2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

Smooth Global and Local Path Planning for Mobile Robot Using Particle Swarm Optimization, Radial Basis Functions, Splines and Bézier curves.

Nancy Arana-Daniel, Alberto A. Gallegos,
Carlos López-Franco and Alma Y. Alanis
Centre of Exact Sciences and Engineering (CUCEI)
University of Guadalajara (UDG)
Guadalajara, Jalisco, México
Email: nancyaranad@gmail.com

***Abstract*—An approach to plan smooth paths for mobile robots using a Radial Basis Function (RBF) neural network trained with Particle Swarm Optimization (PSO) was presented in [1]. Taking the previous approach as an starting point, in this paper it is shown that it is possible to construct a smooth simple global path and then modify this path locally using PSO-RBF, Ferguson splines or Bézier curves trained with PSO, in order to describe more complex paths in partially known environments. Experimental results show that our approach is fast and effective to deal with complex environments.**

I. Introduction

Considerable number of research papers exist in the field of robotic path planning. Classic methods used to solve path planning include grid based path planning algorithms as the one of the most commonly used methods [2], [3], [4], [5]. Unfortunately, even though algorithms like A* and D* are complete (they will always find a solutions if there is one), the paths obtained by this algorithms lack of smoothness; nonholonomic mobile kind of robots will often have to stop and readjust their trajectory to continue following the path with every drastic change of direction. For the modeling of the environment, the map is discretized, by doing this, lots of solutions may be excluded, and also it could cause non-smooth paths in algorithms like A*, D* and potential fields. In addition, in the case of potential fields, the algorithm could be trapped in a local minimum formed by concave obstacles [4]. Smooth paths are important in robotics because nonholonomic mobile robots are commonly used in practice and it's easier to design continuous control algorithms to follow this type of paths. Other approaches that generate smooth paths that are designed to work in global path planning environments are found in [6], [7]. Their main goal is finding smooth paths between a start and goal point. The approach presented in this paper not only seeks this, but it also gives emphasis on exploring other points of interest before arriving to the goal point.

Particle Swarm Optimization (PSO), is a method for optimization of continuous nonlinear functions, created by James Kenedy and Russell Eberhart in 1995 [8]; inspired by the social behavior of bird flocks and schools of fish. In PSO, each individual would be the equivalent of a bird of a flock, each "bird" is named "particle", and the "flock" is called a "swarm". A particle is analogous to a chromosome in Genetic Algorithms. Compared to other Evolutionary Algorithms, classic PSO has no crossover and mutation calculation, this is actually one of the things that makes it really easy to implement and fewer parameters to adjust. This makes PSO attractive for the optimizing phase, because optimal smooth paths can be obtained with lower computational effort and in a shorter amount of time. PSO only evolves the particles social behavior and their movement towards the best solutions [9]. The search can be carried out by the speed of the particle during the development of several generations, and only the most optimistic solution can pass their information over iterations.

We can find in bibliography, that PSO has proven to have good results in path planning to perform obstacle avoidance [10], [11], [12].

In the previous version of our work [1] the PSO-RBF approach for global path planning was presented. Nevertheless, our first approach was designed to plan global paths and then, as an improvement made for this approach, re-plan them using local PSO planning when a change in the environment is detected (new obstacles appear in the environment) or when the path can not be described with a smooth function.

The problem to be solved by PSO-RBF was addressed as an interpolation problem, and therefore the paths planned by the global PSO-RBF approach are simple paths, namely paths without backward movements, or paths that could be described as functions. In order to be capable of describing more complicated paths, handle unexpected changes in the environment and at the same time to take advantage of the desirable features achieved with our previous approach (such as programming simplicity, fast convergence and smooth paths planned), the use of a new algorithm with two phases for path planning is proposed. The first phase consists of the planning of a global path using the approach presented in [1]. If the path reaches all the control points of the path and it's collision-free, then the process ends with this stage. Otherwise, a second phase is performed, which consists of the execution of local path planning processes between the control points that could not be reached by the global path found in the first stage. The second stage can be executed using the PSO-RBF approach (using it in a local way), PSO-Ferguson spline or PSO-Bézier curves trained with PSO.

978-1-4799-1488-3/14/$31.00 ©2014 IEEE

II. Particle Swarm Optimization

PSO exploits a population of potential solutions, each solution consists of a set of parameters, representing a point in a search space $A \subset \Re^D$. The population of solutions is called swarm and each individual from a swarm is called a particle. A swarm is defined as a set of N particles. Each particle i is represented as a D-dimensional position vector $x_i(k)$. The particles are assumed to move within the search space A iteratively. This is done by adjusting their position using a proper position shift, called velocity $v_i(t)$.

On each iteration k, the velocity changes by applying equation (1) to each particle.

$$v_i(k+1) = \alpha v_i(k) + c_1\varphi_1(P_{ibest} - x_i) + c_2\varphi_2(P_{gbest} - x_i), \quad (1)$$

where φ_1 and φ_2 are random variables uniformly distributed within [0,1]; c_1 and c_2 are weighting factors, also called the cognitive and social parameters, respectively; α is called the inertia weight, which decreases linearly from α_{start} to α_{end} during iterations. P_{ibest} and P_{gbest} represent the best position visited by a particle and the best position visited by the swarm till the current iteration t, respectively.
The position update is applied by equation (2) based on the new velocity and the current position.

$$x_i(k+1) = x_i(k) + v_i(k+1). \quad (2)$$

The basic algorithm is as follows:

1) Initialize each particle of the swarm, with random values for position and velocity in the search space.
2) Evaluate each member of the swarm with the fitness function (which has to be designed according to the application).
3) Compare the value obtained from the fitness function of the particle i, with the value of P_{ibest}. If the value of the fitness function is better than the P_{ibest} value, this new value takes the place of P_{ibest}.
4) If the value in P_{ibest} is better than P_{gbest}, then P_{gbest} is replaced by P_{ibest}.
5) Modify the velocity and position of the particles using equations (1) and (2), respectively.
6) If the maximum number of iterations or the ending condition isn't achieved, return to step 2.

To solve the uncontrolled increase of magnitude of the velocities (swarm explosion effect), is often used to restrict the velocity with a clamping at desirable levels, preventing particles from taking extremely large steps from their current position.

$$v_i(k+1) = \begin{cases} v_{max} & if\ v_i(k+1) > v_{max}, \\ -v_{max} & if\ v_i(k+1) < -v_{max} \end{cases} \quad (3)$$

III. PSO-RBF

Path planning for car-like mobile robots can be realized through a search space of functions [13], [10]. In this case we reduce the space to a sub-space of radial basis functions (RBFs).

As shown in Fig. 1, the structure of a RBF neural network consists on three layers [14]:

- The input nodes layer.
- The hidden neuron layer.
- The output layer.

The RBF neural network is designed to perform a nonlinear mapping from the input space to the hidden space through RBFs, followed by a linear mapping from the hidden space to the output space [15].
The radial basis equation for interpolation consists in selecting f as:

$$f(x) = \sum_{i=1}^{N} w_i \varphi(\parallel x - C_i \parallel), \quad (4)$$

$$\varphi(r) = exp(-\frac{r^2}{2\sigma^2}). \quad (5)$$

where,

- $\varphi()$: Is a nonlinear function, known as a radial basis function. Equation (5) is a Gaussian function (there are other type of RBFs, like multiquadratics and inverse multiquadratics).
- w_i : Represents the weight of the connection between the neuron i from the hidden layer with the output layer.
- $\|\|$: Represents the euclidean norm.
- C_i : Are the centers of each of the gaussian functions; where $C_i \in \Re^p, i = 1, 2, 3, ..., N$.
- σ^2 : Represents the variance.
- x : Is the set of input points of the signal to approximate.
- y : Is the label or target value.

Usually, RBFs are trained by algorithms like K-means with a totally supervised learning method [16]. In the approach presented in this paper, PSO is used to replace these phases.

Due the use of a discrete map, the values used for the control points are also discrete. Each particle in the swarm represents an RBF network and therefore the structure of the particle is composed by the parameters C_i, σ^2 and w_i to be used by the RBF function (Eqs. 4 and 5), to approximate a function (smooth path) that passes by a predefined set of control points (x, y). This set of (user defined) control points

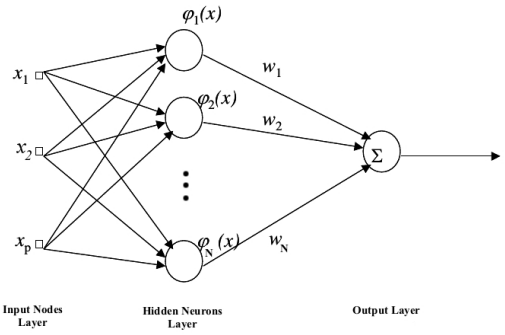


Fig. 1: RBF Neural Network Structure

is taken as the RBF input points for the input nodes layer. They represent trajectory constraints, i.e. coverage control points that can be seen as places where is desirable for the robot to explore in the environment during each navigation episode.

The input points are not static, except for the first and the last point (that are the start and goal points respectively). For each PSO execution, the rest of the points vary their position randomly with each iteration, after updating the particles position, in a range of 1 map state in any direction with respect to their original state.

The following fitness function was used for this approach

$$g = \beta_1 * RMSE + \beta_2 * l + c, \tag{6}$$

where:

- $RMSE$ is the root mean square error of the RBF neural network.
- l is the length of the path.
- c is the collision variable.
- β_1 and β_2 are scaling factors.

β_1 and β_2 are (user defined) scaling factors that weigh $RMSE$ and l and c in order that the error of the RBF neural network and the length of the path have more or less influence in the final path obtained (the scaling factors where not included in the original approach [1]). To obtain $RMSE$ we use equations (7) and (8)

$$RMSE(f) = \sqrt{\frac{\sum_{k=1}^{n}(e_k)^2}{n}}, \tag{7}$$

$$e_k = y_k - f(x_k), \quad where\ f(x_k) \approx y_k, \tag{8}$$

where $f(x_k)$ is the output of the RBF network computed for the input x_k.

The length value l is obtained as the sum of the euclidean distance between any two consecutive points that form the path

1: **for** $n = 1 \rightarrow |o|$ **do**
2: $(x', y') \leftarrow o_n$
3: **for** $j = -2 \rightarrow 2$ **do**
4: **for** $i = -2 \rightarrow 2$ **do**
5: $Map(x' + i, y' + j) \leftarrow MAX(Map(x' + i, y' + j), H(i + 3, j + 3))$
6: **end for**
7: **end for**
8: **end for**

Fig. 2: Pseudocode to compute f_m values of each state of the discretized map of the environment.

$$l = \sum_{i=1}^{n-1} \sqrt{(\Theta_{i+1} - \Theta_i)^2 + (f(\Theta_{i+1}) - f(\Theta_i))^2}, \tag{9}$$

where Θ is the set of discrete values between x_1 and x_k. To calculate c, is necessary to consider the set of states that are in a certain range of the obstacles S to obtain the sub-set $s \subseteq S$, which are states in the range of the obstacles that the path crosses

$$S = \{(x', y') | (x', y') \in obstacles\, range\}, \tag{10}$$

$$c = \sum_{k=1}^{n} Map(s_k)\ where\ s \subseteq S, \tag{11}$$

where n is the number of elements in the sub-set s. The value of c, obtained in Eq. (11), increases with the number of states that a path crosses and that are occupied by obstacles or near them.

The map is discretized, so a value $Map(x', y')$ is assigned to each state that is within a certain range from an object by using a Gaussian mask (12) and Alg. 2. The maximum value in the mask is 0.02 and it represents a state occupied by an obstacle, which are the states that correspond to the sub-set $o \subseteq S$. Any other value in the interval of $(0, 0.02)$ represents a state that is near a state occupied by an obstacle. A value of zero for a state means that it is out of range of any obstacle. A state in s can take a value from the interval $(0.0001, 0.02]$.

$$H = \begin{bmatrix} 0.0001 & 0.0006 & 0.0012 & 0.0006 & 0.0001 \\ 0.0006 & 0.0049 & 0.0099 & 0.0049 & 0.0006 \\ 0.0012 & 0.0099 & 0.0200 & 0.0099 & 0.0012 \\ 0.0006 & 0.0049 & 0.0099 & 0.0049 & 0.0006 \\ 0.0001 & 0.0006 & 0.0012 & 0.0006 & 0.0001 \end{bmatrix} \tag{12}$$

IV. Robot Path Planning using Particle Swarm Optimization of Ferguson Splines

In [12] a method for path planning, based on PSO and Ferguson splines is proposed. This approach uses cubic Ferguson splines to form a smooth path between two points, based on the fact that the problem of finding a smooth collision-free path can be seen as an optimization problem

with restrictions.

The main idea of this approach is to optimize the $P_0^{'}$ and $P_1^{'}$ values (which are tangent vectors), from the Ferguson splines equation (Eq. 13), meanwhile P_0 and P_1 remain static (they are the start and goal point of the trajectory respectively). Each Ferguson spline is defined by equation:

$$k : X(t) = P_0F_1(t) + P_1F_2(t) + P_0^{'}F_3(t) + P_1^{'}F_4(t), \quad (13)$$

$$F_1(t) = 2t^3 - 3t^2 + 1 \quad (14)$$

$$F_2(t) = -2t^3 - 3t^2 \quad (15)$$

$$F_3(t) = t^3 - 2t^2 + t \quad (16)$$

$$F_4(t) = t^3 - t^2. \quad (17)$$

where $t \in [0, 1]$. Two or more cubic Ferguson splines can be concatenated to obtain a path. Since $\tilde{P_0} = P_1$ and $\tilde{P_0^{'}} = P_1^{'}$, this values can be used for a new Ferguson spline (Eq. 18)

$$\tilde{k} : \tilde{X}(t) = \tilde{P_0}F_1(t) + \tilde{P_1}F_2(t) + \tilde{P_0^{'}}F_3(t) + \tilde{P_1^{'}}F_4(t), \quad (18)$$

where only the start and goal points of the path formed by the concatenated Ferguson splines remain static (P_0 and P_1 for the case of two concatenated Ferguson splines), the other vectors ($P_0^{'}, P_1, P_1^{'}$ and $\tilde{P_1^{'}}$) will form the model of the particle for PSO to optimize.

This approach uses the following fitness function

$$g = \frac{l}{l_{MIN}} + (\frac{\alpha}{d})^2 \quad (19)$$

where l_{MIN} is the Euclidean distance between the actual and the desired robot position, the constant α represents the influence of the obstacles, l is the length of the trajectory and d is the minimal distance between the path and the closest obstacle.

V. Modified version of Ferguson Splines Approach

To make the Ferguson splines approach useful when used in the approach proposed in this paper, some modifications were made. These changes would make the Ferguson splines approach to be evaluated more like the PSO-RBF approach. To begin with, the map is discretized like mentioned before in Sec. III and only one Ferguson spline is optimized per path between two points. The original fitness function changes to a fitness function that resembles the fitness function shown in Eq. 6, but without the $RMSE$ computation, see Eq. 20:

$$g = \frac{l}{\beta} + c \quad (20)$$

where l is the length of the path, β is a (user defined) scaling factor and c is the collision variable. Just like in the PSO-RBF approach, to obtain c we must consider the set of points inside the range of an obstacle that are crossed by the path s.

VI. PSO-Bézier curves for local path planning

A Bézier curve is a parametric curve defined by a set of control points $P_0, ..., P_n$, where n is the order of the polynomial which defines the Bézier curve. The intermediate control points generally do not lie on the curve. The formula for a $n-$order Bézier curve is shown in Eq. 21.

$$B(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i P_i \quad (21)$$

where $\binom{n}{i}$ are binomial coefficients and $t \in [0, 1]$.

In this work it was used a cubic Bézier curve, like the one shown in Eq.22

$$B(t) = (1-t)^3P_0 + 3(1-t)^2tP_1 + 3(1-t)t^2P_2 + t^3P_3 \quad (22)$$

The model of the particle for PSO contains the parameters P_1 and P_2 which are the middle control points of the Bézier curve. Points P_0 and P_3 are the initial and target points of the local path respectively and they remain static.

The fitness function used for PSO to adjust the parameters of the Bézier curve is the same that the one used with our modified approach of PSO-Ferguson Splines (see Eq. 20).

VII. From local to global path planning

The motivation behind local to global planning approach comes from the idea of simplifying a difficult task in to more simpler task to obtain an optimum result. The task of optimizing a smooth function to describe an optimum global path is a challenge; as the map grows in complexity several things could happen: the map won't be explored efficiently, there could be places where a drastic change on the path would be needed to avoid collisions and the function that describes the path won't be able to perform that change smoothly, or places where the path can't be represented as a function, etc. But, by dividing the problem, a higher flexibility is obtained to deal with this problems in a simple and more effective way.

In order to deal with more complex environments than those presented in the previous work by the authors [1], an algorithm to construct a smooth and safe global path through the union of several local paths is proposed. It must be remarked that this approach doesn't seek to obtain the shortest path, but instead, a smooth and collision-free path that explores through most of the areas that are desirable for the robot to pass (the control points).

Require: $CtrlPoints$, Map, P_{ini}, P_{end}
Ensure: Map and M are global arrays
1: Execute PSO-RBF path planning to obtain a global path GP that approximates $CtrlPoints$, for the map Map, from P_{ini} to P_{end}.
2: **if** GP has collisions or describes a trajectory that covers very few control points **then**
3: Identify all the sub-paths without collisions that pass between (or near enough) any pair of control points in GP and store them in M.
4: Set the starting control point as P_{ini} and the goal control point as P_{end}.
5: All the control points in $CtrlPoints$, except P_{ini}, are stored in the $NCPoints$ array to indicate that they will be treated as "non-covered" control points.
6: Call to algorithm $GLP = Loc2global$ (Alg. 4) with parameters ($NCPoints$, P_{ini}, P_{end}).
7: **return** GLP
8: **else**
9: **return** GP
10: **end if**

Fig. 3: Pseudocode to initialize arrays and build global paths calling to Alg.4

By treating the problem as a set of local path planning problems, a little sacrifice of the path's smoothness is made in every local division of the global path (the joints of the sub-paths that are located in the control points), but the smoothness remains on each sub-path obtained.

The initialization of the arrays to be used by the main function of the algorithm (Alg. 4 or the $Loc2global$ function) is made in Alg. 3. Which executes the PSO-RBF algorithm (described in Sec. III) in order to find a first approximation to an optimum global path between the starting control point P_{ini} and the goal control point P_{end}. It takes into consideration the initial trajectory control points from PSO-RBF $CtrlPoints$, that are the places where is desirable for the robot to explore, and the complete map of the environment Map. Where P_{ini}, P_{end}, $CtrlPoints$ and Map are defined by the user. If the PSO-RBF algorithm returns a good enough trajectory then, the algorithm will return GP as the final global path and Alg. 4 won't be executed.

On the other hand, if GP is trajectory with collisions or describes a trajectory that covers very few control points (high values of $RMSE$) then Alg. 4 is used, which takes trajectory GP as base to obtain a safe path with higher complexity that passes through most of the control points.

The parameters required by Alg. 4 are: the list of control points that haven't been reached $NCPoints$, the stack of points that had been rejected because of collisions $RejPoints$, the map with the obstacles Map, the global matrix where all the points from the collision-free sub-paths that had been found by the by PSO-RBF algorithm are stored M, the goal point P_{end} and the current point P_k (that takes the value value of the starting point of the trajectory P_{ini} at the first iteration).

Prior to the execution of Alg. 4, it is needed to identify all the sub-paths without collisions that pass between, or near enough, any two control points in GP and store them in the

Require: $NCPoints$, $RejPoints$, Map, M, P_k, P_{end}
Ensure: Map and M are global arrays
1: Set GLP to **empty**;
2: Set $RejPoints$ to **empty**
3: **while** $NCPoints \neq$ **empty** and $P_k \neq P_{end}$ and GLP = **empty do**
4: Find the closest control point P_c to P_k from $NCPoints$ using Manhattan distance computation
5: Remove P_c from $NCPoints$
6: Look for a local sub-path LSp stored in M that connects P_k with P_c
7: **if** there is not a local sub-path LSp in M that connects P_k with P_c and LSP is not $Unreachable$ **then**
8: Train an RBF or a Ferguson Spline or a Bézier curve with PSO to obtain a local path LSp between P_k and P_c with $retSubPath$ function (shown in Algs. 5, 6, 7)
9: **if** the LSp is collision free **then**
10: Add LSp to M on position (P_k, P_c)
11: **else**
12: Set $Unreachable$ value to M on position (P_k, P_c)
13: **end if**
14: **end if**
15: **if** the LSp is collision free **then**
16: Add $RejPoints$ and $NCPoints$ to $tmpNCP$
17: Recursive call to tmp =Loc2Glob($tmpNCP$, P_c, P_{end})
18: **if** $tmp \neq Unreachable$ **then**
19: Add LSp and tmp to GLP
20: **else**
21: Add P_c to $RejPoints$
22: **end if**
23: **else**
24: Add P_c to $RejPoints$
25: **end if**
26: **end while**
27: **if** $P_k \neq P_{end}$ and GLP is **empty then**
28: GLP=$Unreachable$;
29: **end if**
30: **return** GLP

Fig. 4: Function Loc2global. Pseudocode to build a global path based on local path planning

global array M; by doing this it is possible to omit the future calculation of a few sub-paths. All the control points of the trajectory are treated as non-covered control points and stored in the $NCPoints$ array, with the exception of the initial point P_{ini}. This is because when the trajectory is refined with Alg. 4, it is desirable that all the possibilities of making a path between P_{ini} and any other control point are available for consideration, because of that, the array for the rejected points $RejPoints$ is initialized as **empty**.

Alg. 4 iterates while three conditions are mantained: i) there is at least one control point in $NCPoints$, ii) the current point P_k is different from P_{end} and iii) a global path GLP has not been found.

On each iteration of Alg. 4 the array M is consulted for a local sub-path LSp between the current point P_k and the closest point to it that is contained in $NCPoints$, P_c. If a

path exists between P_k and P_c in M, we recover the set of points that were stored previously in Alg. 3. Otherwise we get an **empty** value if no attempt of forming a path between this points was detected or an $Unreachable$ value if an attempt was made, but collisions were detected.

In the case where there is no path between P_k and P_c, a call is made to the $retSubPath$ function, that will return a path (that can be obtained with PSO-RBF, modified PSO-Ferguson splines or PSO-Bézier curves) or an $Unreachable$ value if no safe path was found between this two points.

The pseudocode presented in Alg.4 constructs or refines a global path through the union of several local paths between control points. It uses backtracking combined with an heuristic method. Since the calculation of all the possible combinations of points to obtain an optimum path would be impractical (it happens when backtracking is not restricted or has no guidance), the search for an optimum path is directed always towards the nearest non covered control point in $NCPoints$ (P_c) to the actual position P_k.

If a sub-path between the current position and its nearest non covered neighbor collides with an obstacle, the nearest neighbor P_c is rejected and stored in $RejPoints$ since the sub-path generated between this two points has no possibility to be part of an optimum global path; then the next closest non covered neighbor that hasn't been rejected is taken into consideration, and so on, until a safe sub-path is found. When a safe sub-path is found, the actual position P_k takes the value of the next closest non covered neighbor P_c that hasn't been rejected. And from the new actual position, the search to reach a new position is repeated, taking into consideration for the search the control points that were rejected $RejPoints$ from the previous position (the sets of rejected points are independent for each position) and the ones that haven't been visited $NCPoints$, but excluding the already visited control points. If from the actual position no other position could be reached safely, the algorithm returns to the previous position P_{k-1}, taking it as the actual position P_k, and adds the position that leads to nowhere to the previously rejected $RejPoints$ control points from that position and continuous the search. The algorithm ends when the goal control point is reached or no possible smooth and collision-free global paths between a pair points could be generated by the algorithm.

When Alg. 4 reaches the goal point, it returns the entire set of points that form the global path GLP; or if the goal point could not be reached, then GLP takes **empty** as its value. Both modified PSO-Ferguson splines and PSO-Bezier curves can be used by this approach without changes, but certain changes must be done to PSO-RBF in order to work properly in local path planning.

To use the PSO-RBF approach for local path planning (Alg. 5), instead of training the algorithm as described in [1], the only control points that would remain completely static are P_c and P_k. The rest of the control points to approximate, are generated one for each discrete state of the map on the x axis that lays between P_c and P_k. This points remain with static discrete values on the x axis, while they are only allowed to move freely on the y axis.

The original fitness function (Eq. 6) is maintained

Require: $P_k = (x_1, y_1)$, $P_c = (x_2, y_2)$, Map
1: **if** $abs((y_2 - y_1)/(x_2 - x_1)) \leq 1$ **then**
2: $GBest \leftarrow pso(x_1, x_2, y_1, y_2, Map)$;
3: $interval \leftarrow$ obtain the set of discrete points from x_1 to x_2;
4: $F(x) \leftarrow rbf(GBest, interval)$;
5: $path \leftarrow (interval, F(x))$;
6: **else**
7: $GBest \leftarrow pso(y_1, y_2, x_1, x_2, Map^T)$;
8: $interval \leftarrow$ obtain the set of discrete points from y_1 to y_2;
9: $F(x) \leftarrow rbf(GBest, interval)$;
10: $path \leftarrow (F(x), interval)$;
11: **end if**
12: **return** $path$

Fig. 5: Pseudocode for the retSubPath function using PSO-RBF

Require: $P_k = (x_1, y_1)$, $P_c = (x_2, y_2)$, Map
1: $GBest \leftarrow pso(x_1, x_2, y_1, y_2, Map)$;
2: $path \leftarrow fSplines(GBest)$;
3: **return** $path$

Fig. 6: Pseudocode for the retSubPath function using modified PSO-Ferguson splines.

with out changes, but instead of taking into consideration the $RMSE$ for all the control points, it is only needed to evaluate the $RMSE$ for P_k and P_c.

A disadvantage of using an RBF neural network compared to a Ferguson spline or a Bézier curve is that the path obtained will only have values between P_k and P_c; then the difficulty to obtain a safe path between P_k and P_c increases as the number of discrete states in the x axis between them decreases (the space to generate a safe path becomes more restricted). But this can be solved by transposing some of the input data. If we have higher number of discrete states between P_k and P_c on the y axis than the number of discrete state on the x axis, it's preferable to work with control points that remain static on their y axis values and leave the values on the x axis move freely. This is presented in Alg.5 when $abs((y_2 - y_1)/(x_2 - x_1)) > 1$, (where $P_k = (x_1, y_1)$ and $P_c = (x_2, y_2)$), then we use the transpose of the map Map^T. This way, the algorithm can reach points where the slope between two points is higher than 1 or lower than -1.

The variable $interval$ in Alg.5 is a set of discrete points, the pso function returns the parameters of the best global path found with PSO using a RBF neural network $GBest$ and the rbf function, using the values from $GBest$, evaluates Eq. 4 using the results from $interval$, which returns the path that goes from P_k to P_c.

The pso function in Alg. 6 returns the parameters of the best global path found with PSO using Ferguson splines $GBest$ and the $fSplines$ function, using the values from $GBest$, returns a path by evaluating Eq. 13 with a set of points from the interval $[0, 1]$. Alg. 7 works in a similar way,

Require: $P_k = (x_1, y_1)$, $P_c = (x_2, y_2)$, Map
1: $GBest \leftarrow pso(x_1, x_2, y_1, y_2, Map)$;
2: $path \leftarrow Bezier(GBest)$;
3: **return** $path$

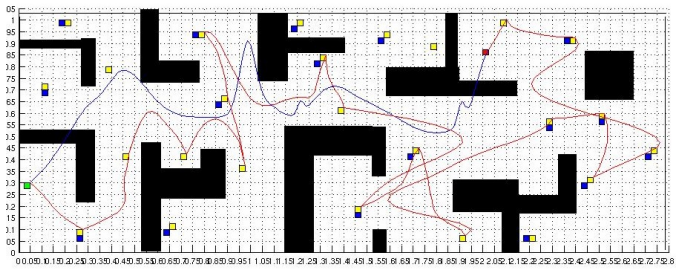
Fig. 7: Pseudocode for the retSubPath function using Bézier curve.

but with Bézier curves instead of Ferguson splines.

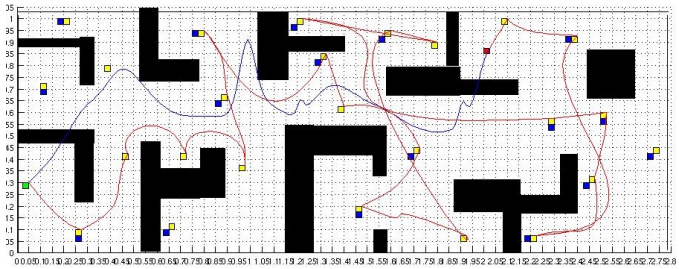
VIII. Simulation Results

The simulations of the environments and smooth paths where tested in the Webots™ robot simulator on a iRobot® Create® differential wheeled robot. For the local to global path planning approach, we used a swarm composed of 15 particles and the next set of heuristically selected parameters: $c_1 = 2$, $c_2 = 3$, $v_{max} = 0.15$, $\omega_{start} = 1$, $\omega_{end} = 0.0005$ and $sf = 15$ for the fitness function. This parameters can be used with PSO-RBF, modified PSO-Ferguson splines or PSO-Bézier curves, either way it gives good results. In Figures 8 and 9 four types of points can be seen. The start point in green, the goal point in red, the set of control points with in their initial positions in blue and in yellow their final position after the update in the first phase. The path in blue is the path obtained on the first phase, and as can be appreciated, most of the control points are not covered by this path and certain cases it is impossible to reach some points without colliding with an obstacle. The second phase solves this by constructing a path with a higher complexity (the path in red) by dividing the problem in several local path planning problems. Even if part of the smoothness is lost in the unions between sub-paths, it is preserved in each sub-path.
In Figure 8 a set of concave obstacles is divided by a hallway. Regardless of the shape, the path can handle this without getting trapped by the obstacles.
In Figure 9 the number of obstacles is increased. The path avoids colliding with the obstacles and passes safely between them if there is enough space to generate a smooth path between any pair of control points.
As can be seen in table I, the performance for this approach is measured in terms of time and the number of control points reached by the path. Since the approach gives importance to explore other points of interest before getting to the goal point, the number of control points reached is proportional to the zones of interest in the map that can be explored by the mobile robot without the risk of colliding. If each point represents a possible location where a person could be trapped in a hazardous zone, a higher number of points covered by the path increases the chances of finding the person on the first attempt.

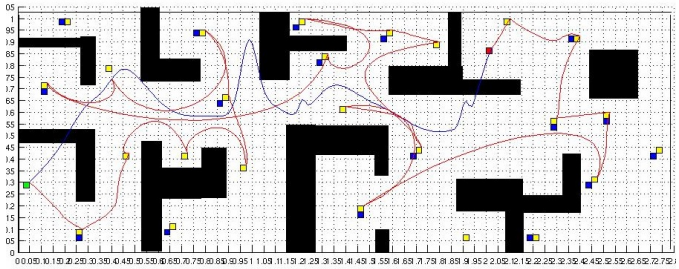
The modified PSO-Ferguson Spline and the Bézier curve approaches are slower in most of the cases than PSO-RBF approach by a considerable margin, which makes PSO-RBF more suitable to be implemented in real time. On the other hand, with modified PSO-Ferguson splines and PSO-Bézier curves is more likely to reach more control points. From the three local path planers, PSO-Bézier curves tends to be the most balanced in execution time and the number of average control points reached by the planner, and generates paths that



(a) PSO-RBF



(b) Modified PSO-Ferguson splines



(c) PSO-Bézier curves

Fig. 8: Map 3 dividing the global path planning problem into local path planning problems.

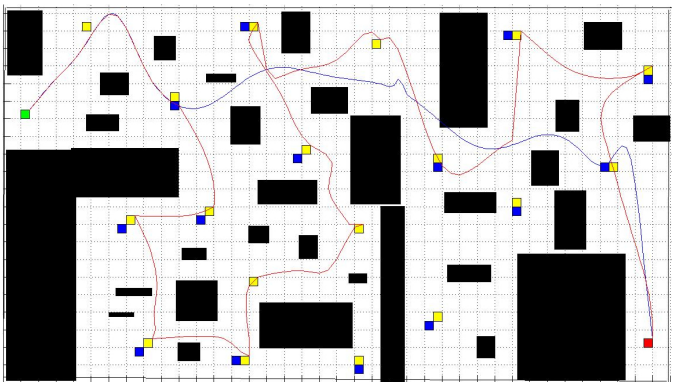
oscillate less than the ones described by PSO-Ferguson splines and PSO-RBF.

IX. Conclusions

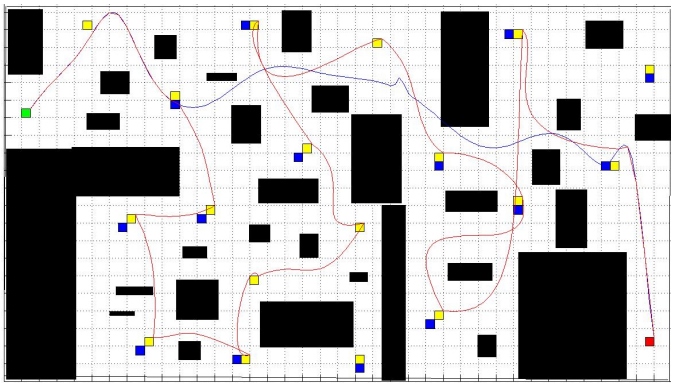
This paper presents an approach to solve the global path planning problem as an unification and optimization problem of several local path planning problems by using different types of local path planners. A set of trajectory constraints based on coverage control points as input pattern, which can be seen as places where is desirable for the robot to explore were used to construct global paths, in order to obtain complex, smooth and collision-free paths. Results in simulation environments show that our approach obtains this paths regardless of the concavity of the obstacles or the number of these taking advantage of the using of coverage control points as trajectory constraints. Furthermore, the option obtaining paths in a short amount of time or the possibility to cover more control points is available.

References

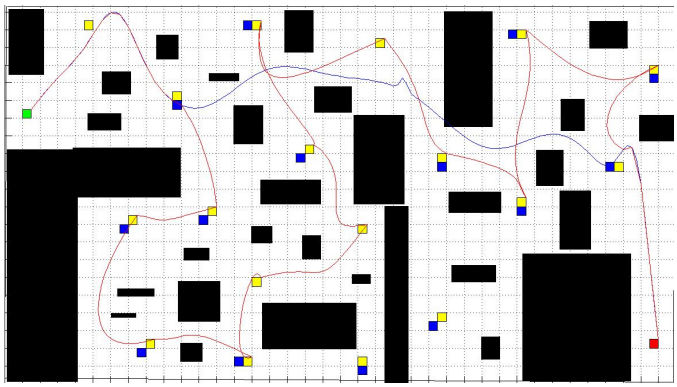
[1] N. Arana-Daniel, A. A. Gallegos, C. Lopez-Franco, and A. Y. Alanis, "Smooth Path Planning for Mobile Robot Using Particle Swarm Optimization and Radial Basis Functions," in *International Conference on Genetic and Evolutionary Methods (GEM12)*, 2012, pp. 84–90.



(a) PSO-RBF



(b) Modified PSO-Ferguson splines



(c) PSO-Bézier curves

Fig. 9: Map 4 dividing the global path planning problem into local path planning problems.

TABLE I: Comparison of local planners using PSO as training algorithm to obtain global paths

Map Number	Average time in sec.	Average of number of control points reached
PSO-RBF		
1	5.47	16.79 of 20
2	14.05	14.88 of 19
3	12.00	21.06 of 27
4	17.21	16.36 of 20
Modified PSO-Ferguson splines		
1	13.18	17.53 of 20
2	18.83	15.78 of 19
3	30.88	20.30 of 27
4	27.42	16.87 of 20
PSO-Bézier curves		
1	11.15	17 of 20
2	13.83	18.38 of 19
3	19.11	21.36 of 27
4	21.22	17.55 of 20

[2] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *Systems Science and Cybernetics, IEEE Transactions on*, vol. 4, no. 2, pp. 100–107, 1968.

[3] A. Stentz, "Optimal and efficient path planning for unknown and dynamic environments," DTIC Document, Tech. Rep., 1993.

[4] J. Barraquand, B. Langlois, and J. Latombe, "Numerical potential field techniques for robot path planning," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 22, no. 2, pp. 224–241, 1992.

[5] Y. Wang and G. S. Chirikjian, "A new potential field method for robot path planning," in *Proc. IEEE Int. Conf. Robotics and Automation ICRA '00*, vol. 2, 2000, pp. 977–982.

[6] H.-C. Huang and C.-C. Tsai, "Global path planning for autonomous robot navigation using hybrid metaheuristic GA-PSO algorithm," in *SICE Annual Conference (SICE), 2011 Proceedings of*, Sept 2011, pp. 1338–1343.

[7] Y.-J. Ho and J.-S. Liu, "Collision-free Curvature-bounded Smooth Path Planning Using Composite Bezier Curve Based on Voronoi Diagram," in *Proceedings of the 8th IEEE International Conference on Computational Intelligence in Robotics and Automation*, ser. CIRA'09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 463–468. [Online]. Available: http://dl.acm.org/citation.cfm?id=1811259.1811353

[8] J. Kennedy and R. C. Eberhart, "Particle Swarm Optimization," in *Proceedings of IEEE International Conference on Neural Networks*, vol. 4. Washington, DC, USA: IEEE Computer Society, November 1995, pp. 1942–1948.

[9] E. Elbeltagi, T. Hegazy, and D. Grierson, "Comparison among five evolutionary-based optimization algorithms," *Advanced Engineering Informatics*, vol. 19, no. 1, pp. 43–53, 2005.

[10] M. Hua-Qing, Z. Jin-Hui, and Z. Xi-Jing, "Obstacle avoidance with multiobjective optimization by PSO in dynamic environment," in *Proceedings of International Conference Machine Learning and Cybernetics*, vol. 5, Luoyang, China, 2005, pp. 2950–2956.

[11] L. W., Y. L., H. D., and Y. X., "Obstacle-avoidance Path Planning for Soccer Robots Using Particle Swarm Optimization," in *Proceedings of IEEE International Conference on Robotics and Biomimetics, ROBIO 2006*, 2006, pp. 1233–1238.

[12] M. Saska, M. MacaÌs, L. Preucil, and L. Lhotska, "Robot path planning using particle swarm optimization of Ferguson splines," in *Emerging Technologies and Factory Automation, 2006. ETFA'06. IEEE Conference on*. IEEE, 2006, pp. 833–839.

[13] Y. Hu and S. X. Yang, "A Knowledge Based Genetic Algorithm for Path Planning of a Mobile Robot," in *Proceedings of IEEE International Conference on Robotics and Automation*, New Orleans, 2004, pp. 4350–4355.

[14] E. N. Sánchez and A. Alanis, "Redes neuronales: conceptos fundamentales y aplicaciones a control automático," *Cinvestav Unidad Guadalara. Editorial Prentice Hall*, 2006.

[15] S. Haykin, *Neural networks and learning machines*. Prentice Hall, 2009, vol. 3.

[16] J. A. Hartigan and M. A. Wong, "A K-Means Clustering Algorithm," *Applied Statistics*, vol. 28, pp. 100–108, 1979.