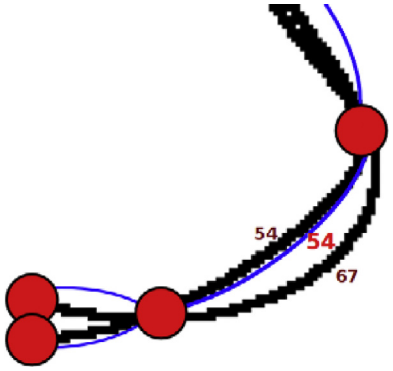
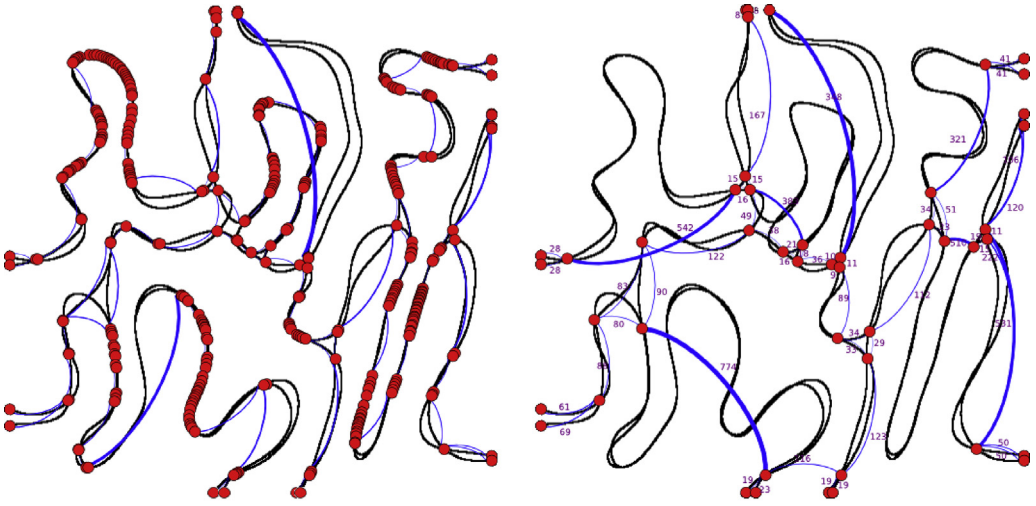



**Fig. 3.** The shortest path between two nodes is selected.

### 4.4. Non-static obstacle avoidance

The steps described above continue over successive executions. If the map remains static, however, there is no need to recompute the whole graph, and only those parts of the map that have changed from a previous execution are recomputed. While the training samples are obtained, the points used in the original execution $\mathcal{P}_o$ are compared to new ones $\mathcal{P}_c$. Only those points in $\mathcal{P}_c$ whose distance



(a) The graph before removing the intermediate nodes. (b) After removal.

**Fig. 4.** The graph before and after the intermediate nodes are removed.

to the closest point in $\mathcal{P}_o$ is over a threshold $\tau_{\underset{dist}{\min}}$ are used to update the graph.

New classes are based on $\mathcal{P}_c$ points and the SVMs are trained to yield the new paths. These new paths, together with the existing ones, are used to generate a new graph (Section 4.3). Notice that $\mathcal{P}_c$ uses the points obtained for the initial execution, since it is based on a map in which only static obstacles are represented.

## 5. Results

In this section, some path planning applications are described, showing the performance obtained by the method. This section is divided into simulated and real-world results. In the first set of experiments, the method is applied to a map computed through the imperfect segmentation of an aerial image. The second set of experiments present the real-time results obtained from our testing platform, Verdino.

The LIBSVM [4] library was used to carry out these experiments and other testing stages, together with the ROS robotic framework.[1]

### 5.1. Simulated results

In this section we present the results obtained in the simulated environment.

#### 5.1.1. Path planning in an uncertain environment

Classical path planning methods assume that the environment in which the robot is navigating is well known and perfectly modeled. By using SVMs, we can, among other things, compute paths even when the map of the environment is not well defined. In order to demonstrate the performance of the method proposed, a random aerial image was selected in which there is a group of houses and, between them, a set of roads with an irregular shape. In an effort to make things more realistic, an automatic, but not perfect, segmentation method was applied to the road extraction, meaning the result is irregularly segmented, making SVMs useful to tackle this real example.

The algorithm will use as its input the blocks of houses between the roads. These obstacles are irregularly shaped. In some cases, parts of the image labeled as road should actually be considered as an obstacle, and in other parts pixels labeled as houses belong to a road. Fig. 5a shows the original sub-optimal segmentation of the roads in the map.

In the automatic segmentation step, some regions are incorrectly classified, which could generate twisting or impossible paths with classical methods. Using the proposed method, the resulting paths should be able to ignore this kind of error. The points input to the algorithm are extracted from these regions (Fig. 5b) and used to obtain boundaries (Fig. 5c).

The two columns on the right side of the Fig. 5c show some interesting details. To make the visualization easier, the output of the SVM training step is shown in the first column with the original image in the background, while the second column depicts the same area showing the input data used (in a darker color inside the resulting regions). The first example shows a three-way intersection. As seen discussed, the paths that will be generated correspond to borders of the colored areas of column 1, and the points at which these paths cross will be the nodes of the output graph.

In the second row, a similar situation is shown with three-ways intersection, but in this case the generated path passes through the badly segmented crosswalk. Something similar happens in the third example, in which there is a signal painted in the road that has been badly segmented, though the algorithm is able to output a smooth path. In the last row, an obstacle (a vacant lot) has been poorly segmented, with parts of the obstacle labeled as a road, but the algorithm manages to compute a good path.

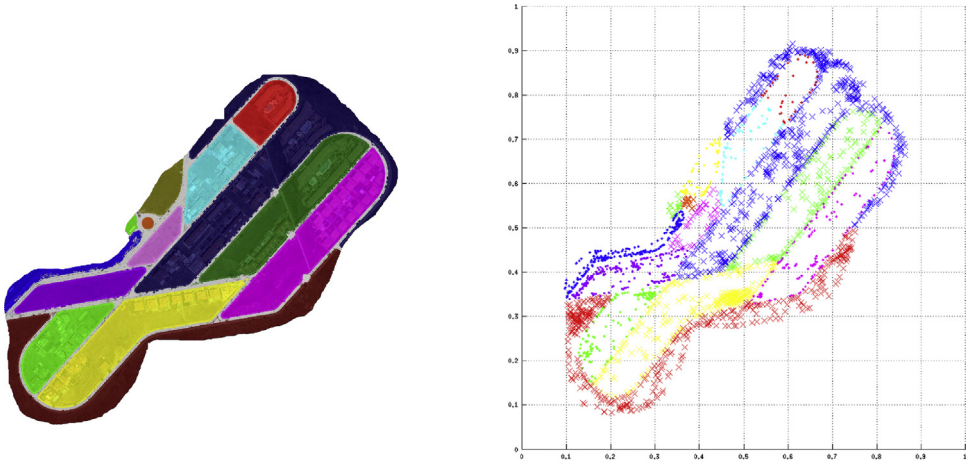
#### 5.1.2. Updating the path in the presence of a moving obstacle

One of the advantages of the proposed method is its adaptability. To demonstrate this, the behavior of the algorithm in a situation in which there is a non-static simulated obstacle is described. In order to make this situation more easily understandable, a simulation based on the roundabout that appears in the previous example will be used, a magnified image of which is shown in Fig. 6.

Fig. 6 shows the initial path graph calculated for the static obstacles in the roundabout. In this image, the various paths that can be taken by the robot are in black, and the nodes where it is possible to change from one path segment to another are represented with green circles.

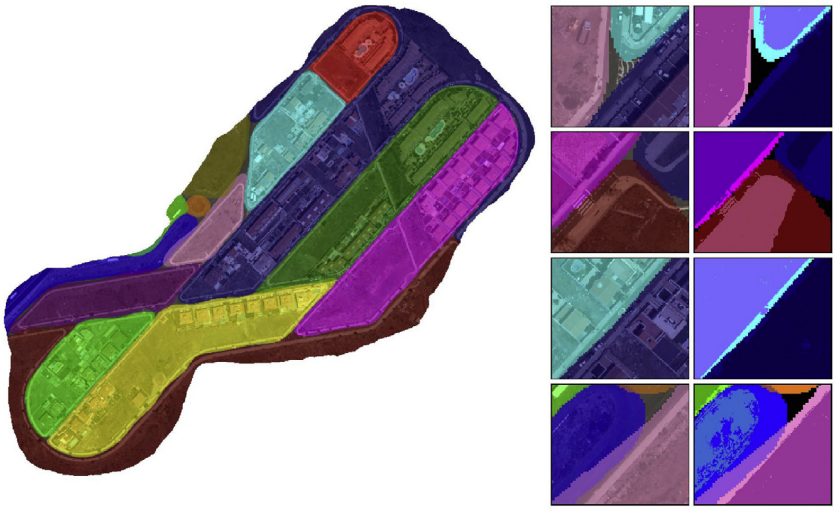
Once this initial path graph is obtained, a recalculation process is carried out for each iteration. In the sequence shown in Fig. 6, a new obstacle (a car) appears, represented in red. For each new

[1] http://www.ros.org/.



(a) Sub-optimal segmentation.

(b) Algorithm input data.



(c) Output of the SVM training step.

**Fig. 5.** Segmentation, input training points and output for the example provided.

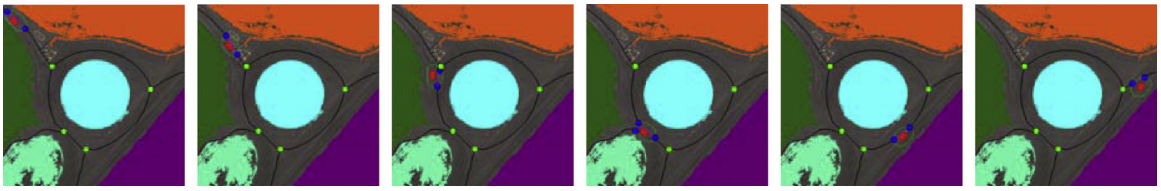


**Fig. 6.** Sequence of a vehicle driving in the vicinity of a roundabout.

obstacle a binary SVM is trained, so points belonging to the obstacle become part of the positive hypothesis and the total number of points belonging to the remaining obstacles (both static and dynamic) are the negative hypothesis. The method for including new obstacles is described in Section 4.4.

The decision limits of the binary SVM trained with the car are the possible paths the robot can use to avoid the obstacle. These paths are combined with the previously generated paths. The points at which the paths cross one another are located and added to the graph. At these points the path is segmented and those nodes can be used to change from the old path segments to the new ones (and vice versa) are identified. In this process, the originally calculated path segments close to the new obstacle are changed. The original graph arcs are infinitely weighted, and new arcs avoiding the obstacle are included after the SVM training. These new arcs are connected with the previous graph at their interceptions. Then, the chosen shortest path algorithm (in our case, Dijkstra's) is used to compute the global path. When the dynamic obstacle disappears, the new arcs generated to avoid it are removed from the graph.

### 5.2. Real-world results

In this section, we will describe the results from the experiments performed using our testing platform, Verdino.

#### 5.2.1. Testing configuration

Our testing platform is an autonomous robotic prototype called Verdino.[2] This platform is an electric vehicle used for to transport people in closed, unstructured areas (such as residential areas,

[2] http://verdino.webs.ull.es.

**Fig. 7.** Testing platform.

parkings and industrial areas), at a maximum speed of 25 km/h. It has been modified to drive autonomously, operated by a computer installed on board. To this end, the original steering system, brakes and accelerator have been modified, with a variety of sensors mounted on it, including Light-Detection And Rangings (LIDARs) and a localization and a vision system [40].

Verdino, shown in Fig. 7, is designed to transport people autonomously in closed, unstructured areas. Because of this, its behavior must be mainly reactive in order to give priority to path safety over path efficiency. We considerer two different planning levels: global and local. The latter uses an approach based on the Frenét space, which is beyond the scope of this paper. For more information on this topic, see [6].

Our prototype works in unstructured areas where it is not possible to determine a global trajectory based on a pre-generated road map, since no such map is available. But a map of the static obstacles has been previously computed for the area using the method in [27]. The map is computed by adjusting the information retrieved over time from the set of LIDARs units installed on board, so it is quite noisy. Examples of these maps are shown in Figs. 8 and 9.

### 5.2.2. Dynamic obstacle avoidance

In Fig. 8, a dynamic obstacle (a group of persons) appears on the road when the vehicle turn around a corner. The algorithm detects that the obstacle was not there when the map was captured. Then, this obstacle is added to the set of obstacles used in the MSVM training stage. Using the decision border, the graph is completed so the vehicle is able to avoid it. The rest of the path remains unaltered.

### 5.2.3. Evaluating the generated path

In this section a comparison between four path planning methods is presented. The methods compared are: *Multiclass SVM* (presented in this paper), the method developed by Miura [24] (referred to as *Single SVM*), an implementation of Voronoi diagrams, one of the most widely applied classical path planning algorithms (referred to as *Voronoi*), and the available implementation of the SBPL lattice planner [50], a grid method based on a variation of the A* search algorithm in C space, referred to as *SBPL A**. Each

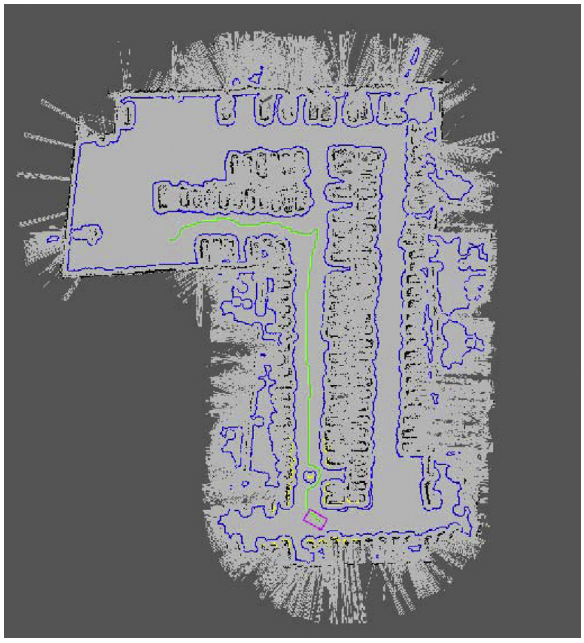
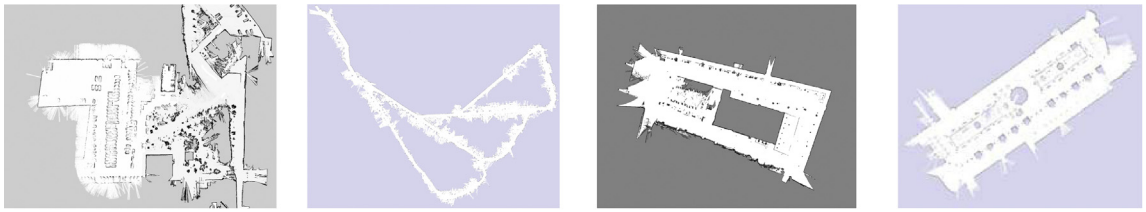


**Fig. 8.** Dynamic obstacle avoidance.

method represents one of the most used algorithms in its category. The results were obtained in actual conditions using real-time information provided by the on-board LIDARs and the localization algorithm described in [27].

In order to assess the differences between the four methods considered, they have been applied to four different real-world scenarios in which maps have been generated using our testing platform. The resulting maps are shown in Fig. 9: the ETSII map (Fig. 9a) corresponds to an extended version of the map shown in Figs. 8 and 11, obtained in our building's parking lot, which measures 201.40 m × 150.40 m; the ITER map corresponds to the gated residential area in which our prototype is usually tested, which measures 467.85 m × 377.50 m; the Tegueste and Orotava maps were obtained in the pedestrian streets of two nearby towns, and measure 146.64 m × 105.99 m and 103.25 m × 86.25 m, respectively.

To evaluate the methods, a total number of 500 tests were performed on each map. Each test involved randomly selecting an initial position and a destination and storing them. These same positions were then used to evaluate all four methods. The same experiment was repeated in similar conditions for each approach, ensuring a fair comparison of the different algorithms without biasing the results. Also, the points detected for real dynamic obstacles were recorded and timestamped, so the vehicle always encountered the same obstacles at the same time. The evaluation values were stored during the execution. These values were:



(a) ETSII (201.40 × 150.40 *m.*) (b) ITER (467.85 × 377.50 *m.*) (c) Tegueste (146.64×105.99 *m.*) (d) Orotava (103.25 × 86.25 *m.*)

**Fig. 9.** Maps used for the tests shown in this section, and their sizes.

**Table 1**
Results obtained for the various algorithms and using different maps. The average of the values measured in every test is shown, as well as the standard deviation (in parentheses).

| Map | Algorithm | Path dist. [%] | Min. distance [m] | Avg. curvature [°] | Time [ms] | Success [%] |
|---|---|---|---|---|---|---|
| ETSII | Multiclass SVM | 100.00 (0.00) | 5.90 (4.65) | 8.63 (2.28) | 391.06 (98.32) | 100.00 |
| | Single SVM | 98.16 (34.21) | 1.57 (1.62) | 7.34 (1.52) | 2045.44 (624.15) | 45.60 |
| | Voronoi | 103.24 (4.39) | 7.22 (5.93) | 21.62 (12.23) | 443.21 (129.70) | 100.00 |
| | SBPL A* | 95.83 (6.91) | 3.55 (1.27) | 10.22 (7.05) | 497.76 (204.11) | 96.00 |
| ITER | Multiclass SVM | 100.00 (0.00) | 2.10 (0.58) | 10.73 (8.72) | 355.96 (105.27) | 100.00 |
| | Single SVM | 99.87 (55.43) | 0.92 (0.34) | 5.01 (0.00) | 5139.06 (1023.77) | 9.80 |
| | Voronoi | 106.67 (7.43) | 6.95 (1.82) | 17.29 (10.19) | 623.38 (297.01) | 100.00 |
| | SBPL A* | 90.56 (9.31) | 1.51 (0.53) | 14.11 (10.84) | 404.80 (114.20) | 83.20 |
| Tegueste | Multiclass SVM | 100.00 (0.00) | 5.71 (1.92) | 27.93 (1.20) | 104.94 (12.41) | 100.00 |
| | Single SVM | 97.21 (28.70) | 3.75 (3.75) | 15.30 (2.01) | 1099(810.25) | 36.00 |
| | Voronoi | 101.79 (8.20) | 5.07 (2.68) | 44.82 (11.47) | 176.3 (32.24) | 100.00 |
| | SBPL A* | 100.49 (11.42) | 4.07 (3.39) | 39.08 (16.44) | 306.84 (60.87) | 97.80 |
| Orotava | Multiclass SVM | 100.00 (0.00) | 1.85 (0.22) | 23.54 (7.35) | 157.41 (11.38) | 100.00 |
| | Single SVM | 96.10 (21.02) | 1.22 (0.61) | 11.69 (0.74) | 1026(601.08) | 21.00 |
| | Voronoi | 102.42 (9.10) | 4.56 (2.88) | 31.92 (5.14) | 235.85 (11.43) | 100.00 |
| | SBPL A* | 101.34 (14.11) | 1.78 (0.40) | 24.46 (18.93) | 328.14 (351.02) | 98.2 |

- *Path length* ($l = \sum_{j=2}^{N} \|p_j - p_{j-1}\|$), where $N$ is the number of points in the trajectory and $p_j$ represents each of these points. This is the length of the computed path. This value is measured in meters.
- *Minimum distance to the obstacles* ($\bar{d} = \min(d_j) \mid d_j = \|p_j - nearest\_obst(p_j)\|, \quad j = 1, \ldots, N$), where $d_j = \| p_j - nearest\_obst(p_j) \|, j = 1, \ldots, N$. $d_j$ measures the Euclidean distance between a point $p_j$ in the path and its closest obstacle $nearest\_obst(p_j)$. The measured value corresponds to the minimum distance $d_j$ for every point in the path, so the higher this measured value is, the better. This value is measured in meters.
- *Curvature* ($c = \angle(p_i, p_{i+1}) - \angle(p_i, p_{i-1})$), where $\angle(a, b)$ is the angle between $a$ and $b$. This measurement shows the angular difference between the tangents of two successive points in the path. This value is measured in degrees.
- *Times* This is the time required to compute the path, in milliseconds.
- *Success* This is the percentage of tests performed for which a path was found.

All of these tests were executed on a i7-3630QM PC with 8GB RAM and a NVIDIA GT 640M, which gave the values shown in Table 1. The average value measured in every test is shown, as well as the standard deviation (in parentheses).

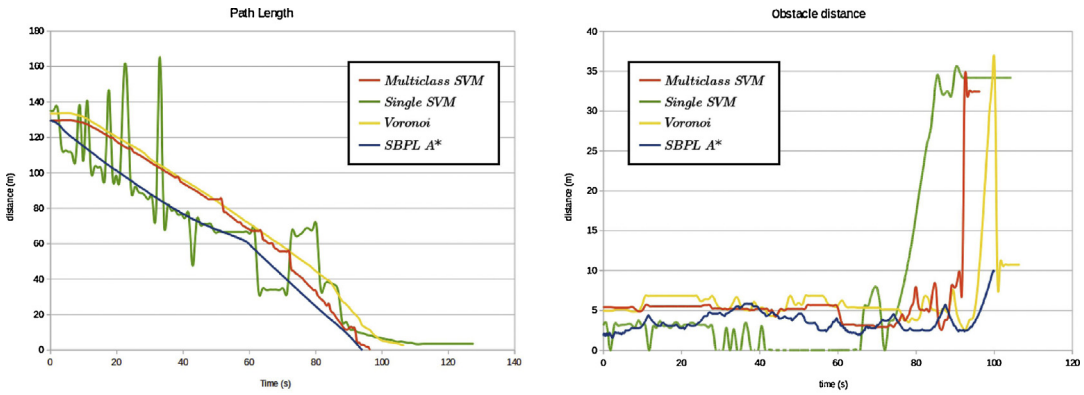
The first column of Table 1 (Path Dist) shows a path length comparison between *Multiclass SVM* and the other methods. These values were obtained as follows: for each test involving a certain initial position and a destination, the path length value was divided by the path length previously obtained with our method (and multiplied by 100.0), which is used as a baseline. This allows comparing the paths generated by the two methods as a ratio (in %). The values shown in the table are the average of the ratios obtained for all the tests, and their corresponding standard deviation. For those methods where paths were not generated in every test, only the successful tests were considered. The path lengths are quite similar in every case, but generally the *SBPL A** provides the shortest paths. However, in the last two maps, since the space available was narrower, some directional adjustments were needed, which increased the path length in some tests. *Single SVM* is too unpredictable due to its random nature, but in general it gives longer paths of varying lengths between tests, even if they look similar. Finally, *Voronoi* is the closest to our method in terms of path length, though it still yields longer paths.

Based on the minimum distance to obstacles (Min. Distance), the worst method is *Single SVM*, followed by *SBPL A**. A possible reason for this is that *Single SVM* tries to optimize the whole map, so it is very difficult to strike a balance between approaching one obstacle or another. The *SBPL A** is restricted to the underlying primitives used to construct the path, meaning it sometimes approaches the borders, especially in sharp turns. The best results were generated by *Voronoi*, though the results using *Multiclass SVM* are quite competitive as well. However, if the average curvature is considered (Avg. Curvature), then *Voronoi* exhibits the worst results. This is because *Voronoi* looks for the midpoints between the obstacles, but smoothing is not applied to the resulting paths, leading to sharp trajectories with many acute angles. In some tests the *SBPL A** includes motion sense changes, which increases the measured curvature.

The time column shows that *Multiclass SVM* is faster than *Single SVM* because *Single SVM* requires comparing several random combinations of obstacles until a path is found, which increases the time variability. If a good path is found in one of the first combinations, this method finishes in a short time; if not, it can take too long. Our method is quite constant in the sense that the number of SVMs being trained is always the same (with some extra computation if a new obstacle appears). The resulting times were around 400 ms in the worst cases, which is fast enough for a local planner like the one described in [6]. These times are for updating the graph when a new dynamic obstacle is present, or when the graph is computed again, but paths can be found faster by simply accessing the last updated graph. Some parts of the algorithm have been optimized using code parallelization, including the use of the GPU. *Voronoi* is slightly slower, followed by *SBPL A**, whose times depend mainly on the cell size and distance. However, these last two methods have not been parallelized, although their calculation times are reasonable for the application at hand.

The path length differences between algorithms are studied in detail using the ETSII map in Fig. 10a. This chart shows the reduction in path length as the vehicle approaches its destination. The shortest path is obtained using *SBPL A**, since A* tries to find the shortest free-cell path using a distance heuristic (ignoring other restrictions like the distance to obstacles). In the case of *Multiclass SVM*, the decrease is monotonic, with only small variations due mainly to the presence of obstacles in the way. However, in the case of *Single SVM*, some peaks are visible in the plot. These peaks stem from random changes in the path, which are inherent to the random way in which the trajectory is computed. The *Voronoi* path length tends to decrease over time, but it is usually longer than that output by the other algorithms.

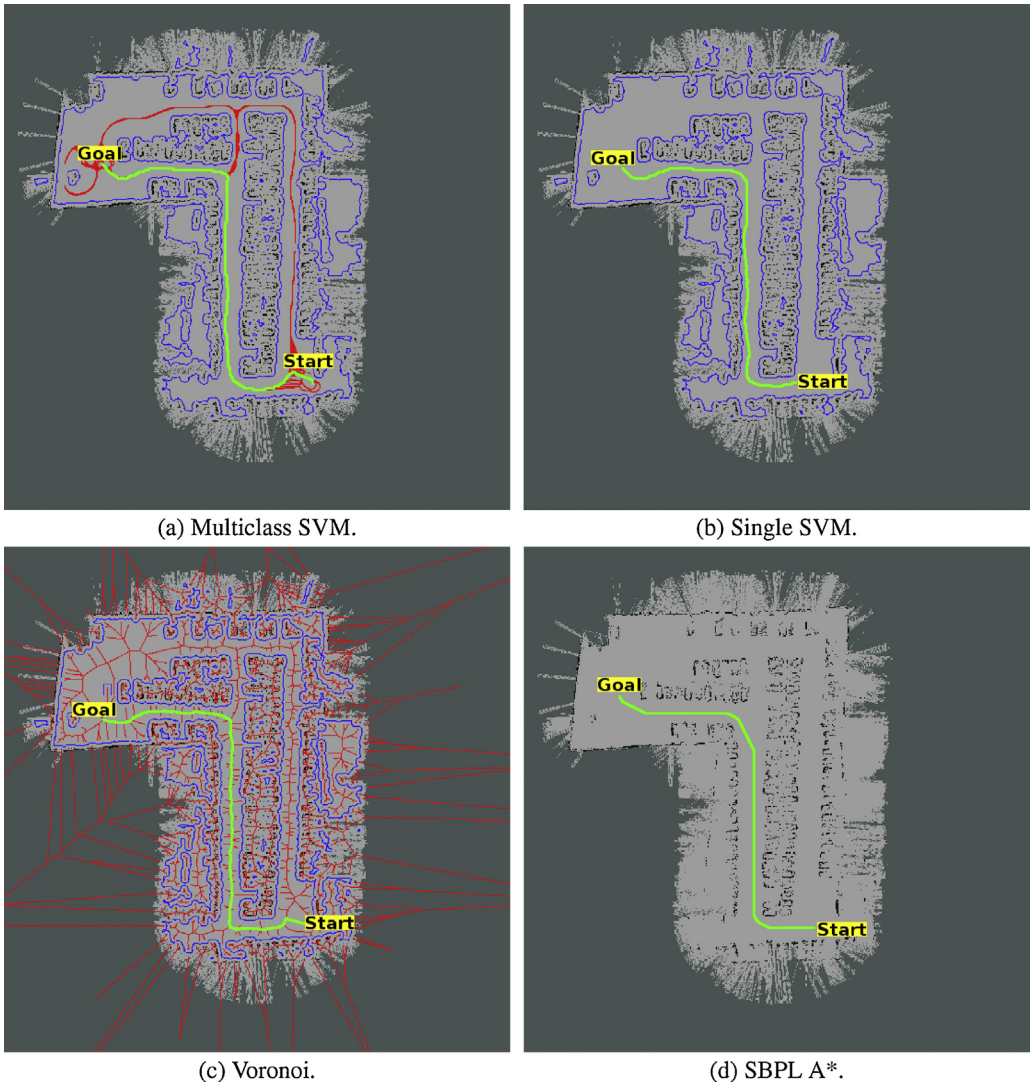
The minimum distance to obstacles is shown in Fig. 10b for the same test. *Voronoi* maintains the longest distance to objects as it




(a) Length comparison. (b) Distance to obstacles comparison.

**Fig. 10.** Results for a random single test on the ETSII map.

looks for the midpoint between obstacles. *Multiclass SVM* is the second method in obstacle distance, achieving better results than *SBPL A**. The worst method is *Single SVM*, which tends to be too close to obstacles. Distances for *Multiclass SVM* are always quite constant and above 5 m. At the end, these values are higher since the robot approaches the destination in an obstacle-free area. For the *Single SVM* approach, there are some considerable distance changes as the robot reaches its target. This is due to path modification caused by the random path computation. In general, the distance values obtained for *Single SVM* are closer to obstacles, falling even below 0.5 m. The reason is that since all the obstacles remain in one class or the other, the decision border is different from that in the MSVM




(a) Multiclass SVM. (b) Single SVM.

(c) Voronoi. (d) SBPL A*.

**Fig. 11.** Examples of the paths generated by the four methods evaluated.

approach. This decision border is optimal at the global level but not at the local level, resulting in the short obstacle distance, as depicted in the chart.

Fig. 11 shows a comparison of the paths calculated by *Multiclass SVM*, *Single SVM*, *Voronoi* and *SBPL A** in a smaller version of the ETSII map. The obstacles defined in this map (mostly parked vehicles) are shown in black, while the traversable area is in light gray. The path generated is shown in green. The inflated obstacles are delimited by the blue lines. All the algorithms are able to generate a path between the origin and destination, but the smoothness and length of the path can change significantly. *Multiclass SVM* generates a smooth path of reasonable length. *Single SVM* can generate a path but sometimes the distance to obstacles is too small. The direction of the path is selected randomly, meaning the path selected is not always the shortest one. *Voronoi* stays a good distance away from objects but paths tend to be less smooth at the local level. *SBPL A** finds a short, continuous path to the destinations, but it tends to approach obstacles in corners in an effort to reduce the path length.

## 6. Conclusions

This paper describes a method that applies Multiclass SVM to robot path planning. Different tests demonstrate that the method is able to find a smooth path, if it exists, while maximizing the distance to obstacles. By generating a non-linear continuous surface whose distance to the support vectors is maximized, the paths can be calculated simply and directly. Also, the effect of the presence of outliers or noise in sensor readings can be reduced.

The method is ready to be applied to real-world situations; however, some drawbacks have been identified and future areas of research have been proposed to improve this method.

- When there is a large gap between two obstacles, the trajectories calculated may not touch one another, failing to complete the graph and possibly leading to the generation of isolated, unreachable paths. One possible solution for this is to create visibility graphs for isolated paths such that they can be reached in these cases.
- Sometimes the change between paths is not as smooth as it could be. In order to solve this, changes between paths could be solved through other soft computing techniques.
- Primitives are not being considered, nor are course changes. One possible solution is to adapt the technique shown in *SBPL A** by directing the A* search through the existing graphs.
- The method limits the number of paths possible since it regards the vehicle's footprint as perfectly circular, with a diameter equal to its longest dimension.

Although there are other path planning approaches based on SVM, the authors have not found any approach that uses MSVM for this purpose. MSVM creates a graph in which every possible path is represented for fast access, meaning paths can be generated easily by using a shortest path algorithm. The method is also compared to three other methods and shown to strike a good balance between path length, smoothness and distance to obstacles. Our method is also more constant since it is more deterministic and the paths generated do not depend on a random configuration of the obstacles, as is the case in other approaches, like the *Single SVM*. A comparison with other standard methods such as *SBPL A** or *Voronoi* shows that *Multiclass SVM* achieves a better compromise between the measured values, resulting in a good path planning candidate.

Finally, our method has been tested both in a simulated environment and in real conditions, using our Verdino robotic platform. These tests have proven that it is able to work in real time for its intended purposes.

## Acknowledgements

This work was funded by the STIRPE DPI2013-46897-C2-1-R project and by the Agencia Canaria de Investigación, Innovación y Sociedad de la Información (ACIISI), cofinanced by FEDER funds (EU).

## References

[1] S. Abe, Support Vector Machines for Pattern Classification, Springer, 2005.
[2] C.J. Burges, A tutorial on support vector machines for pattern recognition, Data Min. Knowl. Discov. 2 (1998) 121–167.
[4] C.C. Chang, C.J. Lin, LIBSVM: a library for support vector machines, ACM Trans. Intell. Syst. Technol. (TIST) 2 (2011) 27.
[6] K. Chu, M. Lee, M. Sunwoo, Local path planning for off-road autonomous driving with avoidance of static obstacles, IEEE Trans. Intell. Transp. Syst. 13 (2012) 1599–1616.
[7] C. Cortes, V. Vapnik, Support-vector networks, Mach. Learn. 20 (1995) 273–297.
[8] M. Pivtoraiko, R.A. Knepper, A. Kelly, Differentially constrained mobile robot motion planning in state lattices, J. Field Robot. 26 (2009) 308–333.
[9] K. Crammer, Y. Singer, On the learnability and design of output codes for multiclass problems, Mach. Learn. 47 (2002) 201–233.
[10] T. Dietterich, G. Bakiri, Solving multiclass learning problems via error-correcting output codes, J. Artif. Intell. Res. (1995) 263–286.
[11] K.b. Duan, S.S. Keerthi, Which is the Best Multiclass SVM Method? An Empirical Study, 2005.
[12] J. Fan, M. Fei, S. Ma, RL-ART2 neural network based mobile robot path planning, in: in: Sixth International Conference on Intelligent Systems Design and Applications, IEEE, 2006, pp. 581–585.
[13] M.P. Garcia, O. Montiel, O. Castillo, R. Sepúlveda, P. Melin, Path planning for autonomous mobile robot navigation with ant colony optimization and fuzzy cost function evaluation, Appl. Soft Comput. 9 (2009) 1102–1110.
[15] T. Hastie, R. Tibshirani, et al., Classification by pairwise coupling, Ann. Stat. 26 (1998) 451–471.
[16] C.W. Hsu, C.J. Lin, A comparison of methods for multiclass support vector machines, IEEE Trans. Neural Netw. 13 (2002) 415–425.
[17] C.W. Hsu, C.J. Lin, A simple decomposition method for support vector machines, Mach. Learn. 46 (2002) 291–314.
[18] L. Kavraki, P. Svestka, J.C. Latombe, M. Overmars, Probabilistic roadmaps for path planning in high-dimensional configuration spaces, IEEE Trans. Robot. Autom. 12 (1996) 566–580.
[19] S.M. LaValle, J.J. Kuffner Jr, Rapidly-exploring random trees: progress and prospects, in: Algorithmic and Computational Robotics: New Directions, 2000.
[20] T. Lozano-Pérez, M.A. Wesley, An algorithm for planning collision-free paths among polyhedral obstacles, Commun. ACM 22 (1979) 560–570.
[21] D.V. Lu, D. Hershberger, W.D. Smart, Layered cost maps for context-sensitive navigation, in: in: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2014), IEEE, 2014, pp. 709–715.
[23] E. Masehian, D. Sedighizadeh, Classic and heuristic approaches in robot motion planning – a chronological review, World Acad. Sci. Eng. Technol. 23 (2007) 101–106.
[24] J. Miura, Support vector path planning, in: in: 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2006, pp. 2894–2899.
[25] N. Morales, J. Toledo, L. Acosta, Generating automatic road network definition files for unstructured areas using a multiclass support vector machine, Inf. Sci. 329 (2016) 105–124, Special issue on Discovery Science.
[26] C. Mou, W. Qing-xian, J. Chang-sheng, A modified ant optimization algorithm for path planning of UCAV, Appl. Soft Comput. 8 (2008) 1712–1718.
[27] D. Perea, J. Hernandez-Aceituno, A. Morell, J. Toledo, A. Hamilton, L. Acosta, MCL with sensor fusion based on a weighting mechanism versus a particle generation approach, in: in: 16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013), IEEE, 2013, pp. 166–171.
[28] J. Platt, Probabilistic outputs for support vector machines and comparison to regularized likelihood methods, Adv. Kernel Methods Support Vector Learn. (2000) 61–74.
[29] J.C. Platt, N. Cristianini, J. Shawe-taylor, Large margin DAGs for multiclass classification, in: in: Advances in Neural Information Processing Systems, 2000.
[30] S. Sarkar, E.L. Hall, M. Kumar, Mobile robot path planning using support vector machines, in: in: ASME 2008 Dynamic Systems and Control Conference, Parts A and B, ASME, 2008, pp. 709–715.
[33] S. Tennety, S. Sarkar, E.L. Hall, M. Kumar, Support vector machines based mobile robot path planning in an unknown environment, in: in: ASME 2009 Dynamic Systems and Control Conference, vol. 1, ASME, 2009, pp. 395–401.
[34] V.N. Vapnik, V. Vapnik, Statistical Learning Theory, Wiley, New York, 1998.
[35] Q. Zhang, J. Sun, G. Xiao, E. Tsang, Evolutionary algorithms refining a heuristic: a hybrid method for shared-path protections in WDM networks under SRLG constraints, IEEE Trans. Syst. Man Cybern. B (Cybern.) 37 (2007) 51–61.

[37] Q. Zhu, Y. Yan, Z. Xing, Robot path planning based on artificial potential field approach with simulated annealing, in: in: Sixth International Conference on Intelligent Systems Design and Applications, IEEE, 2006, pp. 622–627.
[38] S.M. Lavalle, Motion Planning: Wild Frontiers.
[39] O. Souissi, R. Benatitallah, D. Duvivier, A. Artiba, N. Belanger, P. Feyzeau, Path planning: a 2013 survey, in: in: Proceedings of 2013 International Conference on Industrial Engineering and Systems Management (IESM), 2013, pp. 1–8.
[40] N. Morales, J.T. Toledo, L. Acosta, R. Arnay, Real-time adaptive obstacle detection based on an image database, Comput. Vis. Image Underst. 115 (9) (2011) 1273–1287, ISSN:1077-3142.
[41] M. Davoodi, F. Panahi, A. Mohades, S.N. Hashemi, Clear and smooth path planning, Appl. Soft Comput. 32 (2015) 568–579.
[42] P. Bhattacharya, M. Gavrilova, Roadmap-based path planning – using the voronoi diagram for a clearance-based shortest path, in: Robotics Automation Magazine, IEEE, 2008, pp. 58–66.
[43] S. Cossell, J. Guivant, Concurrent dynamic programming for grid-based problems and its application for real-time path planning, Robot. Auton. Syst. 62 (2014) 737–751.
[45] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, D. Johnston, S. Klumpp, D. Langer, A. Levandowski, J. Levinson, J. Marcil, D. Orenstein, J. Paefgen, I. Penny, A. Petrovskaya, M. Pflueger, G. Stanek, D. Stavens, A. Vogt, S. Thrun, Junior: the Stanford entry in the urban challenge, J. Field Robot. 25 (2008) 569–597.
[46] K.H. Su, F.L. Lian, C.Y. Yang, Navigation design with SVM path planning and fuzzy-based path tracking for wheeled agent, in: in: 2012 International Conference on Fuzzy Theory and it's Applications (iFUZZY), 2012, pp. 273–278.
[47] J. Wang, J. Wu, Y. Li, K. Li, The concept and modeling of driving safety field based on driver–vehicle–road interactions, in: in: 2014 IEEE 17th International Conference on Intelligent Transportation Systems (ITSC), 2014, pp. 974–981.
[48] Q.H. Do, S. Mita, H.T.N. Nejad, L. Han, Dynamic and safe path planning based on support vector machine among multi moving obstacles for autonomous vehicles, IEICE Trans. Inf. Syst. E96.D (2013) 314–328.
[49] T. Maekawa, T. Noda, S. Tamura, T. Ozaki, K. ichiro Machida, Curvature continuous path generation for autonomous vehicle using B-spline curves, Comput.-Aid. Des. 42 (2010) 350–359.
[50] M. Likhachev, D. Ferguson, Planning long dynamically feasible maneuvers for autonomous vehicles, Int. J. Robot. Res. 28 (2009) 933–945 http://ijr.sagepub.com/content/28/8/933.full.pdf+html.
[51] M. Deng, L. Jiang, A. Inoue, Mobile robot path planning by SVM and Lyapunov function compensation, Meas. Control 42 (2009) 234–237 http://mac.sagepub.com/content/42/8/234.full.pdf+html.
[52] M.A. Contreras-Cruz, V. Ayala-Ramirez, U.H. Hernandez-Belmonte, Mobile robot path planning using artificial bee colony and evolutionary programming, Appl. Soft Comput. 30 (2015) 319–328.
[53] N. Christiannini, J. Shawe-Taylor, Support Vector Machines and Other Kernel-based Learning Methods, 2000.