Computer Software Technologies for Intelligent Robot

VLADIMIR PAVLOVSKY
KIAM RAS
Department of Mechatronics
Miusskaya sq., 4 Moscow
RUSSIA
vlpavl@keldysh.ru

ANTON ALISEYCHIK
KIAM RAS
Department of Mechatronics
Miusskaya sq., 4 Moscow
RUSSIA
aliseychik@keldysh.ru

IGOR ORLOV
KIAM RAS
Department of Mechatronics
Miusskaya sq., 4 Moscow
RUSSIA
i.orlov@keldysh.ru

Abstract: Nowadays new computer techniques for intelligent robots are the important point of investigations. Technologies for the intelligent manipulators capable to make decisions automatically is convenient to develop using laboratory tasks, like games with the objects manipulation. As examples of such tasks a) the game “Gomoku” and b) the game “Go” are considered. A set of problem-oriented technologies were investigated from the point of view of the manipulator ManGo robot, namely its dynamic model and control system in Matlab Simulink. The logical move choice program in the game “Gomoku” is designed in Prolog and C languages. Vision system as supporting technology will be demonstrated as important part of the presentation. It is important to note, different algorithms can be implemented as manipulator software and hardware technologies. Besides classical algorithms utilization, considerable attention has been given to the application of neural network paradigms for manipulators control. Addressing the problem the neural software for ManGo robot will be presented as well. The created robots passed experimental working off which allowed to draw conclusions about the created software and hardware effectiveness in the implementation of various manipulators control algorithms.

Key–Words: Robotics, computer vision, neural networks

1 Introduction

Nowadays one of the most important tasks in robotics is creating intellectual robots. Among them creation of intellectual manipulator robots, able to automatically make decisions during their work, is being widely explored. It is convenient to research such systems on lab tasks such as different games, requiring to manipulate different objects. In this work, hardware and software means of intellectual manipulator robots realization are considered, besides, two tasks and systems solving them are described. The first one is ManGo robot, developed for solving some intellectual gaming tasks for robots, such as logical table games. Initially ManGo was created to play “Go” game with a human, but as the first and easier application “Tic-Tac-Toe” game on a big, potentially unlimited field, is being considered. On this game, controlling the manipulator is being developed.

2 Logic Games Playing Robot

In an ancient Chinese game with a Japanese name “Gomoku” also known as Tic-Tac-Toe or “five-in-a-row”, two player place symbols on an infinite desk, or a desk of $m \times n$ size by turns , until any of them places five of his symbols in a row vertically, horizontally, diagonally, or until the desk is fully covered by symbols, if it is finite. In this case, neither of the players win. In case of infinite desk the number of turns is limited. Each player uses symbols of only one type: either “cross”, or “zero”. Each field of the desk can contain only one symbol. Instead of “crosses” and “zeroes” in the original version stones of different colors (white and black) are used. In “Gomoku” game players place symbols on line intersections instead of fields. The desk size is usually limited by the size of cross-section paper sheet. For “Gomoku” game 15×15 or 19×19 (like in “Go” game) desk size is used. In the robot turn realizing program 20×20 desk is used. It is easy to prove that if the first player plays well enough, he can play draw at least. It is harder to prove that he can win but it is still obvious. The first player has an advantage, for this reason he is given a handicap, restricting some moves. Particularly it happens in “Renju” — one of the analogues of “Gomoku”. It is known that professional “Renju” players, who move first gain an advantage at 10th and

ISBN: 978-1-61804-344-3

win the game "Gomoku" at 15^{th} move. [8] Presents a program Victoria that always wins the game on desk 15×15 if it moves first.

In computer applications, realizing the game "Five in a Row" the move of computer player is often made with the help of estimation function. For each empty field with indexes (i, j) its estimation (a real number) is calculated, considering both profit of the player, and the profit of an opponent that he could gain after making this move on field.

As the estimation function, for instance, the function like $ev(i, j) = ev_x(i, j) + aev_0(i, j)$ Is used, where $ev(i, j)$ — is an estimation of the players move in this field, and $ev_0(i, j)$ — is an estimation of his opponents move. Coefficient a is inverted ratio to aggressiveness of the player. With its high value the strategy of the player has a defensive character and with its low value — offensive. The program for the robot is written in Prolog language. The steps of a computer player in games, written in Prolog are often realized with production rules. In the system both rules and estimating function are used.

At the input of the program the array, consisting of numbers $0, 1, 2$, that stand for empty and occupied with stones of dark and light colors respectively. The program returns the number of a row and the number of a column of a field that robot moves to.

The application in Visual Prolog 7.4, one of the most developed realizations of Prolog language was created for debugging the strategy of the computer player. A fast complier enables to ignore the details of algorithm and to formulate rules by declarative way, so it increases the speed of coding the program. The user can choose the player that makes the first move. If it is a computer, the window is opened with a move already made. The first step is made randomly on a field that is less than 3 rows away from the center of the board. The users move is made by a click of a mouse on the field.

The players move rules are represented in the following way:

- to put own sign fifth in a row;
- not to allow the opponent to put his sign fifth in a row (to perform "block");
- not to allow the opponent to put his fourth sign in a row;
- to put own sign fourth in a row;
- to create a "fork";
- not to allow the enemy to create a "fork";
- to put a sign in the field that is located near the most of opponents three-sign rows and at least one empty field;
- to put a sign in the field, the estimation of which has the highest value.

Among the fields with the maximum value of the estimation function the fields that border with the maximum number of cells, occupied by the opponent are collected. The field among the collected ones is chosen at random. For the move search the estimation function, described above is used. As the row estimation the function $ev(s, k) = 4^{k+1}$ is taken, where s is a sign, k — number of signs s in a row, parameter a is assumed to be equal to 1. For position storage, the program uses lists; in order to lower the searching, only occupied slots are stored. List in Prolog language is a persistent data type, it allows backtrack. Unlike it in modifiable data types, in arrays of *C* language for instance, the changes are made in computer memory, so backtrack is unable for them. The usage of lists allows the simplest situation modelling, for instance, checking if the critical situation exists after placing on one field own and the opponents sign.

3 Manipulator and Control System

Mango manipulator (Fig. 1) has a SCARA-like kinematic, that mostly suits object manipulation tasks on a plane, including desk games [6]. The first steps in cinematic analysis were carried out during the creation of robots design in CAR soft complex, the pneumatics of Italian company "Pneumax" was used as the executing motor. Optimal lengths of parts and attachment points of pneumatics were calculated to cover workspace of 500 x 500 cm size, which is enough to work with almost every knowledge-based logical desk game.

Due to simple two-section cinematic scheme the solution of inverse cinematic problem for robot control is trivial. Therefore, the most reliable in this case trajectory cinematic control with inverse link in the angles of joint rotation. The usage of pneumatics involves both ad-vantages (cheapness, higher speeds of motion, great efforts, etc.) and drawbacks, the main of which is the difficulty of accurate control. In order to solve this problem the riveted PWM-control of the cylinders was realized. The control system of the manipulator ManGo for

Figure 1: ManGo robot

the “Gomoky” game was realized on the pair microcontroller Stm32F4 – PC with Windows OS, and the control system – on the pair microcontroller Stm32F4 – smartphone with Android OS.

3.1 Machine vision

Desk recognition on a picture can be divided into several steps:

- Picture preparation (converting into binary matrix)
- Searching for intersections and edges of the board
- Searching for neighbors and beginning of the grid building
- Completing the grid

The main steps of desk recognition algorithm on a certain example is described further.

3.1.1 Pattern Recognition

Since lines on the board are thin and therefore poorly discernible (especially because of desk bent relatively to camera, natural non monotonous wood texture of the board surface and occasional glints), high quality of the picture is required. Using of camera with resolution of no less than 3 megapixels is recommended.

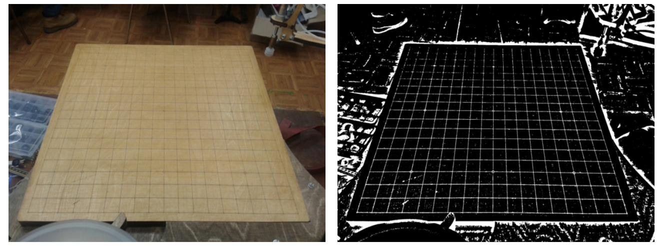

Figure 2: The original picture (on the left), picture after adaptive threshold (on the right)

To get a binary picture with sharp outlines, without noise, Adaptive Threshold [7]–[10] is applied to the original picture(Fig. 2). Operations of dilatation and erosion follow next to smooth and clean the image.

3.1.2 Searching for Line Intersections

Now it is time to get to the next step, which is patterns usage. Before anything else, the edges of the desk must be found. It is very important to find the bottom edge (the top is hard to find due to desk bent, it is often the same color as the surroundings), and at least one of the side edges, to define the position of the desk on the picture. Patterns describe how line intersections should look like in binary picture. All that is need to be done is to apply these patterns during hit-or-miss transform [16]. However, because of image flaws (noise and discontinuous lines), some edge points are missed and some detected points don’t belong to the edge. Next step is to detect an line. OpenCV library offers methods of fitting line in array of points, but their accuracy is not sufficient. The only option is to run through all nearest to each other pairs of points, chose those with more probable angle (since possible range is known for each edge). Associated with pair of points line is added to vector with one “vote” or, if a line with similar parameters has already been added, count of votes of latter increments. Line with biggest count is the one.

Intersections inside the grid are found in the same way, through patterns.

3.1.3 Grid Building

FLANN (Fast Library for Approximate Nearest Neighbors) [11] is a library, that implement nearest neighbor search method (interface for the library is contained in OpenCV). It uses point array in matrix form as input, and the result is k-dimensional tree, data structure, dividing the space for sorting points in k-dimensional space. In this case, the space is obviously two-dimensional. Each point of the desk has no more than four direct neighbors. In process of searching of four nearest for each point, restrictions are placed according to situation:

- Little distance between points, the distances between the point and each one of the neighbors shouldn't differ too much from each other;
- Right and left neighbors must lie on the same horizontal line;
- Top and bottom neighbors must lie on the same line, which is bent from the main vertical to no more than 30 degrees.

Point becomes a part of the grid if it has no less than two neighbors.

During this part of the algorithm, vector of 6-slot arrays is created, each array associated with the single point. The first four slots contain neighbor indexes (or -1, if there is no neighbor in a certain direction), and the other two contain horizontal and vertical line numbers, to which the point belongs. Fig. 3 shows how the points and their relations were found. The points on the top werent found either because the distances between them were too small or detected points in the area were to scarce to find any relations. Next picture shows the found lines as well. Next step is completing the grid

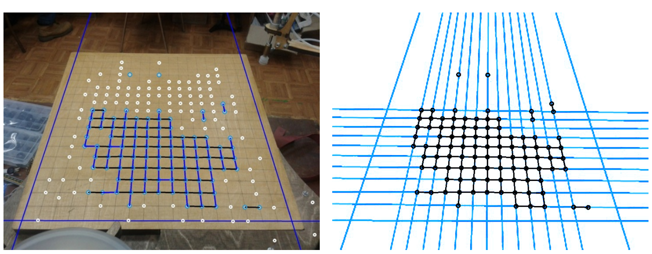

Figure 3: Lines and relations found

to a bounding rectangle. It can be done by calculating intersections of found lines, if there is any. Thats why further actions take place:

1. If the point doesnt have one neighbor, it is calculated due to the coordinates of the point and the opposite neighbor. Afterwards, the new point is added to a vector of new possible points, and the counter of current is assigned 1. 2. Before adding the coordinate to the vector, it's important to check whether there is a point with close coordinates. If so, the counter increments, and the coordinates are changed to the average between the old and the new one, if not, a new point is added. 3. In the end, the point is added to the grid, if the counter value is at least two.

The process continues until there is no more points to add. As a result is supposed to be a matrix N (width) x M (height) size.

3.1.4 Finishing the Grid

As can be seen on Fig. 4, points lies on intersections of found lines and edges.

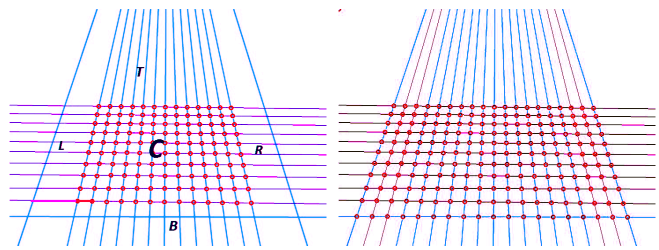

Figure 4: Intersection points are found

Number of missing points between the edges and rectangle can be calculated due to the grid that makes it possible to find average distance between points on each horizontal line. This way grid is expanded to zones L, R and B (Fig. 4, left).

To find the rest of the points the following actions are performed: 1. Diagonals, and their intersections with vertical lines are found, as shown on Fig. 5, left. 2. Now we can find the rest of horizontal lines. 3. Find the rest of the points as intersections of vertical and horizontal lines. Point search completed (Fig. 5, right).

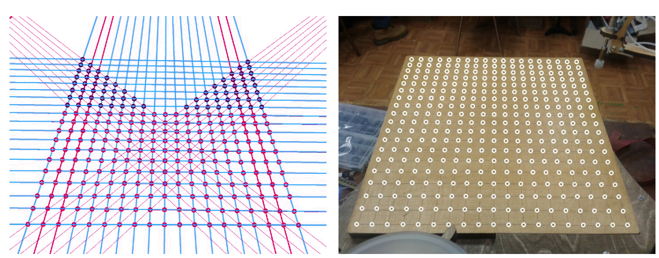

Figure 5: Looking for diagonals, horizontal lines, intersections of diagonals and horizontal lines, points found on the desk (respectively)

ISBN: 978-1-61804-344-3

3.2 Black and White Stones Recognition

The idea of the algorithm is based upon the fact that illumination doesnt change during the game. The initial picture of empty desk is saved. The new picture (with stones on the desk) and the old one (without) are converted to HSV (Hue, Saturation, Value) format. Once the matrix of desk points on the picture is already found, the average value of brightness (Value), and intensity (Saturation) in the point surroundings are compared between two images. The less the Saturation is, the more “grayer” is the color. It means that black and white (stone colors) will have the lowest Saturation value. The part of the image, where the white stones are, will have the higher Value, and the black stones will have lower Value. To remove the flecks off the stones, the picture is corrected by Gaussian blur.

3.3 User Interface

An application for Android is a user interface, where it is possible to set rules of the game and start it. Once play has began, it automatically tries to connect to robot via Bluetooth, after succeeding, if you choose to start a game, it enables you to “calibrate” the desk, recognize it with internal camera of the device. The internal tools of the application include GNU Go engine [12], desk and stones recognition algorithms, and Bluetooth connection between ManGo robot and the device. GNU Go is a free Go playing program. It doesnt have a graphical interface, but it supports two protocols to “communicate” To other programs: Go Modem Protocol and Go Text Protocol (GTP). In this application GTP is used. The program is realized on C++ programming language. Image processing is realized on C++ language with usage of OpenCV (Open Computer Vision) library. The application interface, Bluetooth connection and camera treatment is realized on Java, with usage of Android features for developers (Android SDK), which enables us to control Android API.

4 Neural Network based Control

All functions of manipulator control can be realized by means of neural network models. But before replacing control elements it is worth considering advantages and shortcomings of neural network models in contrast with traditional approaches.

The main convenience of neural network models is the capability of self-organization. Primary idea is just to remember some basic interrelations between input and output vector signals $X_i \to Y_i$ and make some kind of interpolations of Y for $X \neq X_i \in \{X\}$ Neural network shouldnt remember all input signals, but the set of remembered $\{X_i\}$ ought to represent correspondence $\{X\} \to \{Y\}$. The conventional structure of formal neuron ”j”: X, Y, M_j and L_j are vectors. X and Y are representing activity of input and output neurons and M_j and L_j are weights of input and output connections. The formal neuron ”j” activity a_j is calculated like some monotone increasing function of $X * M_j$ dot product. Model of formal neuron can be considered as a memory cell. Weights of connections are responsible for data storage. Input weights M_j are the ”address” of the cell. When X is close enough to M_j to make $a_j = f(X * M_j) > 0$, neuron ”j” plays back the data, stored by weights L_j. The contribution of neuron ”j” to output layer activity Y is proportional to a_j. As a rule several neurons a_j can add up to output neurons activity vector Y. All formal neurons and connections are consimilar. But functions of layers X, A and Y are different. Layer X represents input signal, A is mapping the input signal space of states and Y should represent Y in correspondence $\{X\} \to \{Y\}$. Also the rules for M and L connections weights learning must be dissimilar. Interconnected layers pairs XA and AY only look alike. Pair XA has to remember the states of input layer X, while pair AY must memorize desirable states of output layer Y. Neuron ”j” from layer A should remember the mean activity X while $a_j > 0$ and neuron ”k” from layer Y — vary weights L_k to get suitable activity y_k for different X_i (and corresponding A_i). Learning rules for vectors M_j and L_k should be like

$$\Delta M_j = \eta_1 (X - M_j) a_j \Delta t \tag{1}$$

$$\Delta L_k = \eta_2 (y_k^s - y_k) A \Delta t \tag{2}$$

where y_k^s — specified activity of y_k and η_1, η_2 are coefficients, and $\lim_{t\to\infty} \eta = 0$, $\lim_{t\to\infty} \sum \eta = \infty$.

Both learning rules are designed to minimize the distinction in parentheses, but the directions of vectors M_j and L_k changes (with reference to vectors X_i and A_i) are quite different. The task of converting $\{X\} \to \{Y\}$ allows to choose arbitrary activity level for hidden layer A. Both learning rules (1) and (2) are changing weights only for connections with active neurons of layer A. The more layer A neurons are active simultaneously, the less resolution of the input signal space of states we can get (in the sense of difference between vectors M_j). So it is better to limit the number of active layer A neurons by few units.

ISBN: 978-1-61804-344-3

The question about remembering output signals Y is not so obvious. If there are N neurons in layer A and the activity limit is 1 neuron, there are N different states of layer A activity. If limit is 2 there are $N(N-1)/2$ different states of activity, if 3 there are $N(N-1)(N-2)/6$ states and so on. One can think, that high activity is better. But if we want to memorize more than one independent signal Y, we should be able to solve the system of p equations (p pairs $X_i \to Y_i$ with corresponding inner layer activity $Ai = \{a_{1j}^{Ti}\}$):

$$\sum_{j=1}^{N} l_{jk}a_j^1 = y_k^1, \sum_{j=1}^{N} l_{jk}a_j^p = y_k^p. \tag{3}$$

The system is solvable for $p \ \leq N$ and activity limit doesnt change the number of independent output signals Y, that can be memorized on N elements of hidden layer A. It means that there is no reason for choosing high activity level. Self-organizing maps (SOM) elaborated by T. Kohonen [13], [14] can solve the mapping task for the input signal space of states. Originally SOM answer is the activity of only one neuron of inner layer, "winner takes all (WTA)". For the Y interpolation purposes better to have several active neurons. SOM with WTA can be easily expanded to SOM with activity center (AC). It is worth mentioning, that SOM learning rule is exactly (1). Learning rule (2) is used in backpropagation paradigm [15], [16]. Backpropagation technic was successfully implemented for solving different tasks. But the tendency to use purely rule (2) (without rule (1)) resulted in various problems while learning and in some measure useful attempts to solve these problems by means of randomization, normalization, orthogonalization, using multiple layers (deep learning) and so on. Both rules (1) and (2) are used in counterpropagation network [17]. It was developed in 1986 by R. Hecht-Nielsen. It is guaranteed to find the correct weights, unlike regular back propagation networks that can become trapped in local minimums during training. Amazingly counterpropagation is less popular, then backpropagation model. Perhaps its easier to implement pure rule (2), then think about cooperation of rules (1) and (2). Also, original counterpropagation use WTA competitive network (like original SOM), while for purposes of smooth converting $\{X\} \to \{Y\}$ its better to apply CA. Different counterpropagation network extensions were used for manipulation tasks control. The general idea was to convert some data about actual and specified manipulator states X into control signal Y, as shown on Fig. 6. Two-link manipulator arm is moving in horizontal plane. Two pneumocylinders operate the arm movements. Each pneumocylinder position is controlled by sending pulse-width modulation (PWM) to valves V_3 and V_4 (Fig. 7). Valve 4 should be opened to move the piston rod to the right. If valves V_1 and V_2 are switched to the opposite position, short PWM impulses to V_3 will cause the slow motion of piston rod leftwards.

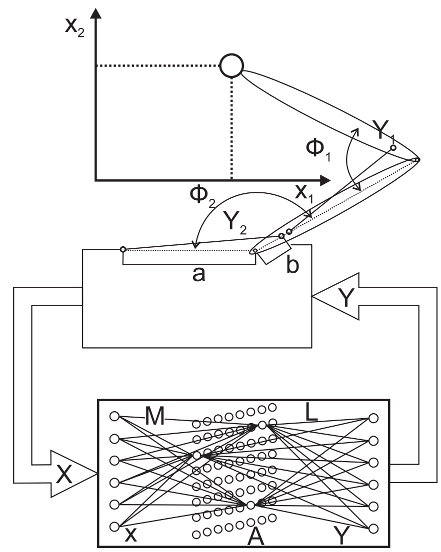

Figure 6: Neural network manipulator control

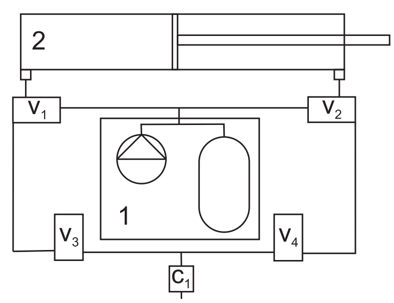

Figure 7: Pneumocontrol structure. 1 — air pressure source; 2 — pneumocylinder ; V_1 – V_4 —electrically controlled valves; C_1 — air flow constrictor

Unpretentiousness of manipulator structure, tasks and control functions was very helpful for detail study and main ideas understanding of neural network manipulator control. It gave possibility to start the work from very simple tasks for neural net and then gradually increase the complexity of the tasks.

First the quasistatic tasks were examined. The simplest task is to control only one manipulator arm link and to be able to stop at specified ϕ^s starting from ar-

ISBN: 978-1-61804-344-3

bitrary ϕ. This task can be easily solved by the instrumentality of standard calculations. The ϕ_2 angle is controlled by y_2 position. The traditional control formula for Δy_2 to reach specified ϕ_2^s starting from arbitrary ϕ_2 is (see Fig. 6 for designations):

$$\Delta y_2 = \sqrt{a^2+b^2-2ab\cos\phi_2^s} - \sqrt{a^2+b^2-2ab\cos\phi_2} \quad (4)$$

And Δy_2 is proportional to valves V_3 or V_4 opening time, $T = k\Delta y_2$, which can be transformed to PWM parameters (frequency, width and number of pulses and valve number (V_3 or V_4)). In experiments the mean mismatch of single movement was not very high, with minimum about 5% of $\Delta\phi_2$ at $\phi_2^s = \pi/2$. But the displacement could be repeated from closer position and $2--3$ movements usually allowed to reach desirable error of link end position less than 2 mm. Can the results of task solution be improved by means of neural net implementation? Equation (4) gives ideal solution for ideal arm link control model. Real arm link and its control can slightly differ from ideal model. During traditional control installation and tuning only a, b, ϕ and multipliers of transformation $\Delta\phi_2$ to PWM parameters could be adjusted, while the function (1) and direct proportionality $T = k\Delta y_2$ to PWM parameters are unalterable. Neural nets are representing transformation function in tabular form and are capable for functions form fine tuning during adjustments to real control conditions.

Only one net unit was used in the first experiment. Two angles of the arm link position (actual $\pi/2$ and specified ϕ_2^s) were forming X for neural net and Y was composed of four PWM parameters. The initial values for vectors M_j and L_k were calculated by (4) and law of transformation Δy_2 to PWM parameters. 1000 input signals X (formed of random ϕ_2^s and previous ϕ_2) and learning rules (1) and (2) application resulted in improvement of minimum mean mismatch to 3,6% for 80 neurons of inner layer. 40 neurons net reached 4,7% after 350 steps of learning. 20 neurons started from 9% of minimum mean mismatch and reached reduction to only 7,6%. So 20 neurons arent enough for good tabular representation of function (4).

More than 80 neurons of inner layer make the learning time longer (proportionally to N^2), but do not lead to sufficient mismatch decrease. 200 and 400 neurons make minimum mean mismatch equal to 3,4% and 3,3% respectively. Its meaning, that 3,3% mismatch is based on unpredicted random errors. Second experiment was based on dividing transformation $\{X\} \to \{Y\}$ into three stages. On the first stage nonlinear transfers actual ϕ_2 and specified ϕ_2^s to y_2 and y_2^s was performed separately on the same net with 15 neurons of inner layer. On the second stage $\Delta y_2 = y_2^s - y_2$ and $abs(\Delta y_2)$ were calculated. And on the third stage Δy_2 was converted into four PWM parameters (7 neurons for frequency, 10 for width, 10 for number of pulses and 4 for valve number (V_3 or V_4). This configuration demonstrated minimum mean mismatch 3,5%. The advantage of three stage approach is that all of three stages are simpler, than conversion in one stage. The single transformation was two-dimensional and start working good from 80 inner layers neurons, while three stages transformations are one-dimensional and need less neurons. The second stage (of three stages) was not realized on neural net, but for these linear transformations 10 neurons are more than enough. The main benefit of neurons number reducing is acceleration of learning process. For three stage approach minimum mean mismatch 3,5% was reached after processing of 70 input signals. For small simple tasks the difference between one and several stages transformation isnt sufficient. But for more difficult tasks with dimensionality more than 10 good tabular representation of functions becomes impractical. Too many neurons are needed and the learning time grows unreasonably. The way of solving this problem is to split the complex task into several simpler tasks with dimensionality less than 6–7, 2–3 is desirable and 1 is the best. After solving the single link quasistatic control task its time to consider two-link quasistatic control task. The positions of links are defined by angles and, since the dynamic of the process is not analysed, can be controlled independently. The only difference with training single link quasistatic control task is that the goals arent angles ϕ_1^s, ϕ_2^s, but coordinates x_1^s, x_2^s. The task of conversion coordinates to angles can be easily solved geometrically, but also it can be solved by means of neural net. Anyway specified coordinates x_1^s, x_2^s should be transferred to ϕ_1^s, ϕ_2^s. Angles ϕ_1 and ϕ_2 are measured by angle sensors. So pairs ϕ_1^s, ϕ_1 and ϕ_2^s, ϕ_2 can be transformed to operating pneumocylinders positions PWM parameters. The traditional control formula (4) or one of two neural nets approaches can be used for fulfilling the conversions. Mismatches of all three methods for two-link arm are proportional to results for one link with multiplier $1,41$. Learning times for neural nets approaches are approximately four times longer. The best results for two-link arm also showed the three stage transformations method. Minimum mean mismatch 4,8% of ΔX was reached after processing of 300 input signals. 1–2 movements were necessary to reach

ISBN: 978-1-61804-344-3

desirable less than 2 mm error of two-link arm end position in the center of desk and 2–3 movements at the desk edge. There are some important questions left to investigate in the quasistatic manipulator control task: balancing of learning rules coefficients, emphasizing of sufficient variables, quick tuning without changing the shape of nonlinear transformations and others. The next step is considering dynamic manipulator control tasks in order to reach the specified points by one movement, without additional correcting movements. The corrections must be made during the movement. These tasks have higher complexity, because except main variables their time derivatives should be taken in account. Also, in contrast to quasistatic control, high transformations performance is needed to have time to make several corrections during quick movement. But any solutions of dynamic manipulator control tasks by neural net means will greatly extend the manipulators application area.

5 Conclusion

The performed experiments have proven the effectiveness of software and hardware tools of intelligent robotics and their correspondence to the tasks. Detailed development of these instruments will provide shorter (faster) worktime of the robots, and improved logical features of the robots. Future plans are developing fully "assembled" intelligent robot-manipulator with planning and playing games abilities based on visual system and neural "nervous" system. The computer technologies that have been developed will be the universal tools for this activity.

Acknowledgements: The research was supported by the Keldysh Institute of Applied Mathematics and in the case of the second author, it was also supported by the Grant Agency of RFBR (grant No. 15-08-08769).

References:

[1] M. Ghallab, D. Nau, P. Traverso, Automated Planning: Theory and Practice. *Morgan Kaufmann Publishers.* San Francisco. 2004.

[2] S. M. LaValle, Planning Algorithms. *Cambridge University Press.* 2006.

[3] Q. Yang, Intelligent Planning. *A Decomposition and Abstraction Based Approach.* Springer Verlag. 1997.

[4] D. Y. Pogorelov, On Numerical Methods of Modelling Large Multibody Systems. *Mechanism and Machine Theory.* 34. pp. 791-800. 1999.

[5] L. V. Allis, H. J. van den Herik, M. P. H. Huntjens, GoMoku and Threat-Space Search. *CS-report,* Univ. of Limburg. 1992.

[6] I. A. Orlov, Synthesis of Motions for Manipulation Systems for Spaces With Complex Relationships and Constraints. *PhD dissertation.* Keldysh Institute of Applied Mathematics, Moscow. 2013. [In Russian]

[7] G. Borgefors, Distance transformations in digital images. Comput. Vision Graph. *Image Process.* 34 (3). pp. 344371. 1986.

[8] P. F. Felzenszwalb, D. P. Huttenlocher, Distance Transforms of Sampled Functions. *Theory of Computing.* 8 (1). pp. 415-428. 2012.

[9] F. Meyer, Color Image Segmentation. *International Conference on Image Processing and its Applications.* pp. 303-306. 1992.

[10] A. Telea, An Image Inpainting Technique Based on the Fast Marching Method. *Journal of Graphics,* GPU, and Game Tools. 9 (1). pp. 23-34. 2004.

[11] M. Muja, D. G. Lowe, Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration. *VISAPP* (1). pp. 331-340. INSTICC Press. 2009.

[12] Free Software Foundation, http://www.gnu.org/

[13] T. Kohonen, Self-organized formation of topologically correct feature maps. *Biological Cybernetics.* 43. pp. 59-69. 1982.

[14] T. Kohonen, Self-Organizing Maps (Third Edition). New York. 2001.

[15] D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning representations by back-propagating errors. *Nature 323 (6088).* pp. 533536. October 8. 1986.

[16] G. Hinton, L. Deng, D. Yu, G. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, B. Kingsbury, Deep Neural Networks for Acoustic Modeling in Speech Recognition — The shared views of four research groups. *IEEE Signal Processing Magazine.* vol. 29. no. 6. pp. 82-97. 2012.

[17] R. Hecht-Nielsen, Counterpropagation networks. *Applied Optics.* 26. pp. 4979-4984. 1987.

[18] Hit-or-miss transform in OpenCV, http://opencv-code.com/tutorials/hit-or-miss-transform-in-opencv/