14

Modification of Kohonen Rule for Vehicle Path Planing by Behavioral Cloning

Ranka Kulić
Faculty of Maritime of Studies
Montenegro

Abstract

The problem of path generation for the autonomous vehicle in environments with infinite number obstacles is considered. Generally, the problem is known in the literature as the path planning. This chapter treated that problem using the algorithm, named MKBC, which is based on the behavioral cloning and Kohonen rule. In the behavioral cloning, the system learns from control traces of a human operator. Kohonen rule connected with the weighting coefficients, while the MKBC algorithm does not use the weighting values as values from the previous time, but permanentlly uses the training values as weighting values. That is something which enables an intelligent system to learn from the examples (operator's demonstrations) to control a vehicle in the process of the obstacles avoiding, like the human operator does. Like that, the very important MKBC characteristic is the symplicity. The MKBC simplicity is something which is so obviously, specialy according to the RBF neural network and the machine learnig algorithm which is used the previously. Following the MKBC given context the problem narrow passage avoiding and the goal position reaching fundamentally is observed. Namely, defining if – then rule, according to the named cases is treated as destroying of the consistency of the methodology. In that sense, using MKBC neural network the solution was found. A the end, the autonomous vehicle mathematical model which is given by nonlinear equations describing a 12 state dynamical system is used and in that case the MKBC algorithm is applied successfully. Eventually, as it has been illustrated the previously, the advantage of the entire methodology lies in the fact that a complete path of the vehicle can be defined off-line, without using sophisticated symbolical models of obstacles. These are facts that MKBC algorithm and the given methodology substantially differ from the others. In the next phase it is expected to confirm results in on – line simulation process.
Key words: vehicle path planning, behavioral cloning, cloning success, obstacle avoiding, machine learning, Kohonen rule, neural network, Shark dynamical model.

1. Introduction

In the last years en increasing interest in mobile robots has appeared, notably in aeronautical space exploration, automatized agriculture, collective mobile robot games, and so on (P. Vadakkepat, X. Peng et al., 2007). These application require the mobile robot to move in partially known environments with the high amount of uncertainty. Moving – obstacle

avoidance with unknown obstacle trajectory remains remarkable challenge and has opened a research area in the control of the mobile robots, but it is treated in our research using the methodology which is connected with unmoving obstacles. Many approaches are available to study this research area. In its simplest form, the motion planning problem can be defined as follows (C. Latombe, 1991; J. Schwartz, M. Sharir, J. Hopcroft, 1987). Let B be the autonomous vehicle consisting of collection of rigid subparts having a total k degrees of freedom, and let B be free to move in two - or three - dimensional space V, avoiding obstacles whose geometry is known. For a given initial position S and a desired target position G of B, the task is to determine whether there exist a continuous obstacle - avoiding motion of B from S to G, and if so, to find such a motion. The simplest collision avoidance algorithm fall into the generate and test paradigms. A simple path from S to G, usually a straight line, is hypothesized and then it is tested for potential collisions between B and obstacles. If collision is detected, a new path is proposed using information about detected collision. This process repeats until no collision is detected. But in spite of its simplicity these methods have not found significant application. They have several fundamental drawbacks. One of these is inability to propose a radically different and better path from local information about potential collision. Another is that collection methods are based on a configuration space approach(J. Reif, 1987; C. Latombe, 1991; J. Schwartz, M. Sharir, J. Hopcroft, 1987). The configuration of rigid body is set of independent parameters that characterize the position of every point of it. For the vehicle B some regions represent illegal configuration space because there are obstacles. So the find path (the vehicle motion planning) approach means that the vehicle have to be shrank to dimension of a reference point and to grow obstacles, i.e. to compute forbidden regions for the reference point. Finding the path of the vehicle is in this way transformed in finding the path of the reference point, moving in configuration space and avoiding obstacles. For the vehicle B moving from position S to position G the desired path is the shortest path which takes into account all constraints of the position of the reference point of B. It is possible to obtain this path by generating an appropriate graph (visibility graph, connectivity graph,…) and finding a path from graph node S to graph node G. Fundamental problem arising during the implementation of these methods is concerned with the obstacle growing and graph searching. For both cases the problem complexity is very large. For example, while in the planar case the shortest path can be found in time that is in the worst case the quadratic in the number of obstacle vertices and edges, finding the shortest path between two points in three dimensions, which avoids a collections of polyhedral obstacles is NP - hard (J. Schwartz, M. Sharir, and J. Hopcroft,1987; J. Reif,1987). This is a specially very large problem if the world model changes. Other classes of approaches are developed as alternative to the traditional ones. A typical such approach (O. Khatib, 1986) regards the obstacles as the sources of repelling potential field, while the goal position G of the vehicle is considered as a strong attractor. The vehicle B follows potential gradient vector field. These approaches try to find the local minimum only. As the next, we can consider the direction which assumes problem solution capability of the vehicle motion planning based on the transfer of skill into controllers of the vehicle (D. Michie, R. Camacho, 1994; C. Sammut, S.Hurst, D. Kedzier, D. Michie, 1992).

2. Motion planning based on the behavioral cloning

Skill is often defined as an ability to perform a high quality sensory-motor coordination and control in real time. Humans exhibits such a skill as a result of training over a period of time. It would be especially useful if we can also provide systems with the capability of acquiring such a skill. In this sense two approaches have been known. The former treats the skill as something that could be acquired in a dialogue with an operator. In that process it is expected from the operator to describe skills he has been governed over the control of the vehicle. Here arises some difficulties because the skill is human subconscious action and so cannot be completely consciously and reliably described. An alternative approach is to start from the assumption that the skill can be reconstructed, using learning algorithms, from the manifestation trace of it (D. Michie, R. Camacho, 1994; C. Sammut, S.Hurst, D. Kedzier, D. Michie, 1992). Sammut, Hurst, Kedzier and Michie give a description of the solution belonging to the flight control area. Our idea (R. Kulic, Z. Vukic, 2006) is to enable an intelligent system to learn from the examples (operator's demonstrations) to control a vehicle avoiding obstacles, like the human operator does.
In section III the intelligent controller concept is given. In section IV the results in controller development are presented. In section V the conclusion is given and possibilities of further development are discussed.

3. Elements of concept controller development

3.1 Learning problem

Suppose that is given a data set giving living area and price of m houses in some place. How can is possible learn to predict the prices of other houses in that place, as function of living areas? To denote the input variables or input features (living areas in this case) x_{1i} is used. And y_i is used to denote the output or target variable which is need training to predict. The dataset $\{(x_i,y_i), i=1,..,m\}$ that will be used to learn is called a training set. If X denote the space input values and Y denote the space output values the goal is for given a training set to learn a function h: $X \rightarrow Y$. The function h(X) is called a hypothesis and it have to be a good predictor for the corresponding values of Y. When the target variable is continuous, the learning problem is called a regression problem. When the target variable take on only small number of discrete values, the learning problem is called a classification problem. Lets consider a slightly richer dataset with a number of bedrooms in each house. The X is two – dimensional vector, with x_{1i}, x_{2i} features. In general, when designing learning problem, it is up to us to decide what features to choose. To perform learning it is have to decide how we are going to represent hypothesis *h*. A choice can be to approximate it as a linear function of $x = \{ x_{1i}, x_{2i} \ldots x_{ni} \}$:

$$h(x) = \theta_0 + \theta_1 x_{1i} + \theta_2 x_{2i} + .. + \theta_n x_{ni}$$

The θ_i 's are the parameters (or weights) parameterizing the space of linear functions mapping from X to Y. To simplify notation it is introduced $x_{i0}=1$. So that

$$h(x) = \sum_{j=1}^{n} \theta_j x_{ji},$$

where n is the number of input variables , without x_0. For given training set, how it is possible learn the parameters θ_j? One reason is to make $h(X)$ close to Y, at least for the training example. To formalize this, it can be defined a function that measures for each value of the θ_j how close $h(x_i)$ to the corresponding y_i. So, the least –squares cost function is defined:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h(x_i) - y_i)^2 .$$

That function is a special case of a much broader family of algorithms. Now, it is needed to choose vector θ to minimize J(θ). To do so, lets us a search algorithm that starts with some initial guess for θ and repeatedly changes θ to make J(θ) smaller, until it is reached the value of θ that minimize J(θ). Lets consider the gradient descent algorithm, which starts with some initial θ, and repeatedly perform the update:

$$\theta_j = \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta).$$

This update is simultaneously performed for all values of j=0,…,n. The learning rate α repeatedly changes in the direction of steepest decrease of J(θ). For one training example we have:

$$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{\partial}{\partial\theta_j}\frac{1}{n}(h(x)-y)^n = n\frac{1}{n}(h(x)-y)\frac{\partial}{\partial\theta_j}(h(x)-y) =$$

$$(h(x)-y)\frac{\partial}{\partial\theta_j}\left(\sum_{j=0}^{n}\theta_j x_{ji}\right) = (h(x)-y)x_j$$

For single training example that gives the rule:

$$\theta_j = \theta_j + \alpha(y - h(x))x_j.$$

This is called LMS (least mean squares) update rule or Widrow – Hoff learning rule. For instance the magnitude of the update is proportional to the error $(y_i - h(x_i))$. For m training example the corresponding update rule

Repeat until convergence exist {

$$\theta_j = \theta_j + \alpha\sum_{i=1}^{m}(y_j - h(x_j))x_{ji} \text{ for every j}$$

}.

The optimization problem have been given here for linear regression has only one global optima. Assuming the learning rate not too large this batch gradient descent method always converges to the global minimum.

3.2. Bihevioral cloning process

The idea of controller development by cloning the human operator (D. Michie, R. Camacho,1994) is illustrated in Figure 1.

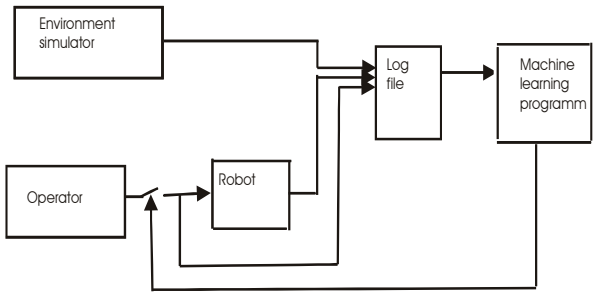


Figure 1. Behavioral cloning process. The autonomous underwater vehicle is named as the robot

In the obstacles avoiding problem, this idea could be interpreted in the following way. During one of the simulation phases, called the training phase, operator guides the vehicle avoiding unmoving obstacles located in its working space. During this phase variables that are evaluated as relevant are written into LOG FILE. In the second simulation phase, called the learning phase, the machine learning program, takes data from the LOG FILE, generates differential equations, that define the operator's trajectory. In the third phase, called the verifying phase, operator is excluded from the vehicle control process and the vehicle is controlled solely by a clone induced in the learning phase. This development of process phases are needed for the repeated changing of both problem domain representation and/or learning system regarding cloning success criterion. Using "several" vehicle models (problem domain representations) and "several" machine learning systems, we attempt to find an appropriate domain model and an appropriate machine learning system that will enable the vehicle to avoid obstacles according to cloning success criterion.

3.3 The vehicle kinematical model

The following kinematical model of the vehicle is used:

$$\psi(n) = \psi(n-1) + \Delta t\, r(n),$$

$$x(n) = x(n-1) + \Delta t\, v \cos(\psi(n)),$$

$$y(n) = y(n-1) + \Delta t\, v \sin(\psi(n)),$$

where: ψ is the heading angle of the vehicle (ψ=0 if the vehicle is oriented parallel to x-axis); r and v are control variables i.e. desired rotation speed and translation speed respectively; x, y are position coordinates, Δt is the sampling time and n is the time index. The vehicle is represented as a geometrical figure. Its dimensions are not neglected and we should point out that this is a very important fact. The selection of the vehicle model is inspired by conventional methodology that is used in control systems for a given path (R. Stojic, R. Kulic, M. Zivanovic, 1990).

3.4 Environment models

Environment model amounts to the distances of the vehicle gravity center from the goal position (dx_G and dy_G) and from the obstacles (d_i). dx_G and dy_G are calculated as:

$$dx_G = x - x_G,$$

$$dy_G = y - y_G ,$$

where x_G, y_G are the goal position coordinates. Obstacles are represented by its characteristic values, as illustrated in Figure 2. Obstacle area is divided into sub-areas .

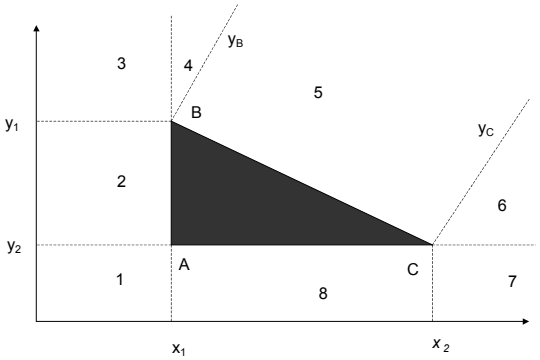


Figure 2. Triangular obstacle as the environment

A procedure, for the simulation purpose, calculating the vehicle distance d_i from i-th obstacle is explained for a triangle obstacle as:
SubArea-4:
if$((x>=x_1)$***and***$(y>=y_B))$ ***then*** $d_i = ((x-x_1)^2 + (y-y_1)^2)^{1/2}$, SubArea5:
if$((x>=x_1)$***and***$(y<y_B)$***and***$((y>=y_C)$ ***then***
$d_i = |[y+[(y_1-y_2)/(x_2-x_1)]x +[(y_2-y_1)/(x_2-x_1)]x_1-y_1]/[[(y_1-y_2)/(x_2-x_1)]^2+1]^{1/2}|$,
y_B and y_C are lines that are normal onto the line BC at points B and C.

3.5 Cloning success criterion

Performance error is very important for the evaluation of the quality of clone that was constructed. Regarding the ideal case the goal concept and the approximation concept of the vehicle trajectory are identical and the performance error is equal to zero. Ideal trajectory in x-y plane without obstacles is, for example, a straight line between the start S and the goal G positions of the vehicle. Operator, in a training phase, mostly does not manage to realize this trajectory. Position error E_{xy} is based upon a distances $d_{op(i)}$ and $d_{cl(i)}$ of operator and clone trajectories, respectively, from the named straight line. Our problem is to avoid obstacle and so we can consider only E_{perf} as:

$$E_{xy} = \frac{\sum_{j=1}^{N} \frac{\left|d_{op}(j) - d_{cl}(j)\right|}{\max(d_{op}(j) - d_{cl}(j))}}{N} \tag{1}$$

For avoiding n obstacles we have to find $(d)_m = \min\{ d_i , i=1,..n\}$ in order to define:

$$E_{xy} = \frac{\sum_{j=1}^{N} \frac{\left|(d_{cl}(j))_m - (d_{cl}(j-1))_m\right|}{\max(d_{cl}(j))_m - (d_{cl}(j-1))_m}}{N-1} \quad (2).$$

Eventually, we can say that between two clones more successful is the one which produces lower performance error regarding equations (1) and (2).

3.6 The vehicle goal position as the obstacle

The vehicle goal position can be treated as the obstacle, but it has not been applid earlier in [12], [13]. In that sense the distance from the vehicle goal position and the vehicle gravity centre was taked into account, as d_{goal}. Now the attribute number is extended and ¨minimum distance¨ is determined by relation: $d_{min} = \min \{ d_{goal}, d_1, d_2 , ...,d_k \}$, where d_1, d_2 , ...,d_k are the mimial distance for avoiding K obstacles. Eventually, when we consider the vehicle goal position as the obstacle, we enable the vehicle path generation to become simple, but not to become optimal.

3.7 Learning systems

3.7.1 Radial basis function (RBF) of neural network

The model is commonly referred to as the radial basis function (RBF) network. The most important attribute that distinguishes the RBF network from earlier radial based models is its adaptive nature. It generally allows to utilize a relative small number of locally tuned units. RBF network were independently proposed by several authors (D. S. Broomhead, D. Lowe ,1988; S. Lee, R. Kill, 1988; M. Niranjan , F. Fallside,1988).The following is a description of the basic RBF architecture Figure 3. The RBF network has a feedforward structure consisting of a single hidden layer of Q locally tuned units, which are interconnected to an output layer of L linear units. All hidden units simultaneously receive R dimensional real –valued input vector p. Notice the absence of hidden layer weights. Each hidden unit output a_j is obtained by calculating the closeness of the input p to n dimensional parameter vector μ_j (IW in Figure 2). This parameter is associated with jth hidden units. The response characteristics of the jth hidden units are given by

$$a_j[p] = K(-\frac{\| p - \mu_j \|}{2\sigma_j^2}),$$

where K is a strictly positive radially symmetric function (kernel) with a unique maximum at its center μ_j and which drops off rapidly to zero away from the center. The parameter σ_j is the width of the receptive field of the input space for unit j. This means that a_j has an appreciable value only when distance $|| p- \mu_j ||$ is smaller than the width σ_j. A specially but commonly used RBF network assumes a Gaussian basis function for the hidden units, i.e.:

$$a_j[p] = \exp(-\frac{\| p - \mu_j \|^2}{2\sigma_j^2}),$$

where σ_j and μ_j, are the standard deviation and mean of the jth unit. The norm is the Euclidian norm. The output of the RBF network is the L dimensional vector a2, which is given by:

$$a2 = \sum_{j=1}^{L} LW_{ij} a_j \ .$$

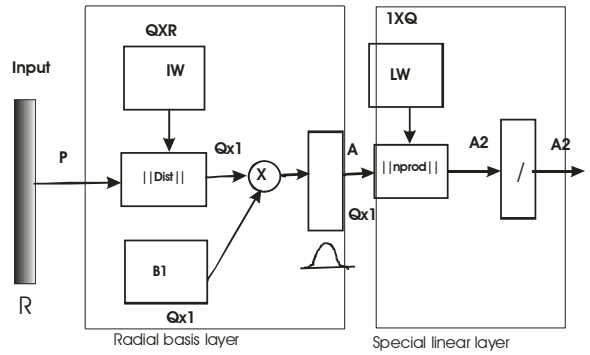


Figure 3. A radial basis function neural network consisting of a single hidden layer of locally tuned units that is fully interconnected to an output layer of linear units

RBF networks are best suited for approximating continuous real valued mappings f : $R^n \rightarrow R^L$, where n is sufficiently small. According to the previously named equations the RBF network may be as approximating a desired unction f(p).The degree of accuracy can be controlled by three parameters the number of basis functions to be used, their location and their width. The RBF networks are considered as universal approximators (T. Poggio, F. Girosi, 1990). The training of RBF network is addressed. .Consider training set of m labeled pairs $\{x_j, y_j)$ which are represent samples of a continuous multivariate function. The criterion function is an error function E to be minimized over the given training set. It is desired to develop a training method that minimizes E by updating the free parameters of the RBF. These parameters are σ_j, μ_j and w_{ij}. One of the first training methods that comes to mind is a fully supervised gradient descent methods over E, as it is given in section 3.1.

3.7.2 RBF neural network algorithm and behavioral cloning

The algorithm RBF neural network algorithm is given below for kinematical model which is given in 3.3.

```
for m:=1 to N  do                          % N is number of training examples
  begin
    dist[m,1] := (0.8326 / spread)^2 * (IW[m,1] – d_min )^2;     % d_min = d_min ;
     a[m,1] := e^-dist[m,1];               % IW[m,1] is the training examples set of the distance d_min
  end;
for j :=1 to N  do
  begin
    a2  := a2+ LW[1,j] * a[j,1];
    am :=  am + a[j,1];
  end;                                     %  LW[m,1] is the training examples set of the angle ψ
```

if ((am > 0) or (am <0)) then a2 := a2/am;
psides := a2;

r := (psides –psi) / dt; % $r = \frac{\psi_{des} - \psi}{\Delta t}$, ***dt= Δt=0.05 s***

Control strategy was r = (ψ_{des} - ψ) / Δt, v=0.1, where ψ_{des} is the desired value of angle ψ.

3.7.3 Modification Kohonen rule and behavioral cloning - MKBC algorithm

Thus, as it is given in Figure 3, the neuron whose weight vector was closest to the input vector is updated to become even more closer. Suppose that the i^{th} neuron wins, then the elements of the i^{th} row of the input weighting matrix are adjusted as shown:

$$_iLW^{1,1}(q) = {}_iLW^{1,1}(q-1) + \alpha(p(q) - {}_iLW^{1,1}(q-1)),$$

where p(q) is input vector, q is the time index and α must be specially tuned according to the domain. With regarding to the original Kohonen rule connected with the weighting coefficients, as the neural network, ***this algorithm does not use the weighting values as the values from the previous time, but permanentlly uses the training values as the weighting values. This enables an intelligent system to learn from the examples (operator's demonstrations) to control a vehicle in avoiding obstacles, like the human operator does***. This seems, at least for this case, to be better according to the original idea of the Kohonen rule. The algorithm is given below.

```
for i:=1 to M do
begin
AS(1,1) :=0;
end;
 alfa(i,n);              % Must be specially  tuned
                         % according to the domain;
                         % n is  the time index;
for i:=1 to M do       % M is the number of inputs;
begin
for j :=1 to N  do     % N is the number of the
                       %training  examples;
  begin

    LW[i,j,n] :=  alfa[i,n] * ( p[i,n0]  -  p[i,n]);  %  p[i,n] is the  input
                                                      %  vector;
                                                      %  p(i,n0) are the training example set
   AS(i,n) :=  AS(i,n) + ( LW[i,j,n0] – LW(i,j,n))^2 ;  % LW[i,j,n0] is the training examples set

  end;
   a(i,n) := sqrt (AS(i,n));                          %  a(i,n) is the  output
                                                      %  vector;
end;
```

The algorithm means that by using N training examples the weight vector *LW [i,j,n0], {i=1,M; j=1,N}* can be formed[1]. Then *LW[i,j,n0]* need to be modified according to: 1) the actual values of the input vector[2] *p[i,n], {i=1,M}* and 2) the actual value of the tuned vector *α[i,n],{i=1,M},* where *n* is the time index. The resulting weighting vector is *LW[i,j,n]* and it has deviation according to the input vector *p[i,n].* The output vector is *a[i,n]* and it is connected with the weight vector as is illustrated by two last lines of the algorithm.

3.8 The MKBC algorithm and the motion planning

The MKBC algorithm is needed to be adapted in order to be used in the motion planning domain. The motion planning with the ***weight vector*** uses LW (Table 1), the training values of the learned variable which is the heading angle ψ (Section 3.3). Firstly, the algorithm requires the modification of the training vector LW. The modification executes, as is given below, using: 1) N training examples in IW and LW, where IW are the training values of the minimal distance d_{min} (section 3.6), 2) the current input value d_{min} and 3) tuning factor α . Secondly, the algorithm finds the desired output value using, in some sense, least – square cost function of the heading angle. The algorithm output value enables the obstacle avoidance and it is the desired value of the heading angle (ψ_{des}). The algorithm is given below.

AS :=0;
*alfa = K_c – K_a * d_min; % The values K_c and K_a must be tuned;*

for j :=1 to N do % N is the number of the training examples;
begin

LW[2,j] := alfa * (IW [1,j] – d_{min});
AS := AS + (LW[2,j] – LW[1,j])^2 ;

end;
psides := sqrt (AS); % $\psi_{des} = \sqrt{AS}$;
% r := (psides –psi) / dt; , dt= Δt=0.05 s;

On the other hand, when the modified Kohonen rule with the weighting parameters is used, then the algorithm does not use the weighting values IW as the distance d_{min} from the previous time, but always uses IW as the training values of the distance d_{min} to determine the training values LW of the heading angle ψ . This enables an intelligent system to learn from the examples (operator's demonstrations) to control a vehicle in avoiding obstacles, like the human operator does. The very important MKBC characteristics are the operator cloning and symplicity, the simplicity specialy according to the RBF neural network. The coefficient α has values that are changed from time to time. Firstly, when it is tuned this factor enables the heading angle ψ to increase when the vehicle distance from the obstacles edges is small (i.e. $d_{min} \approx 0.08$). The performance error E_{xy} has a minimum for that α value. While α factor increases, the total time T of the autonomous vehicle moving without touching the obstacle also increases.

[1] *n0 means the training example set.*
[2] *The training value of the input vector is p[i,n0].*

4. Experiments and results

In order to form training instances the task was only to avoid obstacles. The robot start position was its goal position. The minimal number of robot traveling from the start to the goal was to be one. *It is the framework for selecting appropriate training instances. The idea is to combine this control strategy with control strategy without obstacles in order to form* a full controller.
When an obstacle was included, the robot trajectory for training scene is illustrated in Figure 4.

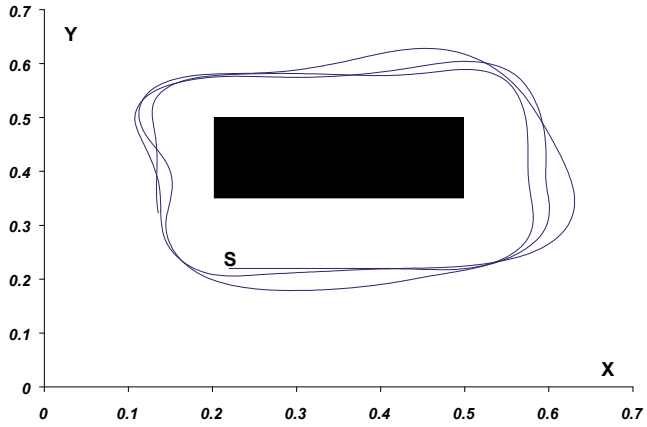


Figure 4. The robot trajectory used in order to gather training example

Control strategy was $r = (\psi_{desired} - \psi) / \Delta t$, v=0.1.

4.1 Experiments with RBF neural network

In the following is described application of RBF network in mobile robot motion planning. A relatively small corresponding training set is given in Table 1.

dex J	Distance (IW[j,1]) or d_{min}	Angle Ψ or LW[1,j]	Angle modified Ψ_m or LW[1,j]	Factor f_m
1	0.04	10.906	10.906	1
2	0.0494	13.8025	13.8025	1
3	0.0591	11.425	11.425	1
4	0.08	0.0035	81.025	23151
5	0.08	9.513	85.617	9
6	0.081	12.070	12.070	1
7	0.0894	12.692	12.692	1
8	0.0923	12.556	12.556	1
9	0.1006	7.6405	7.6405	1
10	0.12	0.6895	3.4475	5

Table 1. The original end the modified training examples according to the movement of the vehicle

The firstly by tuning spread factor (section 3.7.1 and 3.7.2) is tuned σ_j. For step k=1 spread =1, and then spread decreased in 0.01. Process was stopped for spread=0.002. The performance error E_{xy} has minima for that spread value. While spread factor decreased, total time T of the mobile robot moving without touching of obstacles increased: (spread=0.005, T=534 s; spread=0.003, T=600 s; spread=0.002, T=637 s). In order to decreased performance error for small the robot distance from obstacles edges ($d_{min} \approx 0.08$), LW[1,j] is multiplied by f_m [j] as it is given in Table 1 and in Figure 5.

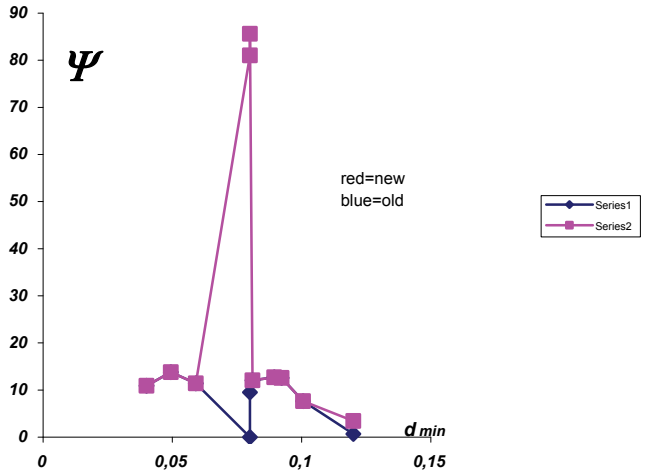
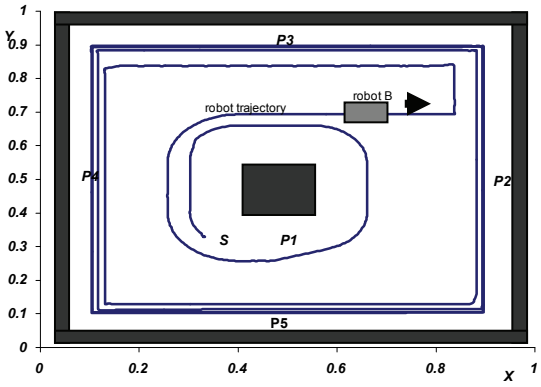
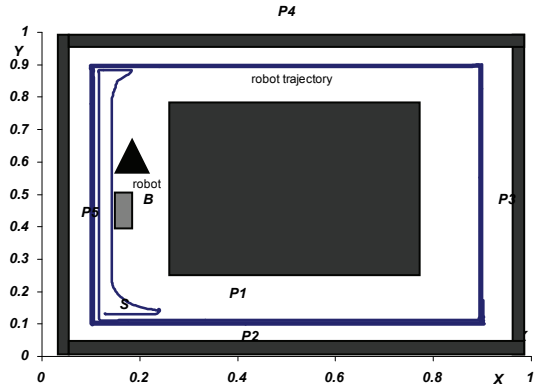


Figure 5. Modifying the LW_{ij} parameters or Ψ training instances for the RBF network in order to decrease performance error $\mathbf{E_{xy}}$

For f_m [4]=23151, f_m[5]=6 and another {f_m [j]=1,j=1,2,3,6,7,8,9,10}, it was T=1248 s. But for f_m [j] which have values as it was given in Table 1 performance error was E_{xy} = 0.00382 and robot moving was not time limited. Two scenes with five static obstacles is given in Figure 6. In that case we have situation when the mobile robot moves away from P1 obstacle to be close, until some critical distance and without touching, to obstacles P2,P3,P4 and P5. The robot safely avoids the obstacles touching. Changing f_m[j] repeatedly takes a step in the direction of steepest decrease of E_{xy}, like the cases described for function J in section 3.1. But some changes are inappropriate. Accordingly generally speaking we wished, but we could not manage, to synthesize the controller to guide the mobile robot to "oscillate" between obstacle P1 and set of obstacles P2,P3,P4 and P5. Exists the tendency of the clone to guide the robot in such a fashion that its distance from obstacle edges enhance gradually. The problem is when the controller solves turning to the left, i.e. it have to conclude that the robot distance, for example, from vertices of obstacle P4 is greater than from the robot distance from vertices of obstacle P5 . Until that the robot avoids obstacle P4 attempting to turn, not to the left, but to the right. It is reason that the robot more and more is closed to obstacles P2, P3, P4 and P5. For small distance of the robot from obstacle vertices we attempt to increase reaction time when turns around a square.



a)



b)

Figure 6. The mobile robot trajectory in xy plane for five static obstacles . The mobile robot is controlled by RBF neural network controller

4.2 MKBC algorithm as advance according to RBF neural network algorithm

The training instances Table 1 (selected 10 from 1801) the previously were tuned using RBF neural network (Section 4.1), i.e. an appropriate f_m factor was selected. In order to form training instances the task was to avoid rectangle obstacle and achieve the minimal number of one vehicle travels from the start to the goal position. On the other hand, the goal is the number of instances to be as small as possible. The MKBC algorithm (section 3.8) was tested as it is illustrated by Figure 7 for K_c =1000 *and* K_a =50 *and*

$$r = \frac{3.1623 * \psi_{des} - 0.9 * \psi}{\Delta t}$$ control strategy, for Δt = 0.05.

The vehicle start position and obstacles are given by relations:
Start S = (0.15, 0.15),
Position 1 P1 = (0.3,0.3,0.7,0.7),
Position 2 P2 = (0.0, 0.95,1.0,1.0),
Position 3 P3 = (0.95,0.0,1.0,1.0),
Position 4 P4 = (0.0,0.0,1.0,0.05),
Position 5 P5 = (0.,0.0.05, 1.0), for $v = 0.2$ *[m/s].*
For training instances given in Table 1 and for $f_m[4] = 23151$, $f_m[5] = 6$ and another $\{f_m[i] = 1, j = 1,2,3,6,7,8,9,10\}$, as it is given in Table 1 the simulation time T using the exposed algorithm was greater then about 6000 [s]. LW[1,i] is multiplied by $f_m[i]$ as is given in Table 1. For $f_m[i]$ with values given in Table 1 performance error was $E_{xy} = 0.00180$ and the vehicle movement was time unlimited. So we take that f_m is exactly as it is given in Table 1. In this case we have situation when the autonomous vehicle moves away from P1 obstacle to be closed, until some critical distance and without touching obstacles ***P2,P3,P4 and P5.*** Here, the tendency of the clone exists to guide the vehicle in such a fashion that its distance from the obstacle edges enhance gradually. The problem appears when the controller solves the turn to the left. Namely, it should conclude that the vehicle distance, for example, from vertices of obstacle P4 is greater than from the vehicle distance from vertices of obstacle P5. The vehicle avoids obstacle P4 attempting to turn, not to the left, but instead to the right. This is the main reason why the vehicle is closer and closer to obstacles P2, P3, P4 and P5. Changing $f_m[j]$ repeatedly takes a step in the direction of steepest decrease of E_{xy}, like in the cases described early. Like that, if we make the possibility that the coefficint α changes according to the d_{min} changing, we will going to induce a stroughly encahement with regard to the simulation time which is the touch free.

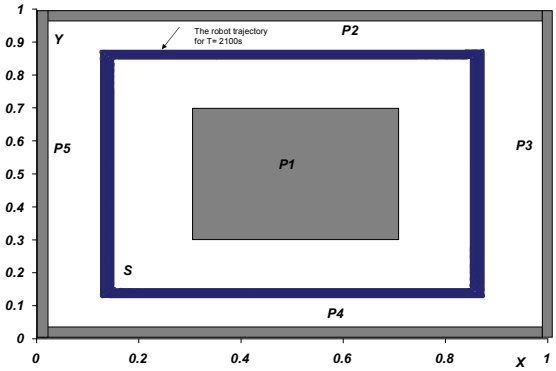
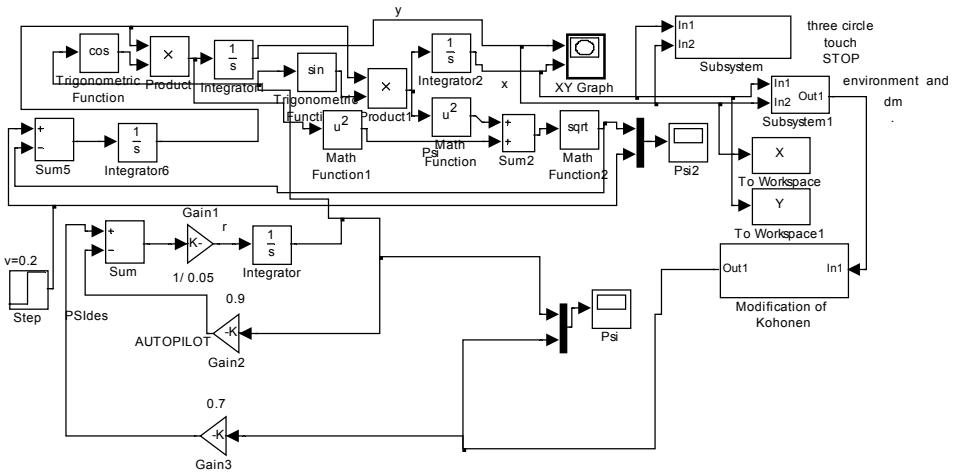
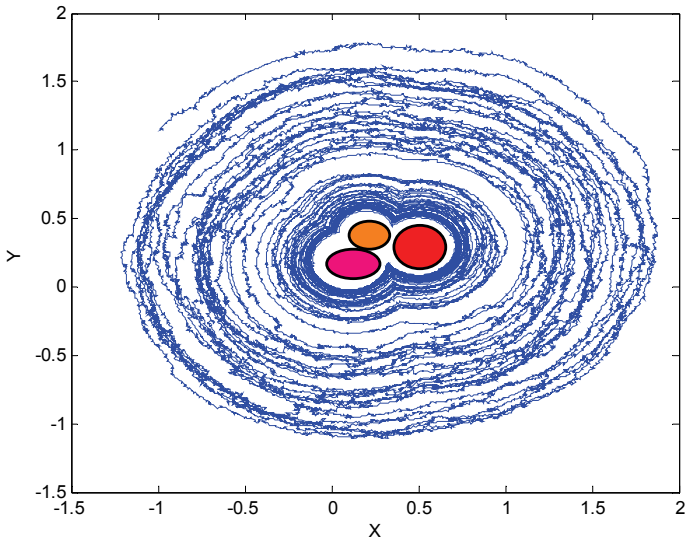


Figure 7. XY trajectory of the vehicle controlled by clone based on modified Kohonen rule

For $K_c = 1000$ *and* $K_a = 50$ and for the named control strategy we have practically infinite the time which is touch of free. The simplicity and that is reason to accept the MKBC algorithm as an advancement according to the RBF neural network. At the end, Simulink implementation MKBC for the vehicle kinematical model is very simple and it is illustrated in Figure 8, but according to Matlab integration method constants K_{cte},K_α and $K_{psidesired}$ must be corrected as K_{cte}=500,K_α=100 , $K_{psidesired}$=0.7.



a) Simulink model



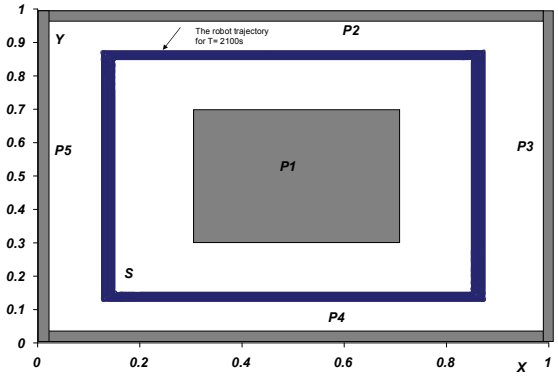
b) 2D trajectory for three obstacles when Kcte=500,K_α=100

Figure 8. Simulink model for MKBC algorithm and an appropriate result

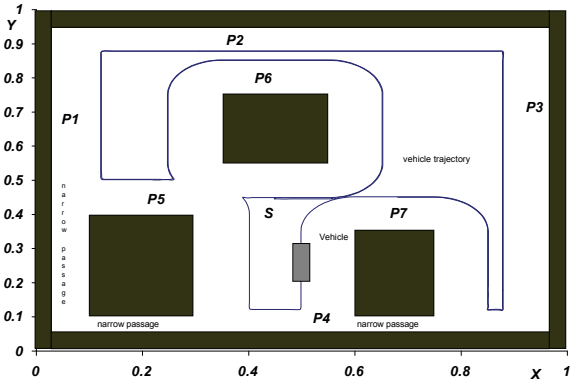
4.3 The narrow passage avoiding by MKBC algorithm

In literature (R. Kulić, 2004) is treated the narrow passage avoiding. In order to avoid the robot traveling through narrow passages between obstacles is needed to define a rule regarding dimension of the autonomous vehicle. In this section MKBC algorithm, as an advancement according to the RBF, is used. That algorithm is applied in order to solve the situation which is given in Figure 9b) -9h). In different situations which are illustrated by

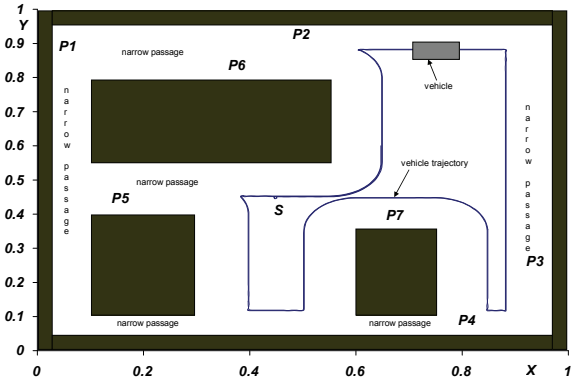
Figure 9b) – 9h) the vehicle moves from different start positions to different goal positions avoiding obstacles and narrow passages also. All experiments are quite successful. The narrow passages are avoided using the reason which is described the previously. In Figure 9b), for example, the autonomous vehicle moves away from P6 obstacle to be close P7, until some critical distance, and so on, to be close until some critical distance from obstacles P4, P3, P2, P1, …, and always changes Ψ rapidly in order to be safely avoided the obstacle touch for long time. Changes Ψ rapidly is connected with the changing $f_m[j]$ also rapidly as it illustrated in Figure 5. It is the manner repeatedly takes a step in the direction of steepest decrease of E_{xy}. According to the different situations the vehicle managed to avoid obstacles again and again, and never did not have the chance to stop, i.e. that process is has not time limiting and we observed that all experiments are quite successful. It was found that the obstacle avoiding and also the narrow passage avoiding is enabled by the same MKBC algorithm or by the methodology which is the previously described and it means that the solution consistency is saved.



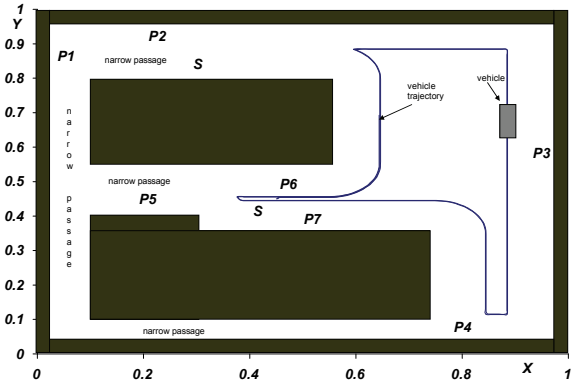
a)



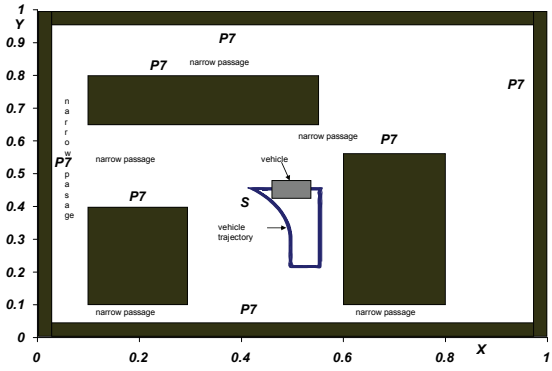
b)


Y
X
P1
P2
P3
P4
P5
P6
P7
S
narrow passage
vehicle
vehicle trajectory
narrow passage

c)


Y
X
P1
P2
P3
P4
P5
P6
P7
S
narrow passage
vehicle
vehicle trajectory
narrow passage

d)


Y
X
P7
S
narrow passage
vehicle
vehicle trajectory
narrow passage

e)