Dynamic and Safe Path Planning Based on Support Vector Machine among Multi Moving Obstacles for Autonomous Vehicles

Quoc Huy DO†, Nonmember, Seiichi MITA†, Member, Hossein Tehrani Nik NEJAD†, and Long HAN†, Nonmembers

SUMMARY We propose a practical local and global path-planning algorithm for an autonomous vehicle or a car-like robot in an unknown semi-structured (or unstructured) environment, where obstacles are detected online by the vehicle’s sensors. The algorithm utilizes a probabilistic method based on particle filters to estimate the dynamic obstacles’ locations, a support vector machine to provide the critical points and Bézier curves to smooth the generated path. The generated path safely travels through various static and moving obstacles and satisfies the vehicle’s movement constraints. The algorithm is implemented and verified on simulation software. Simulation results demonstrate the effectiveness of the proposed method in complicated scenarios that posit the existence of multi moving objects.

key words: path planning, support vector machine, particle filter, Bézier curve

1. Introduction

In recent years, autonomous vehicles have become an increasingly important research topic in various civilian operations. An important part of vehicle autonomy consists of a path planning and navigation system that enables it to safely maneuver through different type of environments. General path planning methods have been described in great detail by many authors, to such extent that they have become mainstream knowledge in the fields of computer science, artificial intelligence, and autonomous systems. As a result, many thorough reviews of these ideas have been published. These methods are also the simplest to describe and build upon, as more complicated path planning methods involving differential constraints (where paths involve time-dependent factors) or non-holonomic systems (where system behavior is path-dependent) can be derived from the simpler models [1],[2]. Early techniques were often applied to robotic systems and then extended for unmanned vehicles [1]–[8]. To generate a path which can avoid a moving obstacle, we need to stay away from the location occupied by the obstacles (e.g. other vehicles) at some future time. However, for most non-trivial cases, the future location of the obstacles can only be estimated and there will be some amount of uncertainty associated with that motion. For dynamic obstacle avoidance, the Rapidly Exploring Random Tree (RRT) technique is often used [9]. The MIT team for 2007 DARPA Urban Challenge used an algorithm based on the RRT approach to create a path planner [10]. The “Stanford Racing Team”, [11], used a mixture of approaches in their overall system. Their path planner uses the DARPA supplied data to formulate an initial plan consisting of trajectories and velocity requirements for each waypoint and trades off five objective definitions of effective paths, such as the proximity to the road center, clearance to obstacles, and vehicular motion requirements computed from a vehicle model. This method is effective for obstacle avoidance. This algorithm quickly discovers the environment and finds a path to the goal root in the random steps, although the path is not always guaranteed to be an optimal and safe one. The 2007 DARPA Challenge winner CMU team used a path planning method based on Anytime Dynamic A* (Anytime D*) algorithm [12]. In this case, they prepare two kinds of path set including a smooth trajectory and a sharp trajectory and choose an adequate path from among these paths. This approach generates good results in practice.

In terms of safety, path planning methods tend to maximize the distance between the vehicle and obstacles on the map. One typical approach is to use the Voronoi diagram [13],[14] or the potential field [15],[16]. However, these methods contain limitation and do not produce optimized path to the goal position. Moreover, the above mentioned methods cannot identify in a mathematically rigorous manner the most critical points within a given path.

To overcome the disadvantages of the above mentioned methods, we propose the usage of binary classification techniques based on the support vector machine (SVM). In addition to SVM [17], the particle filter method is applied to handle moving obstacles and the Bézier curves guarantee that the resulting path is a smooth route that has safe distances to the obstacles and satisfies the curvature constraints at the start and goal points.

This paper is organized as follows: In Sect. 2, the outline of the safest path planning and the advantages of the proposed method over the typical methods are described. Section 3 presents the vehicle model and path planning constraints. Section 4 describes the local path planning algorithm in detail. The global path planning algorithm is proposed in Sect. 5. The simulations and results will be demonstrated in Sect. 6. Finally, the conclusions and future work are given in Sect. 7.
2. Outline of the Proposed Method

Our objective is to find the safest path for the vehicle amid multiple obstacles. This path can be either globally or locally planned. The global path is usually run as a planning phase before the vehicle begins its journey to avoid static obstacles and prevent the vehicle from being stuck in a dead-end. The local path planning is a reactive process which relies on the latest sensor data to maintain vehicle safety and stability while moving from one global way point to next global way point. The higher control system will call the local planner frequently to deal with the current situation so that it has the curvature continuity and secure from the obstacles collision as described in Fig. 1.

In this section, we introduce the idea of applying the classification technique SVM for path planning and its advantages over typical related methods in literature.

2.1 Support Vector Machine for Path Planning

Our goal is to develop an algorithm for generating a path which is feasible and equidistant to the obstacles. This path resembles to the path shown in Fig. 2. Here, \( d_{\text{safe}} \) is a defined safe margin, \( d \) is the distance from the closest obstacle to the straight line path, \( d_1 \) and \( d_2 \) are the distance from two closest obstacles to the safest path. This algorithm is applied in the situation where a straight path does not guarantee to be safe (\( d < d_{\text{safe}} \)) to provide the safest path with equidistant to the obstacles.

In order to achieve the path that maximize the distance to the obstacles, some methods use the Voronoi diagram or potential field technique but we apply an SVM in path planning for autonomous vehicle. The benefits of using SVM compare to other approaches are:

1. It doesn’t require fully information about the environment.
2. No local minima problem.
3. The complexity of the algorithm is not affected by the shapes of the obstacles.
4. It can provide the most critical points within a path mathematically clearly.

The SVM can classify both linear and nonlinear data. The nonlinear SVM has the following advantages which enable us to generate the safest path:

- It can classify all the data points in the map.
- It can provide the hyperplane with maximum distance to the obstacles
- It provides a margin between two categories of data.

To apply the SVM, first we divide the obstacles into two separate groups. After separating the obstacles, we input the points on the obstacle boundaries as the training data for the nonlinear SVM method. The results of the training process are the hyperplane (the classify boundary) and the closest points to the hyperplane which are called support vectors. Figure 3 shows an example of learning step.

According to the properties of the hyperplane, the projected hyperplane in the original 2D space has the maximum distance to the two categories of obstacles and most critical points along the path are picked up as support vectors.

In our method, there are different ways of dividing obstacles into two groups and each of them corresponding to one separating boundary and one margin as shown in the Fig. 4 with \( w_1, w_2, w_3 \) represent the separating boundary parameters achieved after learning. We compare each margin to find the separating boundary with maximum margin. If the margin is not sufficient for the vehicle then the algorithm will stop. By doing the comparing step, we guarantee that the separating boundary with maximum margin is achieved and another important point is that for the narrowest area along the path, the critical points are picked up.
2.2 Advantage over Typical Related Methods

One of the most benefits of our proposal in comparison to other methods is that it can provide the most critical points mathematically clearly. It also completely solve fatal problems of typical methods as follow:

2.2.1 Potential Field Method

In the potential field method [15], the robot is driven by the attractive force of the goal and the repulsive force of the obstacles. The drawback of this method is that there is a tendency that the robot might get trapped in local minima which is not the goal position. In symmetric environments such as in Fig. 5(a), the total repulsive force of the two obstacles is symmetric to the attractive force so that the combination force is zero and robot stops. But in these kinds of environments, the proposed SVM based method can easily generate a safe path by mean of the hyperplane as shown in Fig. 5(b).

2.2.2 Voronoi Diagram Based Method

The comparisons between Voronoi based method [13] and the proposed method are described in Fig. 6. For the cases that objects have complex non-convex forms such as in Fig. 6, the Generalized Voronoi Diagram can provide the vertices and the edges but the road follow the vertices will go inside the obstacle zone, which might be unsafe for the vehicle driving. The SVM method can prevent this problem because it can provide the support vectors (critical points) on the objects and the hyperplane which can be useful for determining the path’s width and the safe margin so that the result path can lead the vehicle not go inside the obstacle zone.

3. The Vehicle Model and Path Planning Constraints

Our problem is bounded in a 2D world, considering the movement of a ground vehicle with a mission defined by a start point, goal point, heading angle, the curvature at the start and goal points and a set of obstacles. The vehicle model is described as in Fig. 7. \( O_G, O_L \) are the global and local vehicle’s origin of coordination. \( O_L \) is the center point at the front of the vehicle (the position of the laser scanner). For the dynamics of the vehicle, the state and the control vectors are denoted as \( s(t) = (x(t), y(t), \theta(t), \phi(t)) \) and \( u(t) = (v(t), w(t)) \) respectively, where \( (x(t), y(t)) \) represents the position of the center front point of vehicle at time \( t \) in the global coordination. \( \theta \) measures the heading angle (the orientation of the car body) with respect to the Y axis, \( \phi(t) \) is the steering angle, \( v(t) \) and \( w(t) \) are the longitudinal and rotational velocity of the vehicle at time \( t \). \( l \) is the distance between the front and rear wheel axis.

\[
\begin{bmatrix}
\dot{x} \\
\dot{y} \\
\dot{\theta} \\
\dot{\phi}
\end{bmatrix} = \begin{bmatrix}
\cos \theta \\
\sin \theta \\
\tan \phi / l \\
0
\end{bmatrix} v + \begin{bmatrix}
0 \\
0 \\
0 \\
1
\end{bmatrix} w
\]
4. Local Path Planning Algorithm

The algorithm includes different stages as presented in Fig. 8.

4.1 Initial Stage

The path planner generates a local map, including the following information from higher level route planner:

- Start state: Position \((x_s, y_s)\), heading direction \(\phi_s\), curvature \(\kappa_s\), and velocity \(V_s\) at the start point.
- Goal state: Position \((x_g, y_g)\), heading direction, \(\phi_g\), curvature \(\kappa_g\), and velocity \(V_g\) at the goal point.

In the first step, we update the local map using the latest information from the laser scanner. Radial bounded nearest neighbor graph (RBNN) [18] is chosen to cluster the laser scanner data points into disjoint clusters. In this graph every node is connected to all neighbors that lie within a predefined radius. Given data points in two dimensional space, in the scan pattern at time \(t\), \(D_t = (p_1, p_2, ..., p_n)\), \(p_i = (x_i, y_i)\), we use a Euclidean distance metric \(d(p_i, p_j) = \|p_i - p_j\|_2\) to calculate the distance between points \(p_i, p_j\), and put points into one cluster where \(d(p_i, p_j) < r\). The radius is determined according to laser point's density in the local map and the vehicle size. Figure 9 shows an example of a basic data point sets with five clusters segmented by RBNN. Each of these clusters corresponds to an obstacle.

The computation complexity of this method in the worst case is \(O(n \log(n))\) where \(n\) is the number of data points. The latest laser scanner data provide current obstacles’ locations. The previous obstacles’ locations were stored to calculate the velocity of the obstacles and to predict the next possible postures.

In step 2, as the vehicle moving, its position and heading angle changing, it is necessary to convert the obstacles'
current and previous positions into the same coordination system (the local map).

\[
\begin{align*}
    \begin{bmatrix}
        x_{\text{Lnew}} \\
        y_{\text{Lnew}}
    \end{bmatrix} &= \begin{bmatrix}
        \cos \alpha & \sin \alpha \\
        -\sin \alpha & \cos \alpha
    \end{bmatrix} \begin{bmatrix}
        x_{\text{Lold}} \\
        y_{\text{Lold}}
    \end{bmatrix} \\
    &\quad - \begin{bmatrix}
        \cos \theta & \sin \theta \\
        -\sin \theta & \cos \theta
    \end{bmatrix} \begin{bmatrix}
        X_{\text{new}} - X_{\text{old}} \\
        Y_{\text{new}} - Y_{\text{old}}
    \end{bmatrix}
\end{align*}
\]

(5)

where \((x_{\text{Lold}}, y_{\text{Lold}})\) is the position of the obstacle in the previous local coordination and \((x_{\text{Lnew}}, y_{\text{Lnew}})\) is the translated position in the current coordination; \((X_{\text{new}}, Y_{\text{new}})\) and \((X_{\text{old}}, Y_{\text{old}})\) are the current and previous positions of the vehicle in the global coordination; \(\alpha = \theta - \theta'\) is the angle between the previous X-axis and the current X-axis as illustrated in Fig. 10.

In our path planner, for simplicity, we treat the vehicle as a point, which may cause the collision because of the vehicle’s size. Hence, we increased the size of the obstacles to account for the vehicle sizes and ensure to avoid the obstacles. If the distance between two obstacles is smaller than the safe margin, the two obstacles will be merged together and be considered as one obstacle. Figure 11 shows a path generated, considering the existence of obstacles’ boundaries having the safe margin added.

4.2 Prediction Stage

In step 3, we apply particle filter to estimate the positions of the moving obstacles. The key idea of the particle filter is to present the posterior by a set of weighted particles. Based on obstacle’s information such as position and velocity, the next possible positions will be calculated and sampled. For each obstacle, \(M\) number of poses will be generated by the sampling algorithm presented in Fig. 12 [19].

\[
\begin{align*}
    v' &= v + \text{sample}(\alpha_1 | v | + \alpha_2 | w |) \\
    w' &= w + \text{sample}(\alpha_3 | v | + \alpha_4 | w |) \\
    \gamma &= \text{sample}(\alpha_5 | v | + \alpha_6 | w |) \\
    x' &= x - \frac{v'}{w'} \sin \theta + \frac{v'}{w'} \sin(\theta + w' \Delta t) \\
    y' &= y + \frac{v'}{w'} \cos \theta + \frac{v'}{w'} \cos(\theta + w' \Delta t)
\end{align*}
\]

Return \(s_t = (x', y', \theta')\)

Fig. 12 Algorithm for sampling pose \(s_t\) from pose \(s_{t-1}\) and control \(u_t\).

Function \text{sample}(z)\) generates a random sample from a zero centered distribution with standard deviation \(z\).

\[
sample(z) = \frac{1}{2} \sum_{i=1}^{12} \text{rand}(-z, z)
\]

where \(s_{t-1}\) is a state represented by \((x, y, \theta)\) which is the center point of the obstacle at time \(t-1\); \(s_t\) is represented by \((x', y', \theta')\) and \(u_t\) is represented by velocity \((v, w)\), \(\theta\) is the heading direction of the obstacle. The \(\Delta t\) value for each obstacle will be calculated considering the obstacle’s velocity and relative distances to the vehicle.

The sampled points will create a possibility area. This area will be truncated by a probability value \(\beta (0 < \beta < 1)\). The boundaries of this possible area after added the initial
Fig. 13  Obstacle position prediction.

Fig. 14  Ways to divide obstacles into two groups.

Fig. 15  Virtual obstacles outside of the real group.

Fig. 16  Virtual obstacles.

size of the obstacle are used to define the new obstacles’ boundaries. Figure 13 illustrates an obstacle with its predicted position region at the time $t$.

4.3 Classification Stage

In our method, the obstacles will be divided into two separate groups from left to right. Since we only need the paths which pass by the start point, the obstacles can be divided as in Fig. 14. By this way, for a given set of $n$ obstacles the maximum number of hyperplane to be considered is $n - 1$.

For the completeness of the method, two more virtual obstacles are added on the left and right side of the real obstacle’s group as shown in Fig. 15.

By applying the virtual obstacles, this path planning method is also applicable for the case there is only one obstacle. The virtual obstacles’ positions are defined such that the goal point is in the middle of the virtual obstacle and the closest real obstacle. The distance between the virtual obstacle and the closest real obstacle must be larger or equal to the car-size as illustrated in Fig. 15 and Fig. 16. The newly added obstacles are not involved in collision checking step because they are virtual.

After separating the obstacles, we select the points on the obstacles’ boundaries as the training points for the nonlinear SVM method. After solving the dual problem (6) for the SVM with $K$ is the RBF kernel function as described in [21], we achieve the hyperplane and the support vectors.

$$L(\lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{N} \gamma_i \gamma_j \lambda_i \lambda_j K(q_i, q_j)$$

where $N$ is the number of support vectors, $\gamma_i$ are classified label values, $\lambda_i$ are the Lagrange multipliers. In original input space the margin between the hyperplane and the support vectors is as follows.

$$\text{margin width} = \sum_{i,j=1}^{N} \gamma_i \gamma_j \lambda_i \lambda_j (q_i \cdot q_j)$$

To learn the non-linear data, the SVM uses function $\varphi$ which takes input $q \in Q$ (input space) and maps it onto (“feature space”). The margin’s width in feature space is

$$\sum_{i,j=1}^{N} \gamma_i \gamma_j \lambda_i \lambda_j \varphi(q_i)^T \varphi(q_j)$$

The kernel trick enables us to work in high-dimensional feature spaces without actually having to perform explicit computations in this space. Kernel function $K(q_i, q_j)$ takes two inputs and gives their similarity in space.

$$K: Q \times Q \rightarrow R, \quad K(q_i, q_j) = \varphi(q_i)^T \varphi(q_j)$$

We only use the RBF kernel because the RBF kernel is able to classify non linear data and it requires fewer parameter than polynomial and sigmoid kernel. The kernel RBF has the form:

$$K(q_i, q_j) = \exp(-\delta ||q_i - q_j||^2)$$

where $q_i, q_j$ are data in original space, $\delta > 0$ is a given training factor. $\delta$ is an important factor that affects the accuracy of the method so that it is needed to choose carefully. Here, we define $\delta$ by applying the cross validation process:

- Step 1: Prepare pre-classified (correct) data, for
will be chosen as joint-points closest intermediate points to the start point and goal point (the original 2D “input space”).

- Step 2: For each separated data set, we randomly split them into two sets, one set is for learning and one for verifying the classified result as illustrated in Fig. 17.

- Step 3: We train the learning sets with an initial $\delta$ value and then use the trained results to predict the classify value of the verifying data sets and compare with their labeled value to determine the accuracy.

The step 3 is repeated with different $\delta$ value for each iteration until we get a desired accuracy.

This cross validation process is very time consuming so that it needed to be done offline and the result $\delta$ will be used online for training and classification.

The kernel function helped learning nonlinear pattern by using a linear model. Here, the features of the data that we used are the $x$ and $y$ coordination so that the norm used in RBF kernel is the 2 dimension Euclidean distance. The Appendix will prove that in our case, the maximizing-margin simplification case in S. Akaho’s method [23]. However, in S. Akaho’s method the input points are limited and the projected points may overlap each other so that it is not sufficient to create a continuous boundary. Therefore, in our method, we achieve the boundary by the following procedure: By applying the RBF kernel for training data, the hyperplane equation has the following form:

$$f(x, y) = \sum_{i=1}^{N} y_i A_i e^{-\delta^2[(x-a_i)^2 + (y-b_i)^2]} + h = 0 \quad (10)$$

where $(a_i, b_i)$ are the support vectors $x$ and $y$-coordination values, $h$ is the hyperplane’s bias. All of these parameters can be calculated from the learning process.

The $y$-coordinate of the estimating point will be increased from $I_1$ to $I_2$. For each $y$-coordinate value we calculate the corresponding $x$-coordinate value by using the Newton-Raphson Iteration:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \quad (11)$$

where $x_0$ is the $x$-coordinate of the previously estimated point. The loop stops when $|x_{i+1} - x_i| \leq \varepsilon$. $\varepsilon$ is a user defined value. This step is described in Fig. 17.

For the special case $\alpha = 0$ of the cost function (4), the projected hyperplane is the safest path among the obstacles. However, the safest path also has to satisfy the curvature and other constraints. The distance to the hyperplane will be one element to evaluate the path.

### 4.4 Smoothing Stage

In step 5, we utilize 3 continuous Bézier curves (12) to make the path feasible for the vehicle. Bézier curves have shown their applicability and robustness as discussed in our previous work [20].

Given control points $B_0, B_1, \ldots, B_n$, the Bézier curve $P(t)$ of degree $n$ can be generalized as follows:

$$P(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i B_i \quad (12)$$

where $t \in [0, 1]$, $P(0) = B_0$ and $P(1) = B_n$.

The 4th degree Bézier curve with five control points $B_i^j$ are used ($i \in [0, 2]$ is the segment index, $j \in [0, 4]$ is the control point index; $X_i^j, Y_i^j$ is the corresponding X and Y coordinate) because it can satisfy our curvature continuous constraints. The trajectory will be a combination of three curve segments.

For the first segment, we generate the path from the start point (current vehicle position) to the first connection point $I_1$, considering the curvature continuity at the start point.

The 3 y-coordinates are select by the following equations.

$$Y_1 = \frac{3Y_e + Y_g}{4}$$
From Eq. (14) the curvature at the beginning point can be calculated as follows:

\[
\kappa(0) = \kappa_s = \frac{3(X_s(Y_0^1 - Y_2^0) + X_0^1(Y_s - Y_0) + X_1^2(-Y_1 + Y_s))}{4((X_1^0 - X_0)^2 + (Y_0^0 - Y_s)^2)^{3/2}} \\
\]

(15)

\[
X_2 = X_s - C_s \frac{(Y_s - Y_s)^2}{12} \quad X_3 = \text{arg min}\{C_s | C_s: \cos tfunction} \]

(16)

The control point for the second segment is selected on the hyperplane with \( B_1^2 = I_1 \), \( B_2^2 = I_2 \) other control points have equidistance in y-coordinate as shown in Fig. 19.

Fig. 19 First and second segment control points.

The X-coordination values are calculated as follow:

\[
X_0 = I_1 \quad X_4 = I_2 \]

(17)

In order to have continuous curvature at \( I_1 \), \( X_1 \) and \( X_2 \) are calculated as in (18):

\[
X_1^1 = \frac{(Y_1^1 - Y_0^1)(X_0^0 - X_1^0)}{Y_0^0 - Y_1^0} + X_1^0 \]

\[
X_2^2 = \frac{4\kappa_{I_1}[(X_1^0 - X_1^1)^2 + (Y_1^0 - Y_1^1)^2]^{3/2}}{3(Y_0^0 - Y_1^0)} + 2X_1^2 - X_0^0
\]

(18)

where \( \kappa_{I_1} \) is the curvature at point \( I_1 \) and can be calculated by the following formula:

\[
\kappa_{I_1} = \frac{3(X_2^0 - 2X_0^0 + X_0^0)(Y_0^0 - Y_1^0)}{4[(X_0^0 - X_1^0)^2 + (Y_0^0 - Y_1^0)^2]^{3/2}}
\]

(19)

\[
X_3 = \text{arg min}\{C_s | C_s: \cos tfunction} \]

(20)

The last segment’s control points are chosen as follow: The control points are in an even pattern on the y-axis.

\[
B_1^3 = I_3 \quad B_2^3 = \text{Goal point}
\]

(21)

The two control points \( B_1^3 \), \( B_2^3 \) are calculated by the same method as the second segment. The control point \( B_3^3 \) is calculated to satisfy the constraint of the heading angle at the corresponding goal point.

\[
X_2^3 = \frac{(Y_2^1 - Y_0^0)(X_1^2 - X_2^0)}{Y_1^2 - Y_2^0} + X_2^0
\]

(22)

\[
X_3 = X_4 - (Y_4 - Y_1^0)\tan \phi \quad X_2 = \text{arg min}\{C_s | C_s: \cos tfunction} \]

The proposed SVM based method is applicable not only for
local path planning but also for global path planning. Global path planning is a planning method in which all of the carlike robot’s acquired knowledge to reach a goal is included, not just the currently sensed world. It is usually used to make plans for long distances and time periods, allows robot to avoid getting trapped and to reach goal in most efficient manner. In this part, we consider applying the path planning in complicated environment such as the indoor or cluttered environment. The path planning steps are described in Fig. 21.

- Step 1: For global path planning, we first explore the map by using the Rapidly-exploring Random Tree technique [9] to quickly discover the map and find a guiding path to the goal. The basic RRT algorithm is as follows. Starting with a tree T with a root node, the algorithm iteratively adds more nodes to the tree. During each iteration, a configuration qr is randomly generated, and the basic RRT algorithm attempts to connect the nearest node qn in the tree T to qr by a straight line in the configuration space. If the configuration qr and qn can be connected via a collision-free path, the tree is extended from qr to qn and grows. The iteration stops when the goal is reached. By using the RRT we ensure to avoid the local minima phenomenon which may happen in a complicated map. In this step no more constrain except collision avoidance was added so that the RRT can quickly return the resultant path.

- Step 2: The advantage of using the guiding path is that we do not need to collect all the data points on the map to process, in comparison with the Voronoi diagram based method which need to collect all the data to generate the “skeleton map” for later applying path searching algorithm [13], [14], [21]. This path also segments the data into two parts: the left side and the right side. After generating the guiding path, we pick up all the points on the left and right borders of this path to input as training points for SVM method and train them. The training process and hyperplane achieving is performed as described in Sect. 4.

- Step 3: Since the global map does not have a specific range or limitation, the B-spline curve is then utilized to make the final smooth path.

Given m real values gi, called knots, with $g_0 \leq g_1 \leq \cdots \leq g_m$ a B-spline of degree n is a parametric curve composed of a linear combination of basis B-splines $b_{i,n}$ of degree n.

$$S(g) = \sum_{i=0}^{m-n-2} P_i b_{i,n}(g), \quad g \in [g_n, g_{m-n-1}]$$  \hspace{1cm} (23)

The points $P_i \in \mathbb{R}^d$ are called control points or de Boor points. There are $m-n-1$ control points, and they form a convex hull. The $m-n-1$ basis B-splines of degree n can be defined, for $n = 0, 1, \ldots, m-2$, using the Cox-de Boor recursion formula:

$$b_{j,0}(g) = \begin{cases} 1 & \text{if } g_j \leq g \leq g_{j+1}, \quad j = 0, \ldots, m-2 \\ 0 & \text{otherwise} \end{cases}$$  \hspace{1cm} (24)

One of the advantages of B-spline is that we can modify a curve locally without changing the shape in global. In our method, the curve with degree $n = 3$ was used. The control points of the path are placed along the hyperplane. The first and last control points are the start and goal point properly.

An example of global path planning is shown in Fig. 22. The use of the SVM together with the RRT method improves the quality of the calculated path considerably. On the one hand, the path tends to go close to the curve maximizes distance to the road’s border because of the optimal conditions of this area for the carlike-robot movement; on the other hand the trajectories are also considerably smoothed.

6. Experimental Results

6.1 Local Path Planning Simulation

We implemented our algorithm in C and incorporated it into the software platform PreScan 4.0 developed by TNO Science and Industry [22]. The simulated experiments have been done on a 3.0 GHz Dell workstation running WindowXP 64 bit and compiled with MATLAB Simulink.

In the simulation, the vehicle has a virtual laser scanner at-scanning angle. The simulated vehicle has the constant velocity $v = 10 \text{ m/s}$. The obstacles are static blocks or other simulation vehicles and can have different velocities. Every 25 ms, the path planer will be called to generate a new path. In these experiments, we use the $\delta = 10^{-4}$ ( $\delta$ is determined by cross validation to provide the 100% accuracy) for RBF kernel. For obstacles prediction, the number of sample value M is selected to accurately represent the possible position. If M is too small the prediction is not accurate. On the other hand, if M is too large it will increase the computation time.
Through different tests, we found \( M = 500 \) provide sufficient accuracy for our experiments; the longitudinal and rotational error parameters \( \alpha_{1,4} = 0.1 \) are vehicle-specific error parameters. In cost function (3), the value \( \alpha \) is depend on the velocity of the vehicles. In our experiments with the vehicle velocity \( v = 10 \text{ m/s} \), \( \alpha = 0.9 \) provide the most reasonable results. The number of obstructing particles at a given time is depending on the range, the number of laser rays and the corresponding positions of the obstacles to the vehicle.

We generated different maps and scenarios to verify and evaluate the algorithm. Figure 22 shows an example of our tested maps. In all simulated experiments, the vehicle can only see the obstacles in front of it, the environment map will be updated while the vehicle travelling. Based on what the vehicle can “see” at the moment, a new route dynamically is created. Figure 23 (a) shows the positions of the vehicle during the created course along its planned path. Figure 23 (b) presents the paths corresponding to the situation, generated by proposed method, the \( A^* \) and RRT method.

Figure 24 shows the cost value comparison in term of smoothness and safety of the three paths. The safety is calculated based on the average distance from the path points to the hyperplane.

Figure 25 illustrates the motion plan of the vehicle to avoid moving obstacles. In this experiment the vehicle and
the moving obstacles have different velocities. The algorithm is applied only for the main vehicle, the other car (obstacles) come straight forward from randomly various directions. The obstacles’ velocities are randomly generated from 5 m/s to 10 m/s in the simulation system. The red points on the moving obstacles are the critical points picked up by the SVM.

To evaluate the performance of the method, we compare its runtime with the generalized Voronoi diagram method. In this comparison, the number of obstacles are randomly generated, the positions and theirs sizes are also generated randomly on the grid map.

Figure 26 shows the results of the comparison of the proposed method with the Generalized Local Voronoi Diagram by Mahkovic. Both algorithms are implemented on a NEC 2.8 Ghz computer running Vine Linux. The reported runtime is the total CPU time in millisecond for all the steps needed to generate the path from the input data. The runtimes were averaged after executing each algorithm 10000 times. The x-axis represents the number of inputted data points (the points on the boundaries of the obstacles). The runtime for each algorithm is highly dependent on the number of data points.

6.2 Global Path Planning Algorithm Simulation

Figure 27 illustrates our testing environment. The dimensions of the laboratory testing environment are 72 × 50 meters (the cell resolution used in the a slam-map are 5 cm). The slowest computational time detected was 182 ms (on a 2.8 Ghz processor NEC computer running Vine Linux). Figure 24 (c) shows an example of global path planning in the SLAM map, the pink points are the border of the map.
Table 1  Test map 1 runtime comparison.

<table>
<thead>
<tr>
<th>Distance between start and goal (cell)</th>
<th>Average computation time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Voronoi diagram and fast marching</td>
</tr>
<tr>
<td>Map 's diagonal length</td>
<td>192</td>
</tr>
<tr>
<td>0.75* Map 's diagonal length</td>
<td>156</td>
</tr>
<tr>
<td>0.5* Map 's diagonal length</td>
<td>115</td>
</tr>
<tr>
<td>0.25* Map 's diagonal length</td>
<td>97</td>
</tr>
</tbody>
</table>

Table 2  Test map 2 run-time.

<table>
<thead>
<tr>
<th>Distance between start and goal (cell)</th>
<th>Average computation time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Voronoi diagram and fast marching</td>
</tr>
<tr>
<td>Map 's diagonal length</td>
<td>147</td>
</tr>
<tr>
<td>0.75* Map 's diagonal length</td>
<td>110</td>
</tr>
<tr>
<td>0.5* Map 's diagonal length</td>
<td>89</td>
</tr>
<tr>
<td>0.25* Map 's diagonal length</td>
<td>58</td>
</tr>
</tbody>
</table>

Fig. 31  Path’s curvature in test map 2.

Fig. 30  Test map 2.

Table 2  Test map 2 run-time.

<table>
<thead>
<tr>
<th>Distance between start and goal (cell)</th>
<th>Average computation time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Voronoi diagram and fast marching</td>
</tr>
<tr>
<td>Map 's diagonal length</td>
<td>147</td>
</tr>
<tr>
<td>0.75* Map 's diagonal length</td>
<td>110</td>
</tr>
<tr>
<td>0.5* Map 's diagonal length</td>
<td>89</td>
</tr>
<tr>
<td>0.25* Map 's diagonal length</td>
<td>58</td>
</tr>
</tbody>
</table>

after adding the vehicle’s width; the blue and orange points are points on the two borders of the path (the training data points) while the green and red points are the critical points of the path (the support vectors).

To demonstrate the performance of our method, we compared our method’s results with the results of the latest method Voronoi Diagram and Fast Marching [22] in different test maps with consideration of the vehicle’s limited steering angle constraints. The test map 1 in Fig. 27 has 966 x 166 cells. The total number of data on the map to process is 48268 data points. Figures 28 and 29 show generated paths of the two methods. Table 1 and Table 2 show the average run-time of the two methods properly. Each method was run 10000 times on each test map.

Test map 2 has the size of 654 x 496 cells. The total number of data points is 33208 data points.

Figures 30 and 31 show that while maintain the maximize distance to the obstacles property, the proposed method provides a smoother path due to the applying of controllable curves. The smaller distance from start and goal point is, the faster our method is because the amount of data for processing is reduced while in the Voronoi diagram based method the amount of data unchanged. By using virtual obstacles, the proposed method provides a path smaller curvature changes which makes the path more reasonable than the extended Voronoi and fast marching method.

7. Conclusion

This paper presents a new path planning algorithm based on particle filters to predict the obstacle position, an SVM to identify the critical points in a mathematically rigorous manner and to obtain the hyperplane and Bézier curves for autonomous vehicles with curvature and safe corridor constraints. The global path planning which based on SVM is also described. In summary, this method is more effective than the conventional alternatives: the proposed SVM based method can be applied to make the local and global path planner; the path calculated has good safety and smoothness characteristics. To handle moving obstacles, we proposed a probabilistic approach that uses particle filter. The vehicle’s dynamic attributes are always considered for generating the local path. The method was implemented and validated for vehicle application on the PreScan simulation software. Our simulation results show that the path planner successfully solves the problems with static obstacles as well as moving obstacles and its run-time is less than 25 ms which is suitable for real-time application. The method can operate directly over a 2D image map which is validated by our real environment SLAM map. The SVM can work with data in n-dimension space so that we can extend the current method so that it can be applied in higher spaces: 3D space, 2D space added time space or even 3D space added time space.

References

the original classification boundary in the feature space.

In [23], S. Akaho proposed an algorithm to maximize the margin in the input space for general case with general kernel but the method is time consuming and not suitable for real time application. In our paper, we apply the SVM for the specific input space (the robot 2 dimension space) to find a path that goes safely through the obstacles. In our methods, the obstacles are divided exactly into two groups left and right hand sides based on their position. Therefore, the input samples for training in SVM method are always correctly separated in two groups as shown in Fig. A-1.

The case where the input samples overlap each other (as illustrated in Fig. A-2) will not happen.

Besides, since our objective is to find the path goes safely through obstacles, we try to find the maximization margin in a limited area between the two classes of obstacles, not in the whole input space. So, the projection of the classification boundary is always unique in this limited area.

Given a set of input data $\mathbf{D} \in \mathbb{R}^2$; a set of support vectors $\mathbf{S} \in \mathbb{R}^2$ achieved from the original SVM, let $s \in S$ be an arbitrary support vector, $s$ is represented by $\varphi(s)$ in the feature space $H$; $\varphi(s^*) \in H$ is the closest point to $\varphi(s)$ on the hyperplane. $s' \in \mathbb{R}^2$ is the projected point of $\varphi(s^*)$ in the input space. $s$ and $s'$ are as shown in Fig. A-2.

**Proposition 1**: $s'$ is the closest point to $s$ on the projected hyperplane in the input space. ($\forall s' \in D, ||s-s'|| < ||s-s'||$)

-Proof:

Let $\varphi(s') \in H$ be an arbitrary points on the hyperplane in the feature space and $s' \in \mathbb{R}^2$ is the projected point of $\varphi(s')$ in the input space.

The distance between $\varphi(s)$ and $\varphi(s')$ in feature space is

$$d(s, s') = ||\varphi(s) - \varphi(s')||$$

$$d(s, s')^2 = ||\varphi(s) - \varphi(s')||^2 = (\varphi(s) - \varphi(s'))^T(\varphi(s) - \varphi(s'))$$

$$d(s, s')^2 = K(s, s) + K(s', s') - 2K(s, s')$$

where $K$ is the kernel function and $K(s, s') = \varphi(s)\varphi(s')$ is
the definition of the kernel function.

In our method we use the RBF kernel function

$$K(s, s) = \exp(-\delta||s - s||^2)$$

where \(\delta > 0\) so that

$$d(s, s')^2 = 1 + 1 + 2 \exp(-\delta||s - s'||^2)$$

Similarly, we obtain

$$d(s, s')^2 = 1 + 1 + 2 \exp(-\delta||s - s'||^2)$$

because \(s\) is the support vector so that in the feature space,

$$d(s, s')^2 < d(s, s')^2$$

\(\Rightarrow \ ||s - s'||^2 < ||s - s'||^2\)

That mean the projected point \(s'\) of \(\varphi(s')\) in the input space is also the closest point from support vector \(s\) to the projected classification boundary.

**Proposition 2:** The distance from the support vector to the projected classification boundary is the minimum distance from the sample to the hyperplane in the input space.

**Proof:** Let \(s \in S\) be an arbitrary support vector, \(s' \in R^2\) is the closest point of \(s\) on the projected hyperplane (classify boundary), \(x \in D\) is an arbitrary sample point near \(s\) and \(x' \in R^2\) is the closest point to \(x\) on the projected hyperplane as shown in Fig. A.3. The distance from \(s\) to \(s'\) and \(x\) to \(x'\) represents the distance from \(s\) and \(x\) to the hyperplane. In the feature space, \(s, s', x, x'\) are represented by \(\varphi(s), \varphi(s'), \varphi(x), \varphi(x') \in \mathcal{H}\).

Similar to the proof of proposition 1, the distance between \(s\) and \(s'\) in feature space is

$$d(s, s') = ||\varphi(s) - \varphi(s')||$$

and

$$d(x, x') = 1 + 1 + 2 \exp(-\delta||x - x'||^2)$$

because \(s\) is the support vector so that in the feature space,

$$d(s, s')^2 < d(x, x')^2$$

\(\Rightarrow \ ||s - s'||^2 < ||x - x'||^2\)

Here, the features of the vector that we used are the \(x\) and \(y\) coordinates so that the norm used in RBF kernel is also the 2 dimension Euclidean distance. Thus if sample \(x\) is not a support vector in the feature space then the distance from it to the hyperplane in the input space will larger than the distance from the support vector to the hyperplane. This means in this case, the critical points in the feature space reserve their properties in the input space, the projected curve of the classification boundary in the input space maintains the maximizing property.

**Proposition 3:** The distance between the two points mapped by RBF kernel in the feature space is monotonic with the distance between their corresponding points in the two dimensions Euclidean input space.

**Proof:** Given a set of points in input space \(s_0, s_1, s_2 \in R^2\); \(s_0, s_1, s_2\) are represented correspondingly by \(\varphi(s_0), \varphi(s_1), \varphi(s_2)\) in the feature space \(\mathcal{H}\). Let \(d_1, d_2\) are distances between the points: \(d_1 = ||s_1 - s_0||\), \(d_2 = ||s_2 - s_0||\). The distances between the mapped points in the feature space are: \(d_1' = ||\varphi(s_1) - \varphi(s_0)||\), \(d_2' = ||\varphi(s_2) - \varphi(s_0)||\). Assume that \(d_1' < d_2'\) we need to prove \(d_1 < d_2\).

We have

$$d_1'^2 = ||\varphi(s_1) - \varphi(s_0)||^2$$

\(\iff d_1'^2 = (\varphi(s_1) - \varphi(s_0))^T(\varphi(s_1) - \varphi(s_0))\)

Similar to proposition 1, we get

$$d_1'^2 = 2 - 2 \exp(-\delta||s_1 - s_0||^2)$$

$$\iff d_1'^2 = 2 - 2 \exp(-\delta d_1^2)$$

Similarly, we get \(d_2'^2 = 2 - 2 \exp(-\delta||s_2 - s_0||^2)\)

$$\iff d_2'^2 = 2 - 2 \exp(-\delta d_2^2)$$

Since \(d_1'^2 < d_2'^2\) as the assumption, we get \(d_1 < d_2\).

That means the relation \(d_1'^2 < d_1'^2\) in the feature space is preserved in the input space and vice versa. So that the maximum margin property found in the feature space is preserved in the input space.

![Support vectors and hyperplane in input space.](image-url)

**Quoc Huy Do** received the B.S. and M.S. degrees in Information Technology from Hanoi University of Technology at Vietnam in 2006 and 2008, respectively. He served as the faculty of Hanoi University of Science and Technology for the academic years of 2006–2009. He is currently a doctoral student at Toyota Technological Institute. His research field is path planning for autonomous vehicle.
Seiichi Mita received the B.S. degree, the M.S. degree and the Ph.D. degree in electrical engineering from Kyoto University in 1969, 1971, 1989 respectively. He studied at Hitachi Central Research Laboratory, Kokubunji, Japan from 1971 to 1991 on signal processing and coding methods. Now, he is a professor of Toyota Technological Institute (TTI) in Nagoya since 1999 and a director of Research Center for Smart Vehicles at TTI. Currently, he is greatly interested in the research area of autonomous vehicles and sensing systems. Dr. Mita is a member of the Institute of Image Information and Television Engineers in Japan. He is also a member of IEEE. He received the best paper award of IEEE Consumer Electronics Society in 1986 and the best paper and author awards of the Institute of Television Engineers in Japan in 1987 and 1992 respectively.

Hossein Tehrani Nik Nejad received the B.S. and M.S. degree in industrial engineering from Sharif University of Technology at Iran in 1996, 1998 respectively. He got his Ph.D degree in mechanical engineering from Osaka Prefecture University at Japan in 2009. Since 2009, he joined the Toyota Technological Institute at Japan as a Post Doctorate Fellowship at Smart Vehicle Center. Dr. Tehrani is a IEEE member and also member of RSJ (Robotic Society of Japan).

Long Han received his Ph.D. degree from the University of Science and Technology of China in 2007. Then he spent two years in Shanghai Fire Research Institute and TCL Communication. After that he joined the Toyota Technological Institute and worked on the smart vehicles. His research interests dealt with architecture for autonomous systems, actions planning and plans execution control.