Development of the Adaptive System for Moving Control of the Mobile Robot Based on the Pioneer-2 AT

E.V. Chepin
Department of Computer Systems & Technologies
Moscow Engineering and Physics Institute (State University)
Moscow, Russia
e-mail: chepin@dozen.mephi.ru

D.A. Parkhomenko
Department of Computer Systems & Technologies
Moscow Engineering and Physics Institute (State University)
Moscow, Russia
e-mail: buchanen@mail.ru

Abstract[1]

This paper describes a kind of the hybrid approach to create a system which controls automatic movement of a mobile robot in the building. Not only practical parts of that, but mainly theoretical factors which concerns the problem are described here. This paper gives explanation for the five problems encountered during the development of robot's automatic movement control system (MCS):

- simple neural net training and it's constraints;
- route-based analysis of landscape;
- dynamic localization correction;
- hardware acceleration for MCS;
- ways for the evolution of current approach;

This paper may come useful not only for moblie robots' constructors, but also for those who is interested in combining neural algorithms with classical mathematical methods.

1. Introduction

Primarily this project was started for designing the moving control system for mobile robot that could drive inside a building. We examined some common approaches to this problem and have taken out results suitable for current situation. Visually problem looks something like Fig.1.

Main difference to most popular systems is that our MCS does not generating any raster map for calculations of the robot's path. It doesn't mean that it should not be anyway for any other purposes, but it is just excluded from the rout computating. General advantage of such decision is in great flexibility and discharging of on-board computer's resources as it will be described further.

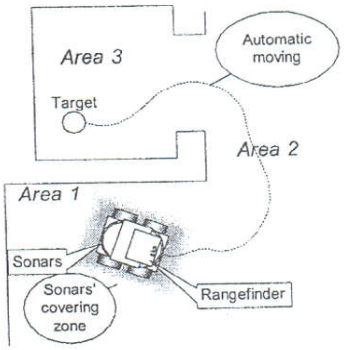


Figure 1. Mobile Robot and it's Target

Our research has also affected many ways to the optimization of solving problem. As a part of the plan for optimization we'd built and tested a hardware accelerator. It hasn't entirely satisfied our hopes, but it could be helpful for more detailed view of the problem. The developed software MCS is itself a training system that consists of the simple perceptron neuron net which potential is greatly increased by route-directed subsystem. This subsystem works with locality of given paths. Locality means an array of one's representative and distinctive points with some measured values that somehow defines the nearby land for each of them. Firstly these paths are given by the robot operator. As it will be described later the localization correcting mechanism is based on this route representation.

All software in this project (except HDL part) were written in Borland® C++ Builder. All schematics were built in Xilinx ™ IDE.

2. Suggested System's Structure Overview

There was developed a kind of an environment simulation software ESS. It is used for MCS testing and

Proceedings of the 8th International Workshop on Computer Science and Information Technologies CSIT'2006
Karlsruhe, Germany, 2006

debugging. Every screenshot that will be given below (if the source was not explicitly mentioned) is based on that software. Most of experiments were done using idea of fig. 2.

There is user interface with display where ESS shows robot model in the land model. In the real mode operator could use two web cameras attached to the robot to see where the robot is. These cams are parts of separated system which will not be described here.

Robot has WiFi extension card so the operator will contact with system by WiFi channel. The only data that transfers between on-board system and operator's computer is mode selection (training/testing system to find given target), robot direct control (when in training mode) and System parameters.

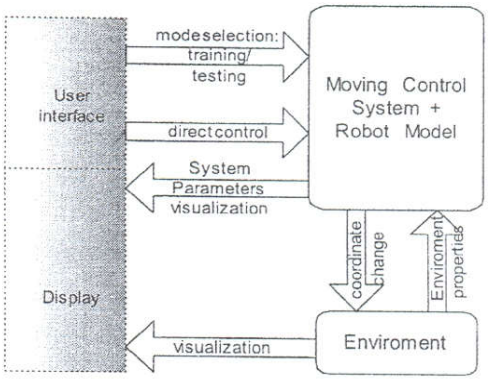


Figure 2. Simplified Experiments' Diagram

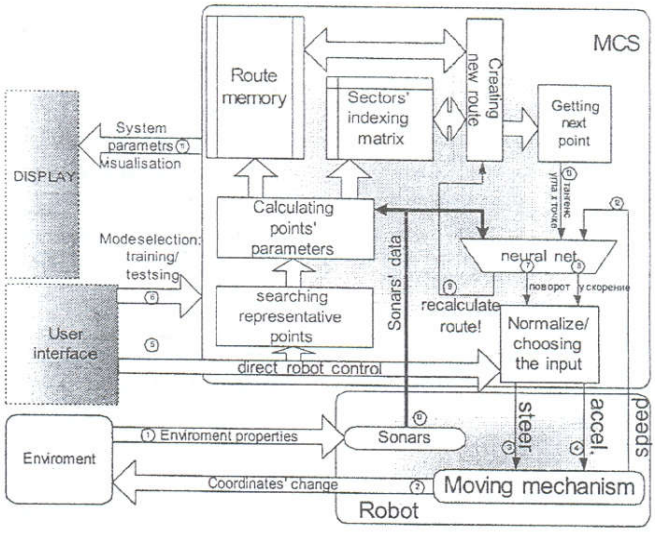


Figure 3. Structure of Suggested System

MCS Block	Comments
Normalize/choosing the input	Used for sending signals to motors from direct user control or neural net according to current mode
Neural net	See 3.
Searching repr. points	Finds points that are outstanding in terms of robot sensors
Calculating points' parameters	Storing parameters from sensors for route memory
Route memory	Saves the representative points' information
Sectors' indexing matrix	Generating matrix for optimal way lookup
Creating new route	Links representative points
Getting next point	Finding new (usually) local target for NN.

And so, the suggested hybrid system's structure in the context of such experiments looks like Fig. 3.

It is a complex composition of simple modules. We will show their work step by step. These modules will be just declared here.

3. Classical Neural Algorithm

Here will be described idea and testing results of the MCS based only on one neuron net. This stage is necessary for evaluation of the system's potential limits which is driven only by one block. It seems to be helpful for building a hybrid system. Because we can add superblocks which could solve more complicated problems than just going around nearby obstacles.

Problem is to make a net that could drive robot to the given target and not to crush.

3.1. Environment for Testing Classical Neural Algorithm

As it were mentioned above, at this time robot's main control unit is the double layered neuron net (perceptron

net). It has sigmoid clipping function (1) in the first layer and threshold function (2) at the second. Such configuration is the most popular one.

$$f(x) = \frac{1}{1-\exp(-kx)} - \frac{1}{2}; \quad (1)$$

$$f(x) = \begin{cases} -x_{max}, x \le -x_{max} \\ x, -x_{max} < x < x_{max} \\ x_{max}, x \ge x_{max} \end{cases} \quad (2)$$

Inputs of the first layer are the following:

#	Name	Range	Comments
1	V	-2.. 2 ‡	Speed of robot
2	S	0 .. +∞ †	Distance to the target
3	tg(α)	-∞ .. +∞ †	Tangent α at the fig. 4 (the angle between speed and vector to the target)
4 …	Ci	0 .. 240 ‡	Value acquired from i^{th} sonar. (0 indicates that it is no obstacles within sonar's covering. 240 indicates collision). Full description of the sonars model will be given later
…	…	…	…

‡ Actually weighting factors makes absolute values of the input signals unimportant.

† Sigmoid clipping function (1) easily normalizes these infinities. It produces –1/2 for -∞ and +1/2 for +∞.

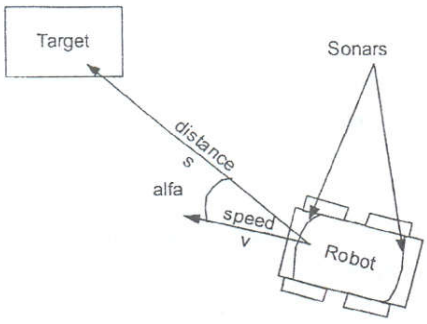


Figure 4. Mobile Robot and Neural Net Input Parameters

Sonars itself in our case are the devices for obstacle detection. They detect nearby objects in a small solid angle. Detection means that they provides a closeness value of the nearest object. There are 7 sonars in the model because it is difficult to make a model of sonar's measure error, but decreasing their number in comparison to that of robot we will reduce output measurement accuracy of whole array and solve this problem for ESS. It could be said that sonars are covering whole circle around the robot; this fact was depicted in the model we built. Absolute value of effective sonar's covering distance in model were chosen equal to 80 pix (about 25 cm for real one).

Let's get back to the neural net signals. Outputs of the net are connected to the virtual joystick of robot base. So it has two coordinates: bearing and acceleration.

#	Name and range	Comments
1	dCx: [-0.5 .. 0.5]	Value of bearing change
2	dCy: [-0.5 .. 0.5]	Acceleration of robot

Let's get back to the neural net signals. Outputs of the net are connected to the virtual joystick of robot base. It has two coordinates: so called bearing and acceleration. It is not obvious if this simple net could successfully drive the robot through a tangled route or even simply guard against collisions, but it really cans (of course with some limitations).

3.2. Net Structure

You could see whole net structure at figure 5. It is a simple perceptron neuron net. It has 30 neurons in hidden layer. Reasons for choosing such number will be described in 4.

Sigmoid functions are used in hidden layer for normalizing each of 30 components of the multiplication result between input vector and matrix of the first layer. Second matrix has two rows to result out virtual joystick's coordinates.

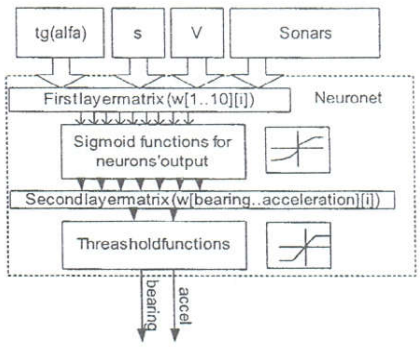


Figure 5. Neural Net Structure

3.3. Mouse Runs

Since the ESS was built it required some testing to see if it works. For this purpose we've manually inited synaptic weights of the neural net so it could drive the robot model

away from the obstacle[2]. Mouse has been representing the only obstacle in this experiment.
Thereby we saw the robot running from the mouse. For more demonstrative way to show it in a static picture it has been leaving traces (fig. 6). So the approximate sensibility of the model was adjusted.

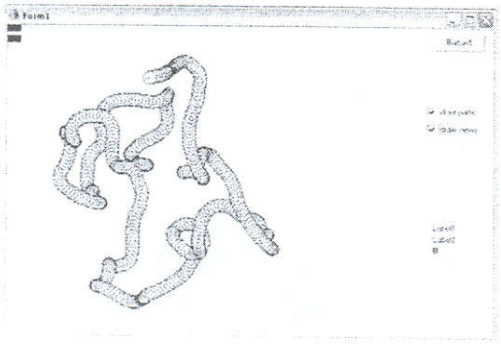


Figure 6. Performing Sonars' Model Testing

3.4. Genetic Algorithm for the Net Training

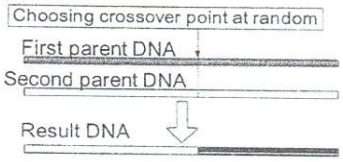
We suggested that GA is the most progressive way to solve our problem. There are many borders on the map. At this step they are represented by small circles (sonars could not know if the border has cutting edges). Touching a circle or crossing own path stops the robot (crossing in this program could be observed by the traces that robot leaves on the map). Route loop detection is important only for the training phase to create the most progressive neural net configuration. Effectiveness of the net initialization estimates (after the test's been passed) as the distance from stop point to the target.

We used simple (gaploid) Genetic Algorithm, because it's configuring net before the MCS runtime and in our case it is no need to run it for dynamical adjustment after the reasonable configuration once was reached.

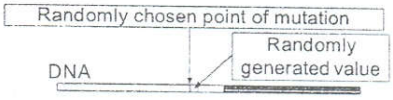
You may see diagram of designed process on fig. 8. The genom operations are represented at fig. 7 (a-c).

At first step we are creating a set of DNA's (vectors with fixed length) filled with random values – it is gene pool of our *population*. Then for every generation of this population we perform four operations: constructing[3] a neuron net by the DNA image for every robot, testing these newly constructed robot's models (one by one), calculating an error-function for every robot by it's distance to a target, reverting to treir DNA's and building new population with model of "natural selection" based on their error-functions.

a. Crossover



b. Mutatuion



c. Reversing

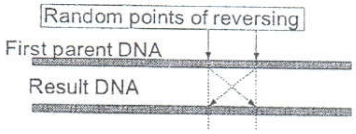


Figure 7. Main Operations for the Used Genetic Algorithm

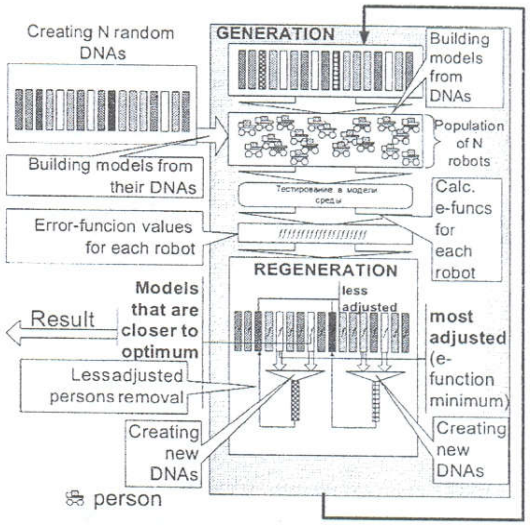


Figure 8. Used Genetic Algorithm

Let's repose on the breeding case operations depicted on Fig. 7. Crossover means that each parent gives to the Result a part of their own DNA. First parent gives first and the second gives second. The separation point is chosen randomly every time. Mutation means that one component of the Result vector is changed to a random value. Reversing means that random part of the Result DNA turns around.

GA Setup

We build the GA-modeling program for current project, so we have no need to setup the algorithm to carry out

[2] A trammel presence inside the covering zone of one sonar in this problem automatic means that robot should drive directly backward from it. Superposition of such needs is a new vector for robot's virtual joystick.

[3] Constructing method does not really mean much for the algorithm. The only constraints are that the construction could be done in two ways (each DNA corresponds to the only net and each net corresponds to the only DNA) and that the DNA components should have same limits (from 0 to 1 in our case).

results for big set of possible applications. In this case some constants were corrected in comparison with their recommended values:

- Number of persons in the population: 200
- Number of "best" persons of every generation: 10
- Potential degeneration limit: 20000 generations
- Chance of mutation (fig. 7, b) and reversing (7, c), crossover (7, a): 30%, 50%[4]
- Chance of crossover (fig. 7, a): 70%[5]

Practically algorithm drives to acceptable neural net initializations at about 400-600th generation. As it was mentioned above nets has 30 neurons in the hidden layer. We supposed that this number overcomes minimum needs of the optimal configuration. But in case of our "accelerated" GA this number looks normal because it brings bigger reserve of possibilities for faster evolution.

GA Results

To improve training process we were changing "landscape" every generation. So models had to find a new path to the target every time. (The start and the end point itselfes didn't move.) At the following screenshots start points are at the right bottom and the targets are at the upper left corners. The training has been done successful. You may see the training progress at figures 9 and 10. There are shown the tracks which leave models during their tests. At the a-picture there are only few models which could get to the target (some of them does it occasionaly). At the second picture number of well studied persons increased. There are thick traces to the target and only a few badly configured models which drives somewhere else. We suggested that the training is done when the most adjusted persons gets to the target every time and in almost every landscape.

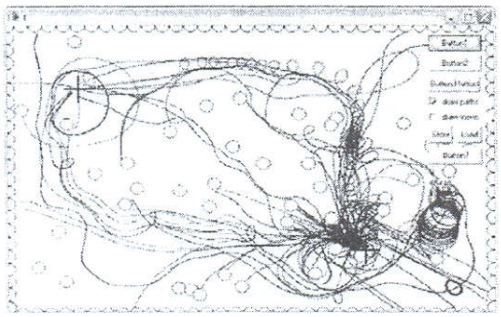


Figure 9. Population Tracks at First Generations

Result of this part of the project is that the neural net could be configured to find even a complicated way to the target. But sometimes when the path is very rough (with U-turns for example) model could end up with a kind of a dead-end. These results mean that neural MCS has problems with paths which has many turningpoints (i.e. extremums) but it is not sensible to that if the landscape is not stable. Second feature is good for the navigation in buildings where people may block the way or move furniture. But the first feature is not.

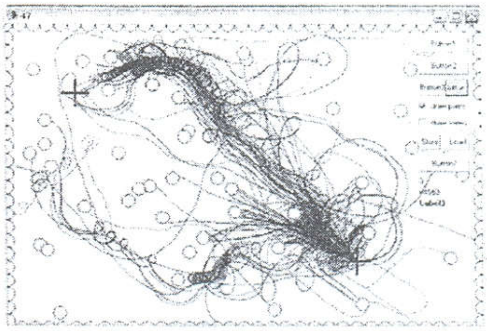


Figure 10. Population Tracks Some Generations Later

4. Creating New Algorithms for Automatic Robot Moving

Suggested system tried to overcome neural net limits. It have a system for finding and storing representative waypoints that could become local targets that splits difficult path. The problem of localization errors were also comprised by this resaerch.

4.1. Algorithm for Route Construction

We believe that the most adequate way to represent problem in common terms is to say that the neural MCS could freely drive inside the rooms and straight corridors (maybe with furniture and people). We call such places a *sectors*. The designing system should find a way based on the given by the teacher map of *sectors*. In suggested MCS this map is a route-based and it contains representative waypoints. Such waypoints could be set manually, but normaly they are placed by a special algorithm (it will be described further) to define a *sector* and *sectors'* joints. Every point has a list of it's neighbours and id of it's *sector*.

After all, we have a hybrid approach that works with neural nets at basic driving level and with route managing at the operators level.

Suggested MCS has two matrixes for calculating an optimal route. An indexing matrix contains references of the sectors' joint points in a corresponding rows and columns (these numbers equals to id's of sectors).

Second matrix contains something like a connection graph which allows to find sector sequence between the start and final target. It was called an adjacency matrix. Flowchart of the indexing process (building indexing matrix) is shown at the figure 11. Adjacency matrix building process is depicted at fig. 12 (a, b). For explicity

[4] Increased for faster results

[5] Decreased in due of retarding interpenetration of the genetic chaos.

we will describe the process in a few words in addition to the image.

In this example teacher drives four sectors. These sectors has representative points from 1 to 23. 4 is the last point of the first sector, 5 is the first for the second and so on. You can see order of indexing matrix filling at fig. 12, a. (it is built as the flowchart 11 says). Grayed cells of the matrix aren't used. At last, the 23rd point is the last for 4th sector but then the teacher drives back to 10th point (fig. 12, b). In this case matrix does not expand but the values of the newly achieved point are written to the 4th row and the 2nd column of it.

Consequently if the robot is in i^{th} sector and the target is in the j^{th} it should move to the point referenced in j^{th} column and i^{th} row.

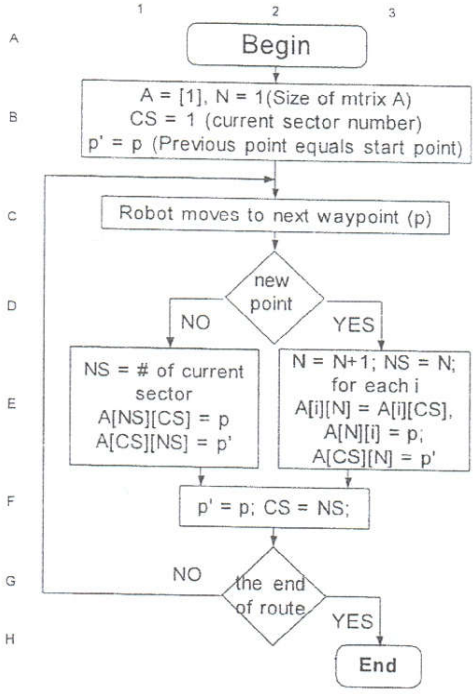


Figure 11. Indexing

5. Localization Correction

Loaclization correction is needed to compensate errors emerged between real and internal co-ordinates[6]. These errors could propagate and cause failures of MCS. Firstly we've analyzed the degree-of-freedom of the route and robot positioning alteration caused by localization errors land transformation and other factors. It lineary grows with the number of representative points. Thus MCS needs dynamic error correction to split the process between local sectors before error grows high. There are two most popular solutions for this problem – GPS and radars with beacons or a dispatcher. In our case we implemented a simple mathematic algorithm partly based on [5].

5.1. Indicative Fields

Problem was narrowed to correction of model co-ordinates to real. We do not detect causes of errors. We struggle only the fact. Actually we use two indicative fields to calculate the compensation of angular and linear errors. Main parameters of suggested algorihm are depicted at figure 13.

For the linear component indicative vector fields (VIF) calculates. Which is an extrapolation of points' vectors that are calculated from the neural net "joystick" output (the v vectors at the fig. 13).

The second algorithm was consists in allocation of the sectors' "mass centres" and calculating the rotation of VIF with "gravitation" to this mass centre of the sector. This rotation field is the second indicative vector field (RIF). We are finding maximum and minimum of that field for each sector. Thereby there are three indicative points: maximum, minimum and mass centre for each sector to derive the field's profile linear distortions.

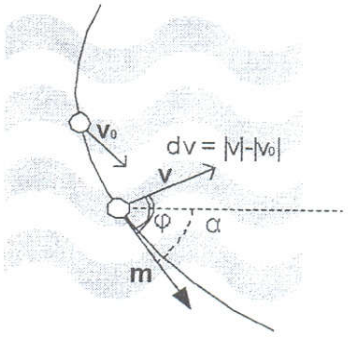


Figure 13. Data that is Used for Localization Correction for Each Representative Point

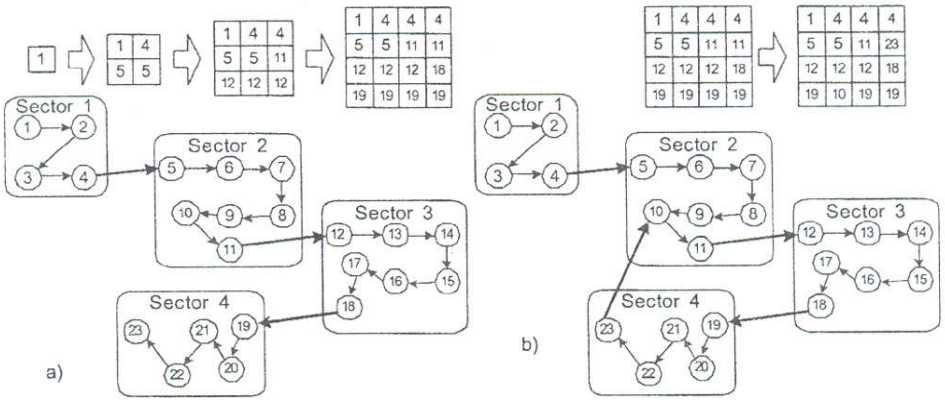


Figure 12. Indexing Example

Let's fix on the formulas. Let m(t) be a normalized vector of robot's speed and v(t) be the VIF in every route point. Then the RIF will be:

$$R(t) = |v||m|\sin(\varphi), \tag{3}$$

where the φ is the angle between v and j.

Evidently |v(t)| has no principle information of the route characteristics, but rather of the movement features. So the current procedure uses normalized v.

5.2. Calculation an Error Correction

Let's view the localiztion error in the checkpoints. This error is an (Δx;Δy) offset and it's rotation by Δα of the model system of axes to the real one.

Procedure of the localization correction by the given route's points is depicted at the figure 14. The given points always belongs to the "teacher's" path. It mean that we consider that the land is stable when the teacher is driving the robot and amount of propagated errors caused by positioning inaccuracy is small that time.

RIF of representative point in case of standard and new point are respectively:

$$\begin{aligned} r &= v\sin\varphi \\ r` &= v`\sin\varphi` \end{aligned} \tag{4}$$

Let's try the situation when the |v| is close to |v`| (this also means that Δα is small: Δα = o(φ)). (Note that if v is equal to v` φ` is equal to φ-Δα). In this case we have:

$$\begin{aligned} r` &= v`\sin(\varphi - \Delta\alpha) \\ r` &= v`(\sin(\varphi) - \cos(\varphi)\Delta\alpha) \\ \Delta\alpha &= \frac{v`\sin(\varphi) - r`}{\cos(\varphi)} \end{aligned} \tag{5}$$

Thus if we know φ of the representative point of the teacher's route then we can find Δα, the angular error. (It is important that Δα may differ from zero even when no angular error is present unlike linear one.)

We the value of Δv for every point new route. It's physical meaning is directional differential for course of m(t). So there is no way to safely predict the change of v outside of route where the robot dislocates. We can find errors only when the robot is on the known route. Based on this evident assumption we consider that correction Δx and Δy should be adjusted strictly along the route. Forward if the sign((v`-v)(v`-v0`)) < 0 and backward if sign((v`-v)(v`-v0`)) > 0. If the expression is zero there is no error.

$$\Delta x = sign((v`-v)(v`-v_0`))\cos(\alpha`+\pi)\Delta v \tag{6}$$

$$\Delta y = sign((v`-v)(v`-v_0`))\sin(\alpha`+\pi)\Delta v \tag{7}$$

There were α` in (6) and (7) but it can be also α of the teacher's route, because an angular error conducts great contradictions only with big distances. α and α` has same effect with these Δx and Δy.

Thus we have localization correction vector (Δx; Δy) and direction correction Δα.

But it is very important to mark that this algorithm of localization correction works only with small errors which are preferably propagated. This fact concerned with that normaliy the VIF is just slightly stormed on the teacher's route (because it lays normally far from the obstacles). So it has a small number of representative points.

This method would work better if the teacher's route were layed near the obstacles. Howerver this variant is not rational but it illustrates the practical intent of the theory: we can find out our real location if we will go near the familiar landscape elements.

Mentioned problem is one of the minuses of the algorithm. Second main minus is that system can not fight against great dislocation that happened suddenly.

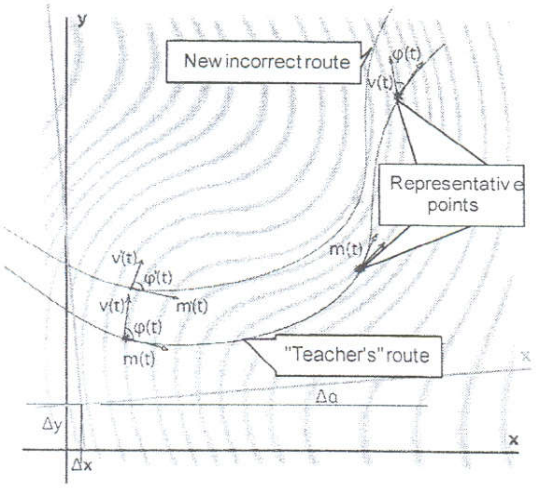


Figure 14. "Teacher's" Route and it's Representation in the Model Axis with an Error

5.3. Labels

Suggested MCS's calculations are based on landscape invariants. The next step for the system evolution is to create an additional patential by the ability of land-invariance recognition. Suggested term for the invariant land information is *label*. Generality of this information is given us by VIF and RIF that are derived from robot sensors' data. To create a real invariant of land in terms of robot route we should know some history of it's moving. Instant shot of data from robot sensors is not enough to fully describe invariant. This problem will be discussed here later.

We need some history-specific methods to find out more representative information. These methods also should have ability to be dynamicly reconstructed and to work in the real-time. We've chosen spiking (pulsed) neural nets as the working theory to solve this problem. We've done a little research in two concepts. They resulted with nothing about labelling yet, but the work is going on. Actually we are testing neo-cortex neural nets trying to decode response as if it were a hash-function of invariant route information. You can contact us by e-mail if you are interested in results.

6. Hardware Acceleration

Main advantage of the hardware acceleration in this case is that we could computate complicated problems with some parallel methods. As far as the MCS is a realtime system we suggested that it is justified to use some kind of a spiking neural nets. We've tried to build a hardware accelerator for MCS based on the CPLD evaluation board we has. This board has Xilinx VirtexE 300 FPGA, some static memory and the flash. It also has two serial interfaces which were used to connect to computer.

6.1. Simple Spiking Neural Nets

Even in the most common spiking nets implementation differs much from "classic" perceptron nets. They usually have links between every pair of neurons. There are many types of spiking neural nets (SNN) but for the first implementation we've chosen a simple structure. Spiking nets has an advantage for the hardware acceleration in case of implementation in CPLD (where the great amount of operation concurency could be reached).

SNN deals with "charges" (or "pulses") which represent processed data. Input and output data (in our case) is measured by on-duty factor of input and output neurons. The speed of pulse distribution inside net is limited. Neurons accumulate incoming pulses, generate output pulses corresponding to internal charge and discharges slowly. Our first SNN topology is given at the Fig. 15. Each neuron of this net has w (synaptic weight vector) and d (latency vector for each of it's neighbours). Net has it's model time "t" and the raster diagram of it's neurons pulses Qext (firstly filled by zeroes).

The amount of jth neuron charge change is defined as $dQ/dt = \Sigma wi \cdot Qext[t-di, i]$ when it's completely discharged ($Q = 0$) and when it has some charge it is $dQ/dt = -k + \Sigma wi \cdot Qext[t-di, i]$, where k means neuron slow discharge. When the neuron charge exceeds it's limit neuron starts to chatter with pulses. Every it's pulse imprints "1" in Qext[t, j].

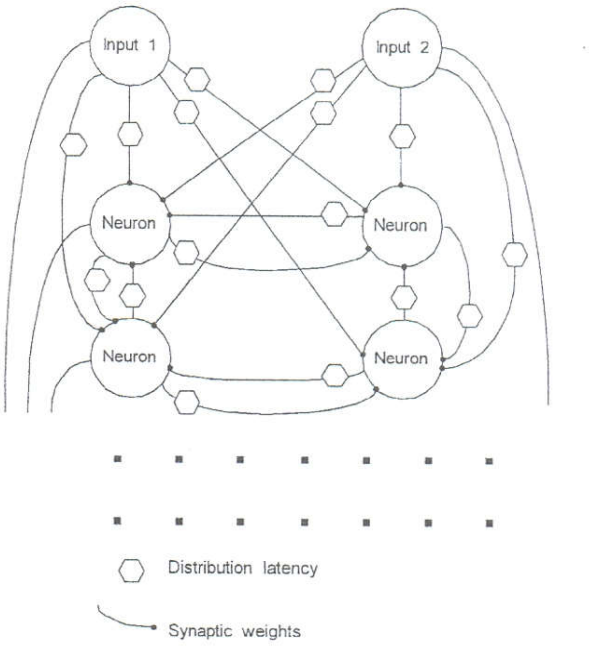


Figure 15. Used SNN Topology

6.2. Functional MCS Diagram with an Accelerator

The diagram for MCS testing with Accelerator is represented at fig. 16. It doesn't differ much from fig. 3. Main difference is that there is a tranceiver vice the neuron net on the fig. 3. Tranceiver is connected to onboard computer serial interface and transferes 10 bit data between accelerator and computer. We used CORE Generator to build the interface on the accelerator side.

6.3. Results for the First SNN Configuration

First SNN has a delay line matrix 4x4 with 4 fully interconnected neurons to simulate pulse distribution latency properly. All constants are generated by special schematic symbols. The research indicates that used configuration is not capable enough to replace software neural net.

We haven't stop there and started to develop another concept for SNN found in the article "Spike-timing Dynamics of Neuronal Groups" [1] and attended works of E. Izhkevich [2-3]. Also we've used matherial from some topical books like [4].

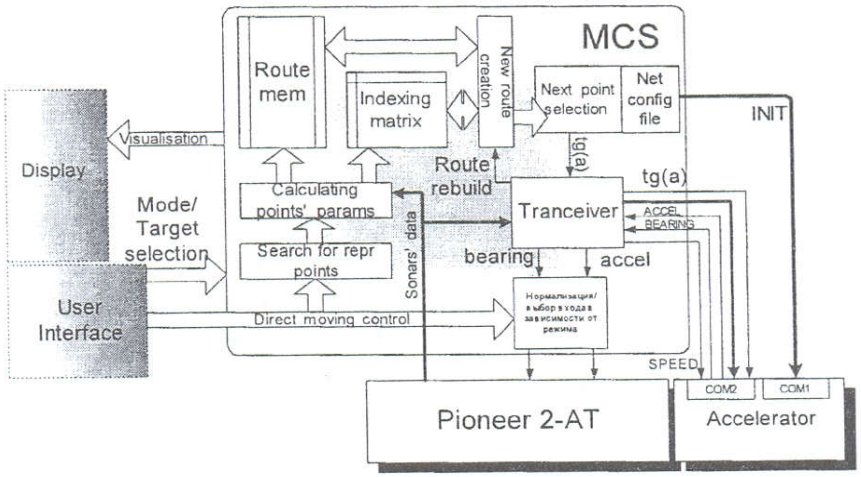


Figure 16. Used SNN Topology

So we have ended up with a variety of very interesting results but none of them could help us to improve the MCS yet.

7. Present Results

We've designed an Adaptive Moving Control System for the mobile robot we have. It is not implemented completely yet, but it's passed main tests. The hybrid approach justified itself in terms of work simplification. Since it is used the problem could be separated in a few subproblems that lives only inside their "tessituras". For example: there is no need to complicate methods that deals with "high level" waypoints if the local problems could be solved by another method. It is much simplier to separate methods' work hierarchical. And this article tries to demonstrate how.

Acknowledgments

This investigation is partially supported by the following grant: № 06-07-89146-a of Russian Foundation for Basic Research (RFBR).

References

1. Izhikevich E.M., Gally J.A., Edelman G.M. "Spike-timing Dynamics of Neuronal Groups". The Neurosciences Institute, San Diego, CA, USA.
2. Izhikevich E.M. "Simple model of spiking neurons". *IEEE Trans Neural Networks*, 2003; 14.
3. Izhikevich E.M., Desai N.S. "Relating STDP to BCM". *Neural Computers*, 2003.
4. Braitenberg V., Schuz A. "Anatomy of the cortex: statistics and geometry". Springer Verlag, Berlin, Germany, 1991.
5. "Robust Monte Carlo Localization for Mobile Robots". *Artificial Intelligence*, 2001.
6. Borenstein J., Everett B., and Feng L. "Navigating Mobile Robots: Systems and Techniques". A.K. Peters Ltd., Wellesley, MA, 1996.