VILNIUS UNIVERSITY

Vytautas Valaitis

NEURAL NETWORKS BASED ROBOT MOTION IMPROVEMENT

Doctoral Dissertation
Physical Sciences, Informatics (09P)

Vilnius, 2016

The dissertation research was carried out at Vilnius University from 2011 to 2015.

Scientific supervisor:

Prof. Dr. Habil. Šarūnas Raudys (Vilnius University, Physical Sciences, Informatics - 09P)

VILNIAUS UNIVERSITETAS

Vytautas Valaitis

ROBOTO JUDESIŲ GERINIMAS NEURONINIAIS TINKLAIS

Daktaro disertacija
Fiziniai mokslai, informatika (09P)

Vilnius, 2016

Disertacija rengta 2011 - 2015 metais Vilniaus universitete.

Mokslinis vadovas:

prof. habil. dr. Šarūnas Raudys (Vilniaus universitetas, fiziniai mokslai, informatika - 09P)

Table of Contents

Introduction

The real world environment constantly changes. Robotic systems also change: parts wears and gets damaged, software might have calibration issues. Systems must be capable to adapt to these changes. While the classic analytic methods are insufficient to cope with this task, new learning methods are more capable. It is mainly due to the fact, if robotic systems are slightly changed, new analytic models must be developed. Sometimes it can be as trivial as changing a few parameters but mostly non linear deviations of unknown cause appear in the system. These deviations can be compensated using learning methods, without clarifying their cause. Multi-layer perceptrons are widely used in robotics, but perceptrons abilities to adapt to sudden changes are mostly overlooked.

Motion consists of goal position finding, trajectory planning, and motion execution. Goal position finding and inverse kinematics of robotic systems are analyzed in changing task environment. Single layer and multi-layer perceptrons ability to adapt is analyzed. Different techniques of tuning single layer perceptron and multi-layer perceptron alter system capability to adapt, and to cope with sudden or gradual changes in the environment, or the system itself.

Natural motions can be classified to reflexes, partly coordinated and fully coordinated motions. All these types of motions require different techniques to be accomplished. Inverse kinematics solution plays a key role in fully coordinated motions. In some cases (like fully coordinated motions) accurate inverse kinematic solutions must be obtained. Computational speed can be sacrificed in benefit of accuracy. On the other hand, reflexes require little accuracy, but high computational speed. Partly coordinated motions benefits from predefined motion primitives.

Feedback is used in robotic systems to get extra information about its components (e.g. actuators), position, and parameters of environment. For example robot ability to travers different terrain can be considered as feedback [44]. Further research of the author uses real time energy consumption as a feedback system. More traditional way is to get a feedback from end actuator position. This is analyzed in Chapter 3. Piezoelectric force feedback sensors are introduced to aid partly coordinated motions' calculation. Feedback system is often needed to perform a less energy consuming but a more precise motion.

Trajectory planning based on primitives is analyzed in the thesis. Motion primitives is a nature based way to plan motion. They are used in state of the art robotic systems. A new method of learning correct primitives is offered in this thesis.

Multi-agent systems were used to model agent's behavior in a changing task environment. The performed simulation allowed to alter the task solved by agents and monitors population performance. The software used was written in Python to work with multiprocessor machines, as well as, distributed networks (using MPI). Multi agent systems and parallel computing are both nature-inspired practices used in modern modeling.

To execute robotic motion a number of things must be taken into consideration. Firstly, the desired position to reach must be identified. Then inverse kinematic calculation must be performed. This calculation can be precise or approximate, based on the planned motion type. Robotic systems inaccuracies have to be taken into consideration and corrective actions must be performed. Nature based algorithms are capable to effectively cope with these problems. As learning algorithms can be effectively used to deal with these problems. The analytic approach becomes too complex and too difficult to implement. By differentiating types of motions (e.g. reflex, partially or fully coordinated motions), different amount of feedback from robotic systems should be considered. Force feedback piezoelectric sensors mimic nature's ability to identify changes in force. Meanwhile motions can be planned based on motion primitives, i. e. high quality parts of motion. When planning a motion from a database of known primitives, and considering the changing environment, small sample size and high dimensionality data problem occurs. However, this problem is common in real world applications, and can be solved by similar means. Furthermore, processes occurring in nature are highly parallel as well. By using parallel algorithms problems are can be solved in a more natural way.

There are three traditional methods used to solve inverse kinematics [35, 38]: geometric [22, 41], algebraic [20, 25, 46, 57] and iterative [39] methods. Traditional methods become very complex (both in a mathematical structure of the formulation and in a computation time), when degree of freedom increases [38]. Furthermore, robots have to work in the real world that cannot be modeled concisely using mathematical expressions. These methods only are a solution to a fixed geometrical configuration of a kinematic chain and if kinematics change, for example, is a robot's leg is damaged, it is necessary to find a new inverse kinematics solution.

A neural network-based exploratory learning [26] and quadratic programming [81] was also used. Elman network was also successfully used to solve the inverse kinematics problem [60]. Radial basis function neural network can also be used as a solution [43, 60].

There are also other algorithms used to solve the inverse kinematics problem: the cuckoo optimization, the imperialist competitive [4], and the genetic [14].

These methods proved to overcome problems of traditional methods and provide a good basis of adaptive kinematic solutions that are not constrained of a kinematics. Efficiency of these networks depends largely on the data taken and the scenario [78]. So there is no single best algorithm to solve the inverse kinematics problem.

Artificial neural networks are widely researched to solve the forward and inverse kinematics problems [10, 58]. Many different intelligent methods (based on Neural Networks [29], Fuzzy Logic [1, 53], Reinforcement Learning [79], etc.) were proposed to solve the inverse kinematics problem of different robotic systems [2, 50, 52, 73].

Research Object

Nature based techniques to improve robotic motions. Single layer and multi-layer perceptrons and their applications in robotic motion control.

Research Methodology

Single layer and multi-layer perceptrons were modeled to aid inverse kinematics calculation. Single layer and multi-layer perceptron algorithms were theoretically and practically analyzed in a changing environment. Perceptrons learning rapidity problem was analyzed. Most algorithms were implemented to use parallel computing. Matlab, Javascript and Python programming languages were used for most of the tasks. Inverse kinematics, homogeneous transformation, trajectory generation, statistical, and artificial neural network based analysis were used. Experiments were conducted using a physical robot model.

Scientific Novelty

Inverse kinematics problem solving in changing task environment. Proposed solution extends inverse kinematics solving analythic algorithms with multi-layer perceptron-based corrections. Impact of different single layer and multi-layer perceptrons activation functions and parameters on systems adaptivity. Analysis of percceptrons rapid learning in solving inverse kinematics problem. Primitives based algorithm of hexapods robot motion trajectory planning. Piezoelectric force sensors as feedback system for robot legs. Algorithms were implemented, taking into account the different kind of motions accouring in nature.

Practical Significance

Results of the thesis were used to improve rough terrain walking hexapod robot inverse kinematics problem solving and motion trajectory planning. Research results can also be used to improve other walking robots and serial manipulators for inverse kinematics problem solving and motion trajectory planning tasks. Results about single layer and multi-layer perceptrons learning analysis can be used in different fieelds, analyzing perceptron based algorithms.

Defending Propositions

1. Multilayer perceptrons can extend analytic methods and increase inverse kinematics accuracy for real robotic system.
2. To accomplish different motion types, different amount of feedback from robotic system is required. Piezoelectric force sensors can be used to get force feedback from hexapod robots legs. Measurements can be used in motion control algorithm development.
3. Trajectory primitives and splines can be used to improve hexapod robots' trajectory planning.
4. Changing task environment can be modeled by using learning agents. Ideas from natural evolution can be applied to analyze the behaviour of single layer and multi-layer perceptrons. These algorithms can be effectively parallelized. Results can be verified with real hexapod (six-footed) robot.

The Scope of the Scientific Work

This thesis consists of 87 pages, divided into 3 chapters. 86 literature sources were cited. 53 figures, 10 tables and 46 numerated equations were used.

Structure of the Thesis

This thesis falls into the 3 chapters. The first chapter theoretically analyze motion control in robotic systems. An artificial neural networks are discussed to aid a robotic motions. In the second chapter practical experiments with simulated data are carried out. The third chapter emphasize experiments with actual walking six-footed robot.

Chapter 1

Motion Control in Robotic Systems

English neurologist J. H. Jackkson (1835–1911) described human motoric system as a hierarchically formed one. Human motion can range from automatic reflexes to highly coordinated motions. Automatic motions are controlled by the spinal cortex and the brain stem, while complex motions are controlled by the brain itself. This is parallel to a learning process – an already learned motion can be executed immediately and more complex ones require re-learning and feedback control.

A Russian neuropsychologist N. Bernstein (1897–1966) described logical operations that takes place in human motion control [63]. First, the desired position is calculated, then it is compared to the current position. The difference between them helps to form a motoric program. A motion is then executed by minimizing the difference between the desired and the current position. If the motion is slow enough, feedback mechanism can be used to correct it. Posture-dependent planning is widespread in the brain [3]. This model is directly connected to the inverse kinematics problem in robotics. By varying the amount of feedback and single- and multi-layer perceptron training parameters, different kind of motions can be mimicked.

Motions of living organisms can be subdivided into several groups:

- Reflexes;
- Stereotypical motions;

- Fully coordinated motions.

These motions vary in feedback amount taken into consideration. Chapter 3 describes force feedback sensor offered to use with hexapods robot leg.

Different kinds of trajectories can be created by combining trajectory or motion primitives [28, 56, 74, 76]. Motion primitives also were used in hexapods robot motion planning [83]. If none of the existing primitives are suitable for the task, new primitives can be constructed, using optimization techniques. Seldom used primitives can be removed (or forgotten) from database.

1.1 Multi-Agent Systems and Evolution in Nature

A multi-agent system is a computerized system composed of multiple interacting intelligent agents. These systems are widely used in computer modeling to tackle issues that are difficult or impossible for an individual agent or a monolithic system to solve [51]. Multi-agent systems are widely used in modeling [15, 24, 72]. They mimic processes that happen in nature. Nature-based evolution can also be modeled using multi agent systems. In robotics it can be used to study system adaptivity, as well as neural network parameters, to tune physical robotics system parameters.

Evolution as a process is composed of two parts:

1. A mechanism that provides a variable of organisms. Changes to organisms are mainly random and affect future generations. They are made disregarding consequences to an organism.
2. A changing environment which screens of organism changes. The environment provides stress on the variable organisms that selectively allows certain changes to become dominant and certain others to be eliminated, without consideration to the future of the mechanism. That same process provides a mechanism (an organism) disintegration, if a strong screening environment is not present. Evolution is a two-way process which does not

always work as an advantage to the organism in the long run and, in fact, it often becomes quite deadly to a given species and, thereby, eradicates it.

This process of evolution was used in modeling multi-agent systems.

1.2 Forward and Inverse Kinematics

For serial robot manipulators, a vector of Cartesian space coordinates c is related to the joint coordinates q by:

$$c = f(q), \tag{1.1}$$

where $f()$ is a non-linear differential function. If the Cartesian coordinates c were given, joint coordinates q can be obtained as:

$$q = f^1(c). \tag{1.2}$$

Figure 1.1 illustrates the inverse kinematics problem. First of all, the desired position is observed in visual coordinate space. Then, joint angles $\Theta_1, \Theta_2, ..., \Theta_n$ are calculated and the transformation in the joint angle vector coordinate space can be made. This reduces the distance between the current and the desired position.

There are still many reasons as to why the deviations of feet coordinates or leg trajectory occur. Even simple real world rigid structure displays errors if compared to the modeled structure. Furthermore, mechanical wear off and structural flexibility adds even more errors to the system. After it was used, the system needs to be re-calibrated so it would perform as expected. Re-calibration of multiple servo motors can be time consuming task. Analytic methods of solving inverse kinematics problem solving turn out being insufficient.

The multi-layer perceptron with hybridization of gravitational search algorithm was proposed to solve inverse kinematics problem of a 6-degrees-of-freedom robotic arm [8].

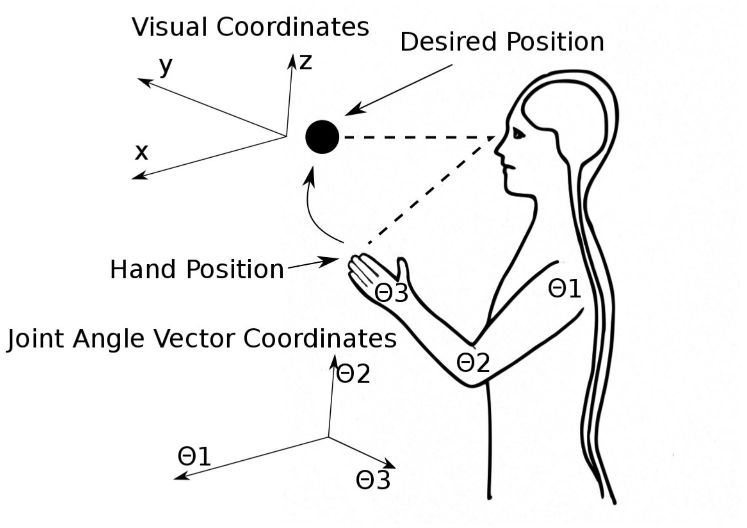


Figure 1.1: Inverse kinematics.

The multi-layer perceptron, as a gradient-based learning algorithm, can cause a very slow training process [23]. This property is tested in Chapter 2, where the multi-layer perceptron is used not only to establish an inverse kinematics solution, but also to retrain it for a new task. Results showed, that an over-trained multi-layer perceptron cannot adapt to sudden task changes.

Instead of finding all possible answers, the neural network can be trained to find a single solution similar to used in training data. This task proved to be challenging, since it is difficult to train the neural network to solve the inverse kinematics problem with the desired precision in a whole coordinate space. Different techniques were proposed to break down the problem into several smaller ones: using several neural networks to solve the problem in different parts of coordinate space [54, 55] or by using a smaller arm mounted on to bigger one, to fine down the crude motion of the bigger arm [84]. All these techniques do not require re-training of the neural network. New approach of retraining existing neural network is proposed in Chapter 2. Experiments with the hexapod robot are described in Chapter 3.

1.2.1 2-Degrees-of-Freedom Robotic Arm

Forward kinematics of 2-degrees-of-freedom arm can be described trigonometrically as follows: We limit angles $\alpha(0-180^\circ)$ and $\beta(0-180^\circ)$. Searching for point (x, y):

$$\begin{cases} \alpha < 90^\circ \\ x_1 = -l_1\cos(\alpha) \\ y_1 = l_1\sin(\alpha) \end{cases} \quad \begin{cases} \alpha = 90^\circ \\ x_1 = 0 \\ y_1 = l_1 \end{cases} \quad \begin{cases} \alpha > 90^\circ \\ x_1 = l_1\cos(180^\circ-\alpha) \\ y_1 = l_1\sin(180^\circ-\alpha) \end{cases} \quad . \tag{1.3}$$

l - points the distance from a coordinate zero point:

$$l = \sqrt{l_1^2 + l_2^2 - 2l_1l_2\cos(\beta)} \qquad . \tag{1.4}$$

δ - points the angle from the coordinate zero point:

$$\delta = \alpha - \arccos(\frac{l_1^2 + l^2 - l_2^2}{2l_1l}). \tag{1.5}$$

$$\begin{cases} \delta \le 0 \\ x = -l\cos(-\delta) \\ y = -l\sin(-\delta) \end{cases} \quad \begin{cases} 0 < \delta < 90^\circ \\ x = -l\cos(\delta) \\ y = l\sin(\delta) \end{cases} \quad \begin{cases} 90^\circ \le \delta \\ x = l\sin(\delta-90^\circ) \\ y = l\cos(\delta-90^\circ) \end{cases} \tag{1.6}$$

Inverse kinematics of the 2-degrees-of-freedom arm can be described trigonometrically as follows: We have a point (x, y), searching for angles α, β.

First we find points the distance to the coordinate zero point

$$l = \sqrt{x^2 + y^2} \qquad , \tag{1.7}$$

β angle can be found by

$$\beta = \arccos(\frac{l_1^2 + l_2^2 - l^2}{2 l_1 l_2}), \tag{1.8}$$

angle between a point and the coordinate zero point can be found by

$$\omega = \arccos(\frac{l_2^2 + l^2 - l_1^2}{2 l_2 l}), \tag{1.9}$$

point against y, by

$$\rho = \arcsin(\frac{y}{l}). \tag{1.10}$$

$$\begin{cases} x < 0 \\ y < 0 \\ \alpha = \rho + \omega \end{cases} \quad \begin{cases} x < 0 \\ y > 0 \\ \alpha = \rho + \omega \end{cases} \quad \begin{cases} x > 0 \\ y > 0 \\ \alpha = 180^\circ - (\rho - \omega) \end{cases} \tag{1.11}$$

Though this simple example of 2-degrees-of-freedom arm can be described analytically, more complex examples become extremely difficult to describe in this manner. Even this example shows non-linear nature of the inverse kinematics problem. Many learning methods were used to address this problem, as described in Section 1.2.

1.3 Artificial Neural Networks

A number of problems can be solved by neural networks [34]:

- Pattern classification. The task of pattern classification is to assign an input pattern represented by a feature vector to one of many prespecified classes. This was used to classify new motions.
- Clustering/categorization. In clustering, also known as unsupervised pattern classification, there is no training data with known class labels. A clustering algorithm explores the similarity between the patterns and places similar patterns in a cluster. Clustering was used to classify motions.
- Function approximation. Suppose a set of n labeled training patterns (input-output pairs), $(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), ..., (\mathbf{x_n}, y_n)$ have been generated from an unknown function $\mu(\mathbf{x})$ (subject to noise). The task of function approximation is to find and estimate say $\hat{\mu}$, of the unknown function μ. This was used to solve the inverse kinematics problem.
- Prediction/forecasting. Given a set of n samples $y(t_1), y(t_2), ..., y(t_n)$ in a time sequence, $t_1, t_2, ..., t_n$, the task is to predict the sample $y(t_{n+1})$ at some future time t_{n+1}.
- Optimization. The goal of an optimization algorithm is to find a solution satisfying a set of constraints such that an objective function is maximized or minimized.
- Content-addressable memory. Associative memory or content-addressable memory, as the name implies, can be accessed by their content. The content in the memory can be recalled even by a partial input or a distorted content.
- Control. Consider a dynamic system defined by a tuple $u(t), y(t)$, where $u(t)$ is the control input and $y(t)$ is the resulting output of the system at time t. In model-reference adaptive control, the goal is to generate a control input $u(t)$ such that the system follows a desired trajectory determined by a reference model.

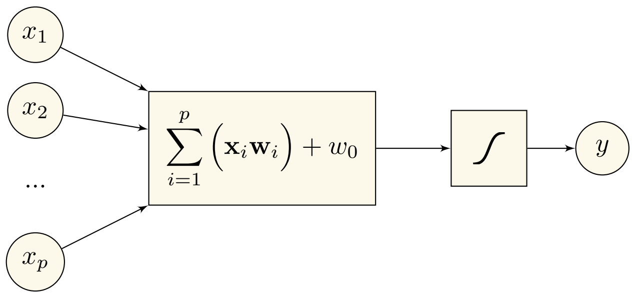


Figure 1.2: Single layer perceptron.

1.3.1 The Single Layer Perceptron

The simplest biologically motivated adaptive information processing unit is a single layer perceptron [61]. It was the first algorithmically described neural network. McCulloch and Pitts (1943) introduced the idea of neural networks as computing machines [47]. Hebb (1949) postulated the first rule for self-organized learning. Rosenblatt (1958) [68] proposed this perceptron as the first model for learning with a teacher (i.e., supervised learning) [31].

Single layer perceptron (Figure 1.2) consists of a number (say p) of inputs $x_1, x_2, ..., x_p$, one output y and performs operation $y = f(arg)$, where $arg = w_0 + x_1w_1 + ... + x_p$. w_p is a linear weighted sum of inputs; $w_0, w_1, w_2, ..., w_p$ are the weights (unknown coefficients, connections to be learned from training data).

Single layer perceptrons can be connected in various configuration networks. Multi-layered feed-forward networks with a non-polynomial activation function can approximate any function [42]. But the more complex a network, the more complex the learning process, the more easily the network adapts to solve a single task. It cannot be easily re-trained. In most cases, for the simplest network is preffered to solve the task with a desired precision.

The basic way to use the neural network is to train it once for a particular task. That is, when a task changes, either the neural network is re-set and trained again or a different configuration of the neural network is used. An already trained neural network can continue to be trained to adapt to a changing task. In some cases this is more efficient than re-training neural network completely. Furthermore, different behavior that exists in nature can be simulated. This includes aging process, ability to adapt to changes, inability to learn, laziness, and other. Perceptron aging was observed, when solving the inverse kinematics problem. Different techniques can be used to control the aging process [61, 63].

1.3.2 The Multi-layer Perceptron

Multilayer perceptron (Figure 1.3) is an artificial neural network with one or more hidden layers. The network shown here is fully connected. This means that a neuron in any layer of the network is connected to all the neurons (nodes) in the previous layer. Signal flow through the network progresses in a forward direction, from left to right and on a layer-by-layer basis. Single layer perceptron's ability to solve the inverse kinematics problem is limited to very simple cases. It is possible to connect multiple single layer perceptrons, but a real multi-layer perceptron has an advantage – the hidden layers. The following three points highlight the basic features of multi-layer perceptrons:

- The model of each neuron in the network includes a non-linear activation function that is differentiable.
- The network contains one or more layers that are hidden from both the input and output nodes.
- The network exhibits a high degree of connectivity, the extent of which is determined by synaptic weights of the network [31].

A popular method for training multi-layer perceptrons is a back-propagation algorithm. The training proceeds in two phases:

- In the forward phase, the synaptic weights of the network are fixed and the input signal is propagated through the network, layer by layer, until it reaches the output. Thus, in this phase, changes are confined to the activation potentials and outputs of the neurons in the network.
- In the backward phase, an error signal is produced by comparing the output of the network with a desired response. The resulting error signal

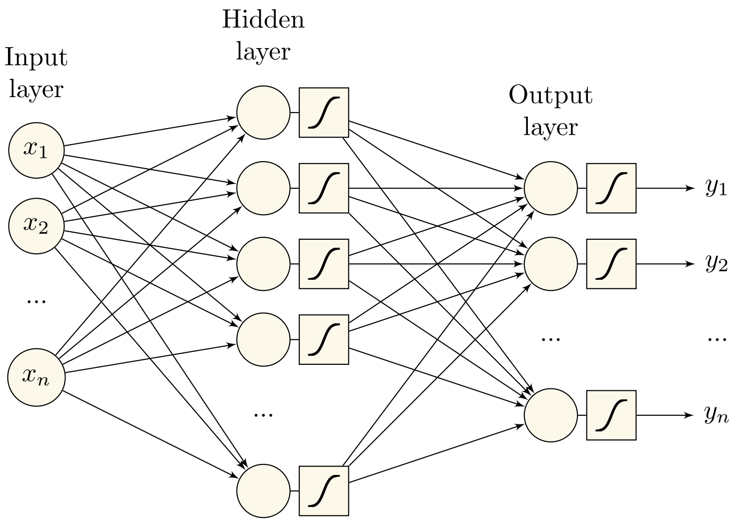


Figure 1.3: Multilayer perceptron.

is propagated through the network, again layer by layer, but this time the propagation is performed in a backward direction. In this second phase, successive adjustments are made to the synaptic weights of the network. Calculation of the adjustments for the output layer is straightforward, but it is much more challenging for the hidden layers [31].

1.3.3 Error Calculation

Several different methods exist to calculate the perceptrons error between targeted and calculated outputs:

1. $MAD = \frac{\Sigma|e_t|}{N}$ means absolute deviation;
2. $SSE = \Sigma(e_t)^2$ sum of squared errors;
3. $MSE = \frac{\Sigma(e_t)^2}{N}$ means squared error;
4. $RMSE = \sqrt{MSE}$ root means squared error;
5. $MAPE = \frac{1}{N}\Sigma|\frac{e_t}{e_y}|(100)$ means absolute percentage error.

Where: e_t - error of one dataset; y_t - calculated value at output; N - count of error terms. MSE and RMSE error calculation will be used in further experiments.

1.3.4 Non-linear Perceptron Learning

In this section perceptron learning is analyzed in a theoretic perspective. Later we use perceptrons to learn the inverse kinematics of a robotic arm. Then the task is changed and perceptrons must adapt to changes. Psychology studies [65, 66] analyze the learning process in the nature and find simmilarities to the model described in the equation 1.13. Understanding the learning process enables rapid learning in robotic systems. This allows the adaptation of an existing system to the changing environment. In order to save the information contained in the initial weight vector and reduce the generalization error, it is necessary to stop training in due time [62]. In this section we demonstrate that in the gradient descent perceptron training excessive magnitudes of the initial weight vector componets could slow the learning speed of non-linear single layer perceptron based classifiers [61].

A feedback chain and non-linearity of the activation function of a single layer perceptron cause a number of non-linear phenomena in the single layer perceptron training and make the learning rapidity analysis very complicated.

The single layer perceptron training is a difficult task, since it faces non-linearities [5]. This is showed in Figure 1.4.

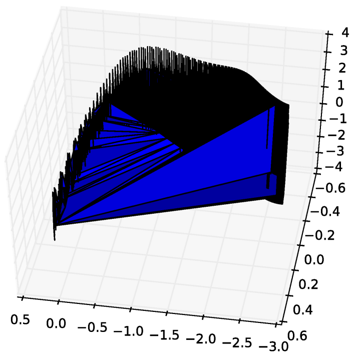


Figure 1.4: Cost function of a perceptron after training to solve the inverse kinematics problem. x axis denotes summed weights, y axis - denotes w_0, and z axis - errors.

Let objects or situations, are classified into one of the two pattern classes, Π_1 and Π_2, be described by p-dimmensionalfeature vectors $\dot{\boldsymbol{x}} = (x_1, x_2, ...x_p)^T$. In order to train a perceptron, we use a training set of composed of vectors

$\dot{\boldsymbol{x}}_j^{(i)}, (i = 1, 2; j = 1, 2, ..., N)$, where the superscript "^{T}" denotes the transposition operation and N_1, N_2 are training sample sizes. In order to find the weights, the cost function of sum of squares is minimized.

$$\begin{aligned} cost &= \frac{1}{N_1 + N_2} \sum_{i=1}^{2} \sum_{j=1}^{N_i} (t_j - f(\boldsymbol{v}^T \dot{\boldsymbol{x}}_j^{(i)} + v_0))^2 = \\ &= \frac{1}{N_1 + N_2} \sum_{i=1}^{2} \sum_{j=1}^{N_i} (t_j - f(\boldsymbol{w}^T \boldsymbol{x}_j^{(i)}))^2, \end{aligned} \tag{1.12}$$

where t_1 and t_2 are targets (desired outputs), and $f(net)$ is a smooth limiting non-linear activation function, $\boldsymbol{w} = (v_1, v_2, ..., v_p, v_0)^T$, $\boldsymbol{x} = (x_1, x_2, ..., x_p, 1)^T$. p-dimensional feature vector $\dot{\boldsymbol{x}} = (x_1, x_2, ..., x_p)^T$. Training set composed of vectors $\dot{\boldsymbol{x}}_{\boldsymbol{j}}^{(\boldsymbol{i})}$, $(i = 1, 2; j = 1, 2, N_i)$.

The main factor that determins the learning speed of a new task, or when task changes is the $(p + 1)$-dimensional gradient vector

$$\frac{\partial cost}{\partial w_i} = \eta \times \frac{1}{N} \sum_{i=1}^{2} \sum_{j=1}^{N} \gamma_{grad}(t_j^{(i)}, net_j^{(i)}) \times \frac{\partial net_j^{(i)}}{\partial w}, \tag{1.13}$$

where $\gamma_{grad}(t_j^{(i)}, net_j^{(i)})$ is a scalar variable called a gradient coefficient, and the term, $\frac{\partial net_j^{(i)}}{\partial w} = \boldsymbol{x}_{(\boldsymbol{i})j}$ is a $(p + 1)$-dimensional training vector. For the sigmoid activation function, term

$$\gamma_{grad}(t_j^{(i)}, net_j^{(i)}) = 2(t_j^{(i)} - f(net_j^{(i)})) \times f(net_j^{(i)}) \times (1 - f(net_j^{(i)})), (i = 1, 2; j = 1, 2, ..., N_i) \tag{1.14}$$

is determined by two product terms:

1) a derivative of the activation function,

$$\frac{\partial f(net_j^{(i)})}{\partial net_j^{(i)}} = f(net_j^{(i)}) \times (1 - f(net_j^{(i)})) = o_j^{(i)} \times (1 - o_j^{(i)}), \tag{1.15}$$

2) an error signal

$$t_j^{(i)} - o_j^{(i)} = t_j^{(i)} - f_s(net_j^{(i)}). \tag{1.16}$$

Equation 1.14 shows that the gradient coefficient, $\gamma_{grad}(t_j^{(i)}, net_j^{(i)})$, depends on the target $t_j^{(i)}$ and on the weighted sum, $net_j^{(i)}$.

Let us assume that the magnitudes of weighted sums, $net(\omega) = x_1\omega v_1 + ... + x_p\omega v_p + \omega v$ depend on a positive scalar ω (a coeficient of the weighted magnitude). In figure 1.5 we see the output values as functions ot the coefficient ω.

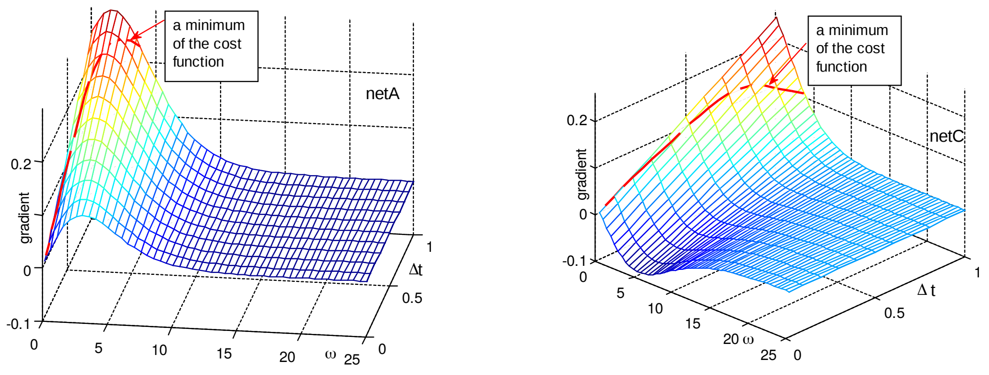


Figure 1.5: 3D plots of the gradient as functions of ω and Δt for two values of the weighted sums for two different networks.

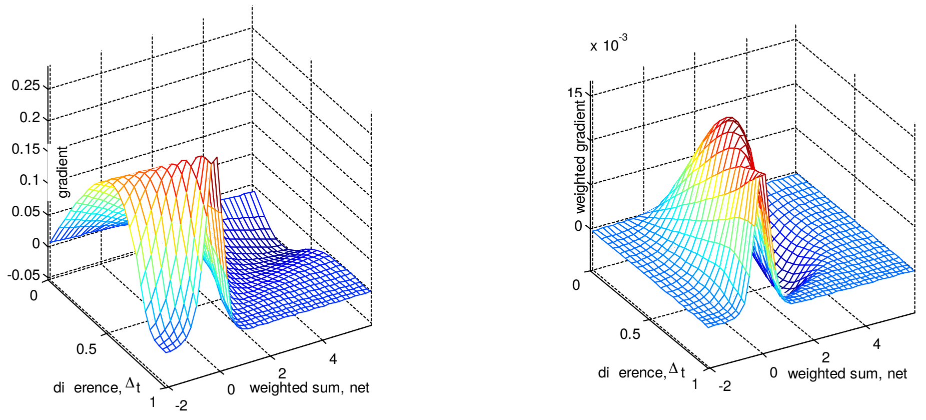


Figure 1.6: Dependence of the gradient and the weighted gradient values on the weighted sums, net, and difference Δt at the end of the training process (after minimization of the cost function).

Experiment results depicted in figures 1.5 and 1.6 demonstrate that the relationship between the gradient and the weight magnitudes Δt values is non-linear.

As seen in the figures 1.5 and 1.6, it is difficult to train a perceptron when gradient is low (flat surface). The gradient shows the rapidity of a learning process. Re-training of the same network may cause local minima problems. If a network is over-trained it contains a large number of flat surfaces in the cost function wherefore it fails to adapt to a changing environment rapidly. This is wery important for solving the inverse kinematics problem in robotics. The two alternatives for training a network in changed environments is noise injection and/or the reduction of targets.

1.4 Trajectory Planning

To ensure smooth walking of the hexapod, the movement of all legs must be synchronized. The simplest method to perform walking motion is to use parabola shaped trajectories to perform all the steps of the robot. This method can be improved by using trajectory primitives and splines.

1.4.1 Parabola Trajectory Planning

Different walking patterns are displayed in Figure 1.7 [36, 67, 75]. One step of a robot is completed when all six legs are moved to a new position. As mentioned in Chapter 3, different walking patterns add different force to the legs of the robot. Moreover, the angle of leg placement can cause slipperage or inaccuracies.

The pattern of a single leg movement is displayed in figure 1.8. This is the simplest parabola shaped trajectory. Parabola shaped trajectories have disadvantages of not being robust and energy efficient. They cannot adapt to the desired walking speed or different walking surface conditions. We propose to improve a trajectory generation by using trajectory primitives and splines.

1.4.2 Trajectory Planning Using Primitives

Each primitive is a single step of a very high quality. Instead of constructing the whole motion out of primitives, only the most important parts of motion can

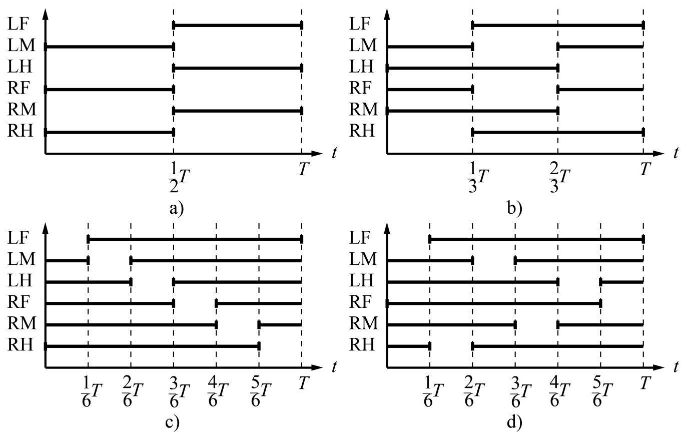


Figure 1.7: Gait diagrams [44]. a) tripod gait, b) tetrapod gait, c) wave gait, d) ripple gait

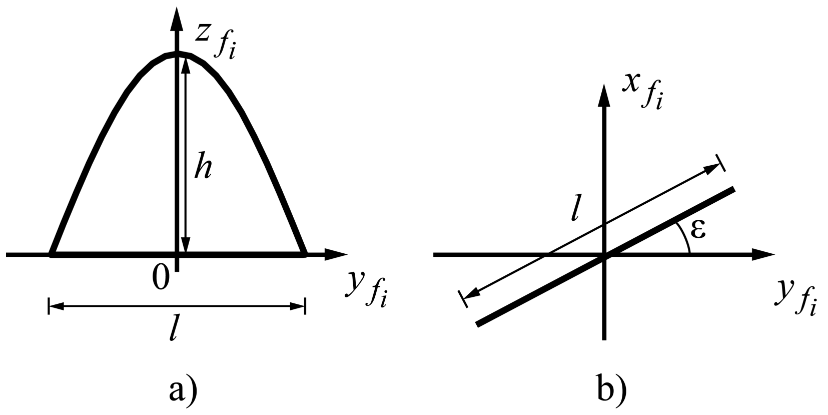


Figure 1.8: Trajectory projections [44].

be described by primitives. This allows the flexibility to generate the rest of a trajectory.

Taking a hexapod walking robot as an example, the most important parts of motion are beginning and ending in the motion, e.g. leg must be lifted straight up from the sand, or steeply and only then run. End of motion can also be different, straight down to a slippery surface, or with an angle on a non-slippery one. The rest of the trajectory is not so important, but it can be optimized by different criteria.

Multiple criteria can be taken into consideration, when choosing the desired trajectory:

- Surface identification parameters that describe the type of surface, the robot is walking on;
- The desired walking speed;
- The maximum and the minimum height of a trajectory;

Optimization problems of these parameters are left for future research, and are out of the scope of this thesis. Trajectories can be combined from a set of primitives, or only key parts of a motion can be described by primitives. These parts must be connected in a smooth manner, to minimize energy loses and jerkiness of the robotic system. Splines are proposed to be used to connect primitives into a smooth trajectory.

1.4.3 Trajectory Calculation

Splines are often used to calculate s smooth trajectory, connecting two or more points. We introduce a spline as a piecewise polynomial parametric curve $Q(t)$. For this application we analyze splines in the two-dimensional space $Q(t) \in \mathbb{R}^2$. There are three commonly used ways in the literature to define the polynomial segments of a spline [77]:

1. Polynomial in parameter t, using standard notation for a polynomial.
2. Weighted sum of control vertices.
3. Product of matrices.

There are several cubic spline families [77]:

1. Cubic Hermite Spline. It consists of segments that are cubic polynomials. Each segment connects two control points. The two remaining degrees of freedom are controlled by specifying the tangent vectors at both, the start and the end point of the segment

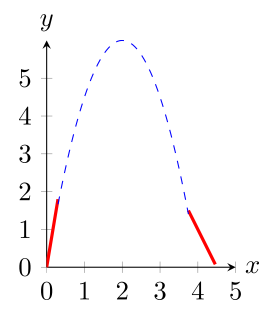


Figure 1.9: 3 stages of motion

2. Cubic Bézier Splines. It also consist of cubic polynomials and therefore have the same expressive power as the Cubic Hermite Splines. They differ from Hermite Splines in the formulation of the segments. Instead of explicitly defining the tangent vectors at the start and the end point, these vectors are extracted from two additional control vertices in the case of Cubic Bézier Splines' segments.

3. Catmull-Rom Splines. So far, to reach C^1 continuity it has been necessary to manually adjust the tangents at the join points. Catmull-Rom Splines overcome this by being inherently C^1 continuous.

4. B-Splines. B-Splines are inherently C^2 continuous. However, this comes at an additional cost of not passing through any of their control points.

Hermite splines (see equation 1.17) were chosen, because of C^1 continuity and because control points are on the curve itself. Catmull-Rom splines are frequently used to get the smooth interpolated motion between key frames. They are popular mainly for being relatively easy to compute, guaranteeing that each key frame position will be hit exactly, and also guaranteeing that the tangents of the generated curve are continuous over multiple segments. One of its drawbacks is the requirement of additional two points at the ends of a curve. But this drawback turns into an advantage, because the curve is being fitted between two primitives.

$$\boldsymbol{p}(s) = \begin{bmatrix} 1 & u & u^2 & u^3 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\tau & 0 & \tau & 0 \\ 2\tau & \tau - 3 & 3 - 2\tau & -\tau \\ -\tau & 2 - \tau & \tau - 2 & \tau \end{bmatrix} \begin{bmatrix} p_{i-2} \\ p_{i-1} \\ p_i \\ pi + 1 \end{bmatrix}. \tag{1.17}$$

1.5 Conclusions

Single layer and multi-layer perceptrons can be used to solve the inverse kinematics problem in robotics. They can also be used to improve the inverse kinematics solution that was obtained by using traditional methods. It was examined how the perceptrons learning rapidity depends on a gradient of cost function. To speed up the computation methods of noise injection and/or reduction of targets were examined. If the weights are large and the activation function is saturated, learning stops and the inverse kinematics learning speed becomes very slow. Single layer and multi-layer perceptrons can be trained to cope with the changing task environment. Weight minimization and noise injection methods help addressung the learning rapidity. Multi-agent systems were used to model the changing the task environment. In certain conditions the speedup can be achieved close to the ammount of the processors count of a computer. By using the trajectory primitives combined with Catmull-Rom splines smooth trajectories were generated. This method addresses problems found in Chapter 3, after force feedback sensor experiments.

Chapter 2

Improvement of Motions in Robotic Systems

Serial manipulators are designed as a series of links connected by motor-actuated joints that extend from a base to an end-effector. An artificial neural network that solves the inverse kinematics problem of robotic arm has often been used in many research. It is difficult for a well-known neural network to solve inverse kinematics problem with an acceptable accuracy for the whole joint space [54]. Different techniques have been used to break down the problem into several smaller ones. In this chapter, we propose to solve the problem in a chosen part of a coordinate space, and then train the neural network further, to solve the problem in different parts of the coordinate space. The approach of re-training the same neural network to solve different problems mimics the learning process and the ability to adapt to the changing environment in the nature. Experiments were carried out to test, if it is beneficial to re-train existing networks, or it is better to train new networks from the initial parameters.

2.1 Single Layer Perceptron Experiment

The inverse kinematics problem of 2-degrees-of-freedom robotic arm was solved using 2 single layer perceptrons (1 for each joint). Distance and angle of the desired position were used as input data and joint angles as output data for the neural network. Three different tasks were chosen in a coordinate space as shown in figure 2.1. The neural network was trained to solve task 1, then retrained for task 2 and task 3. Tasks were chosen as inverse kinematics problems in different places of a coordinate space. Results were verified on testing data set. A means square error over iterations of testing data is shown in figures 2.2, 2.3, 2.4. These figures illustrate how under-learning and over-learning of a neural network affects training error when the task changes.

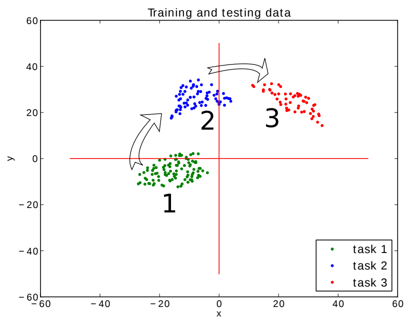


Figure 2.1: Changing task.

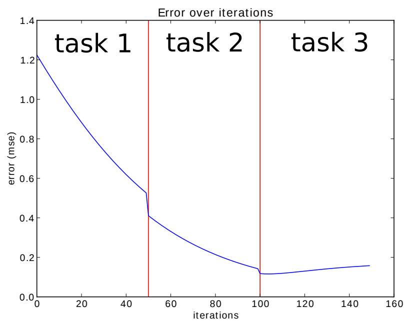


Figure 2.2: NN trained for 50 iterations.

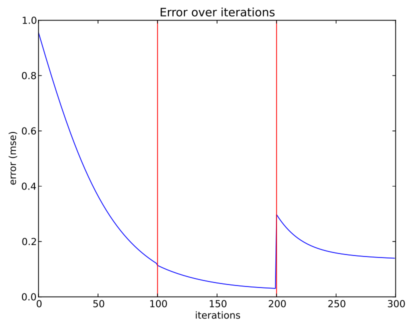


Figure 2.3: NN trained for 100 iterations.

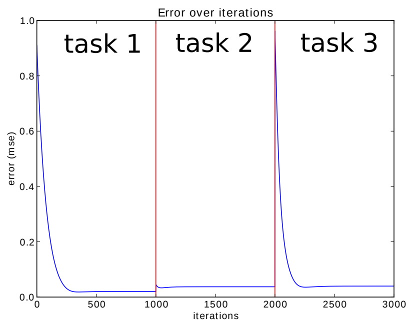


Figure 2.4: NN trained for 1000 iterations.

Task 1 and task 2 were similar to learn for the neural network, while task 3 was different. Similarity is defined, as a similar weight vector of SLP to solve the tasks. Though tasks were similar, they were not the same. The transition between task 1 and task 2 was smooth and neither under-learning (figure 2.2) nor over-learning (figure 2.4) had any effect. But when a task differs (perceptron needs to learn different weights), the over learning effect can be observed (task 2 – task 3 transition). Figure 2.2 shows no over-learning effect, figure 2.3 shows a slight over-learning effect and figure 2.4 shows that it is increasing as iterations increase.

Single layer perceptrons can adapt to the changing inverse kinematics problem. If tasks are similar, only a few iterations are needed to re-train the neural network. If tasks differed more, agents that learned first problem very well, failed on the second one. Over-learning occurs due to the increased weights of perceptron inputs. This process is also called "perceptron aging" [61, 63]. Depending on the situation, different weight reduction techniques like learning step reduction or noise injection to inputs [61] can be used to control perceptron aging. Solving the inverse kinematics problem with neural networks enables choices among learning speed, precision and ability to adapt to the changing environment.

2.1.1 Tasks Clustering and Classification

To obtain different kint of tasks for the robot (e.g., discussed in Section 2.1), existing bigger task can be clustered to the smaller ones (ref. Figures 2.5 and 2.6). Many artificial clustering algorithms exist [86], and are used in statistics, computer science and machine learning. One of the classic clustering algorithms is k-means, first used by James MacQueen in 1967 [45]. Given a set of observations $(x_1, x_2, ..., x_n)$ where each observation is a d-dimensional real vector, the k-means clustering aims to partition the n observations into $k(<= n)$ sets $S = S_1, S_2, ..., S_k$ so as to minimize the within-cluster sum of squares. In other words, its objective is to find:

$$\underset{s}{argmin} \sum_{i=1}^{k} \sum_{x \in S_i} \|x - \mu_i\|^2, \tag{2.1}$$

where μ_i is the mean of points in S_i.

Further research will address motion primitives. To obtain different kind of motion patterns each motion can be carefully selected and described by a set of parameters to fit a desired task. This includes a trajectory primitives selection, as well, as an order of spline being used to connect primitives. By varying these parameters, speed and precision can be traded off. This also can be done in a more natural way, i. e. generating random motion patterns and keeping the useful ones.

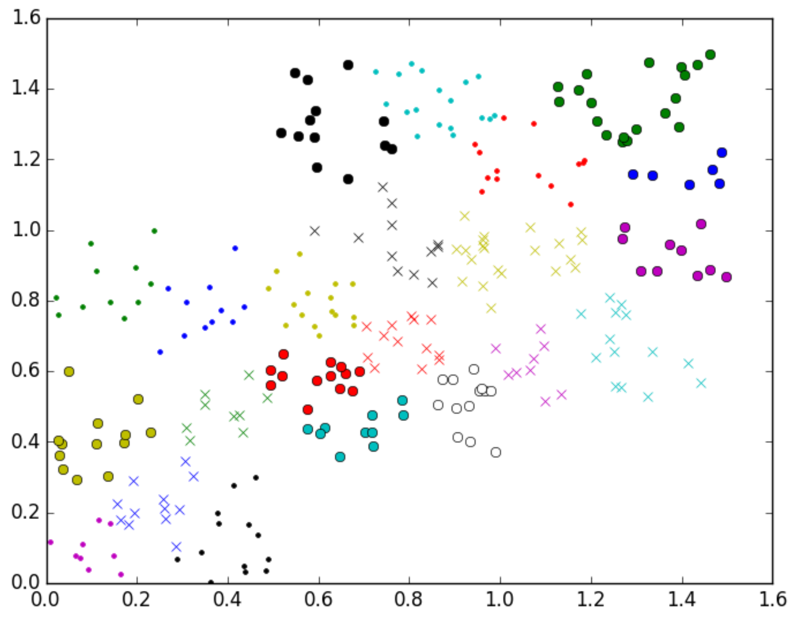


Figure 2.5: 24 classes in 2D space.

When dealing with motion data, a small sample size problem arises. Due to the changing nature of environment, a task being solved and the robotic system (e.g. from parts wearing off, or damage), it is inpractical to have a big database of possible motions data. Decisions must be made from using a small sample size and a high dimensionality data.

2.2 Multi-layer Perceptron Experiment

Multi-agent evolving systems were created to analyze agent's behavior when a task suddenly changes. Each agent incorporates multi-layer perceptron and solves the inverse kinematics problem of 3-degrees-of-freedom robotic arm. The impact of an agent initial parameters was observed while using different activation functions in output nodes of MLP. Linear, logistic, hyperbolic tangent and sine activation functions were investigated.

Intelligent agents and robots operating in new unknown environments must be able to adapt to sudden situational changes [40, 63, 85]. Research of rapid

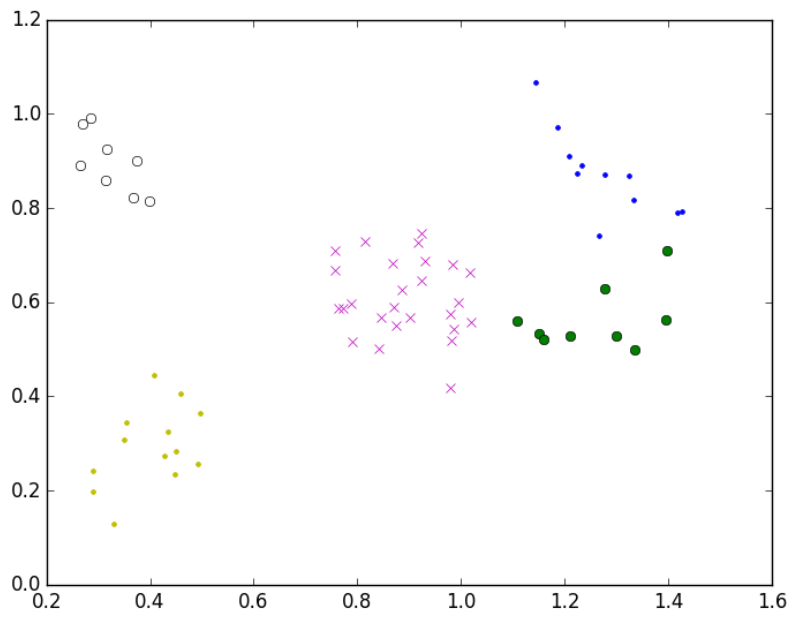


Figure 2.6: Selected 5 classes in 2D space.

adaptation in changing task environment has been carried out in analysis of biology, physics, economy and financial time series [17, 33, 48, 64]. Natural evolution ensures that the only ones, capable to adapt to changes, survives.

The single layer perceptron is the simplest, nature inspired adaptive unit. Perceptrons can be connected to more complex neural networks. The more complex neural networks are being used, the more complex tasks it can solve. Such neural networks are not able to adapt when tasks change. Simpler networks are able to learn smaller tasks, and are able to adapt, when tasks change.

A multi-layer perceptron with 3 nodes in a hidden layer was trained to solve the inverse kinematics problem of 3-degrees-of-freedom robotic arm. Two tasks were chosen in a coordinate space. Figure 2.7 represents tasks in the Cartesian coordinate space. Each task consists of the inverse kinematics problem. Tasks were chosen to be in the opposite sides of arms reachable space. When the multi-layer perceptron is trained to solve one task, other task solving accuracy decreases. This experiment shows how fast accuracy can be regained, when using different activation functions in output layer of the multi-layer perceptron. After the first task was learned for 250 iterations, the second task was suddenly introduced. A change in error was observed when using different activation functions in the output layer of the multi-layer perceptron with 3 nodes in the hidden layer. The impact of agent starting conditions were investigated.

The degree of freedom of a mechanical system is the number of independent parameters that define its configuration. Data of 3-degrees-of-freedom robotic arm was used in this experiment, as shown in the figure 2.7.

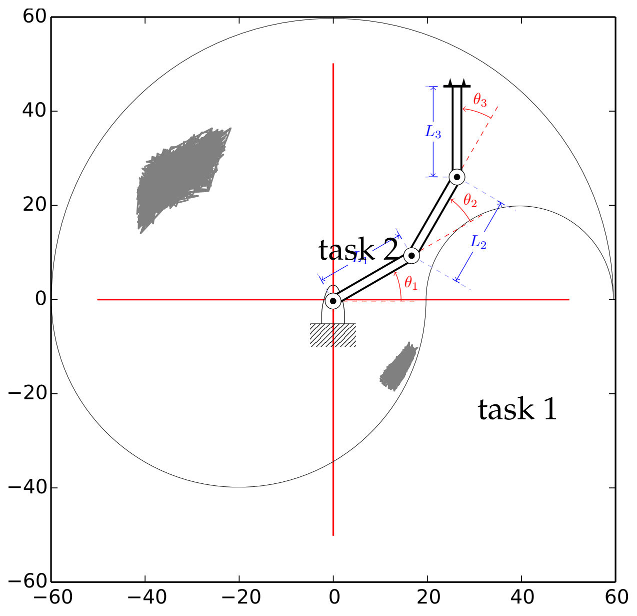


Figure 2.7: Task space of 3 degrees of freedom robotic arm.

There is no single solution to the inverse kinematics problem, but a single solution can be learned by training neural network. If the neural network is trained with consistent data, it will learn to propose a similar solution to a new similar problem.

Every agent was simulated as a multi-layer perceptron with 3 hidden layers. 2 nodes in input layer were used to input 2 dimensional data with desired position coordinates. 3 nodes in output layer were used to calculate joint angles of a 3-degrees-of-freedom serial manipulator. Logistic activation functions were used in hidden layers of multi-layer perceptrons in all experiments. Weights are being updated by the delta rule (equation 2.2). Multi-layer perceptrons had variable learning rate (η) and momentum (α) parameters, and initial weights (w). These parameters were learned by evolution. φ denotes activation an function.

4 different activation functions were used in this experiment (equations 2.6, 2.7, 2.8, 2.9).

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n), \tag{2.2}$$

$$\delta_j(n) = e_j(n) \varphi_j'(v_j(n)). \tag{2.3}$$

$e_j(n)$ is an error signal, computed by a root mean square error (RMSE). The RMSE was measured between actual and predicted outputs:

$$RMSE = \sqrt{\frac{\sum (t_i - y_i)^2}{m}}. \tag{2.4}$$

2.2.1 Data Generation and Preparation

Different methods to gather training data can be used: kinematics equations [7], network inversion [37], simulation programs [18]. Kinematics equations were used to generate data. First of all joint angles $\theta_1, \theta_2, \theta_3$ were chosen from a desired range, then a forward kinematics equation was solved, to get the desired end actuators position. Joint angles of the robotic arm were calculated to be as similar as possible in each desired position. Data was changed from Cartesian coordinates to polar coordinates and normalized as shown in the equation 2.5. Two different data sets were generated, to simulate the two different tasks.

$$norm\_data = \frac{data - mean(data)}{std(data)}. \tag{2.5}$$

2.2.2 Activation Functions

Non-linear activation functions enable the neural network to approximate extremely non-linear functions [27]. An ability to re-train the neural network for a new task is highly based on the activation function being used [63]. An inability

linear
$$\varphi_j(v_j(n)) = v_j(n). \tag{2.6}$$

sine
$$\varphi_j(v_j(n)) = \sin(v_j(n)). \tag{2.7}$$

hyperbolic tangent
$$\varphi_j(v_j(n)) = \tanh(v_j(n)). \tag{2.8}$$

logistic
$$\varphi_j(v_j(n)) = \frac{1}{1+\exp(-v_j(n))}. \tag{2.9}$$

to learn (or aging) is caused by increased synaptic weights, due to the saturation of activation functions [61]. This was also observed while solving the inverse kinematics problem with the single layer perceptron [82].

2.3 Multiagent System Experiment

A multi-agent system was created to test the behavior of an agents in the changing task environment. All experiments were performed, using 100 agents, trained for 100 epochs and for 500 iterations. After 250 iterations task changed and population behavior was observed. All agents follow some simple rules:

- Initial set of agents were created with random initial weights (w), learning rate (η) and momentum (α) parameters;
- After each epoch, agents that performed worst compared to the mean population error were discarded;
- Agents cannot die from old age;
- Population is restored either by cloning the best agent and adding 10% of noise to its parameters, or by creating new agents with random initial parameters (this depends on an experiment).

2.3.1 Multi-threading and MPI Parallelization

An experiment was carried out using a multi-agent neural networks system. Every agent incorporated one single layer perceptron. Every single layer perceptron solved the inverse kinematics problem (like in chapter 2). While solving a

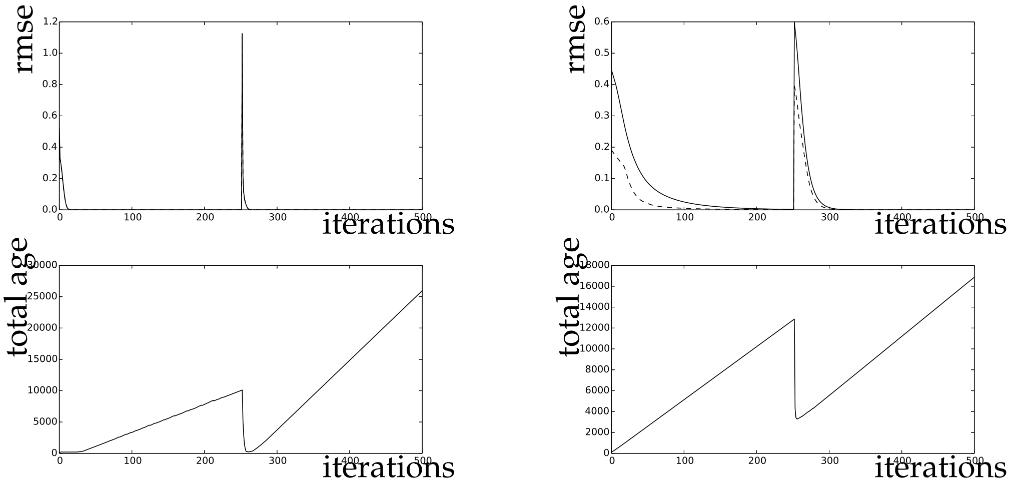


Figure 2.8: The upper figures represent the RMSE over iterations, with the change of task on 250th iteration, using a linear activation function at multi-layer perceptron outputs. Solid lines represent the best 10 agents, dashed lines – the best single agent. Lower figures represent the total age of agents. Figures on the left represent the population and then new agents were cloned from the best agent, on the right – new agents were created with random parameters.

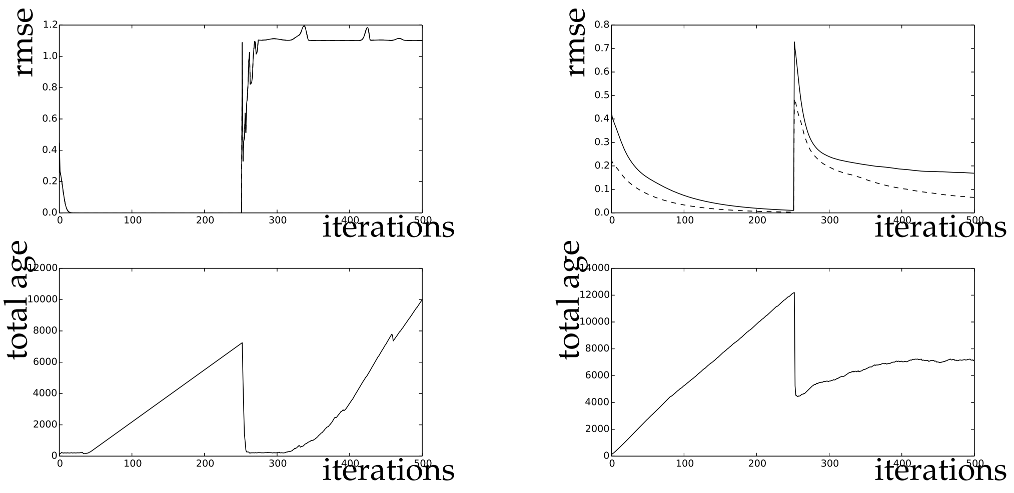


Figure 2.9: The upper figures represents the RMSE over iterations, with the change of task on the 250th iteration, using a sine activation function at multi-layer perceptron outputs. Solid lines represent the best 10 agents, dashed lines – the best single agent. Lower figures represent the total age of agents. Figures on the left represent the population and then new agents were cloned from the best agent, on the right – new agents were created with random parameters.

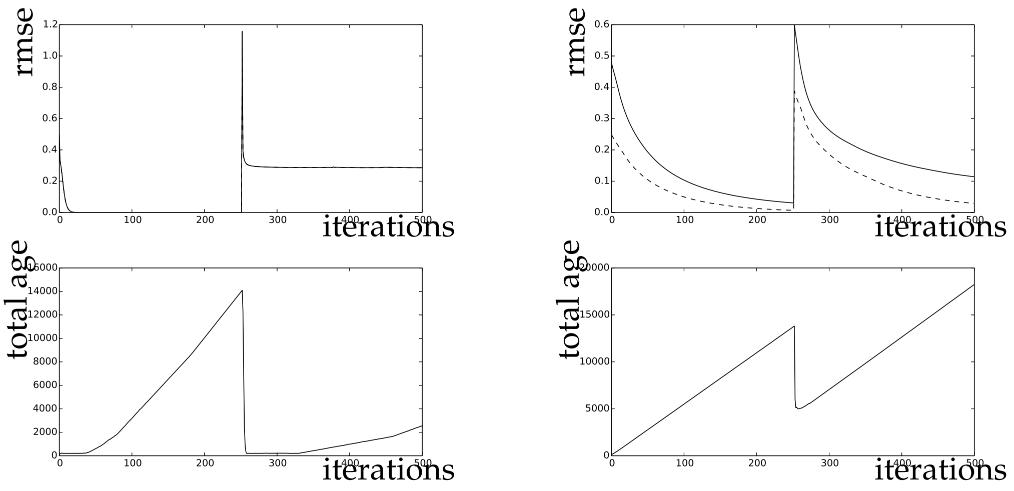


Figure 2.10: The upper figures represent the RMSE over iterations, with the change of task on the 250th iteration, using a hyperbolic tangent activation function at multi-layer perceptron outputs. Solid lines represent the best 10 agents, dashed lines – the best single agent. Lower figures represent the total age of agents. Figures on the left represent the population and then new agents were cloned from the best agent, on the right – new agents were created with random parameters.

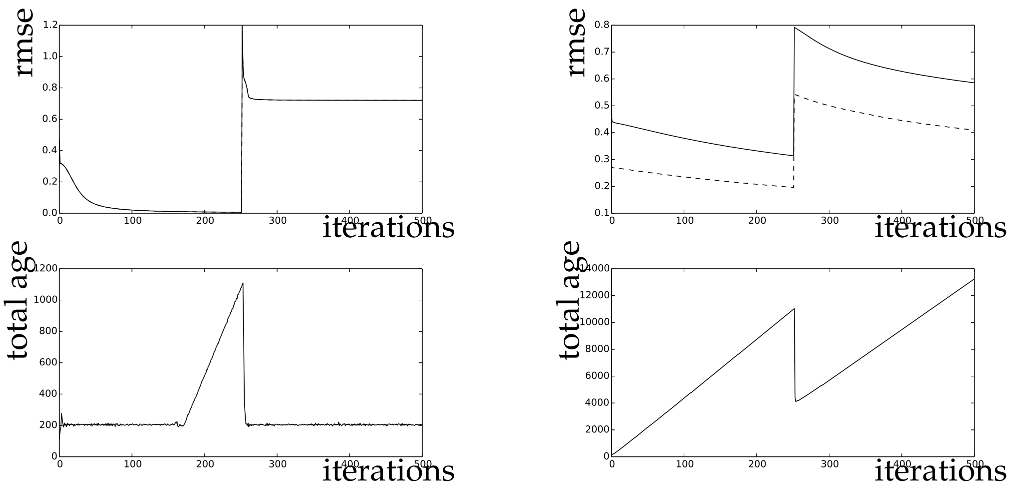


Figure 2.11: The upper figures represents the RMSE over iterations, with the change of task on 250th iteration, using a logistic activation function at multi-layer perceptron outputs. Solid lines represent the best 10 agents, dashed lines – the best single agent. Lower figures represent the total age of agents. Figures on the left represent the population and then new agents were cloned from the best agent, on the right – new agents were created with random parameters.

task, an agent makes a predefined number of iterations. A number of iterations affects a task completion time. Serial problem solving is depicted in Figure 2.12. Agents solve their task independantly from each other.

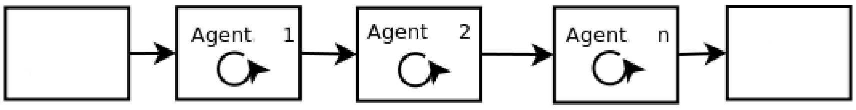


Figure 2.12: Serial flow of software.

It is not difficult to parallelize the work of an agent (see Figures 2.13 and 2.14. Every agent executes the same code, but tasks can be different, if different parameters are being sent. An experiment was carried out with different iterations count per agent and different agents count. The task was solved by dividing agents in some number of threads. The first task was solved starting from one thread (meaning serial), then with 2, 3, 5, 10, 15, and ending with 100 threads. An increase of threads in personal computer caused a system to crash. When more threads were executed in a cluster no interesting results were obtained. When using multi-processing method speedup stopped increasing when the number of threads reached CPU count.

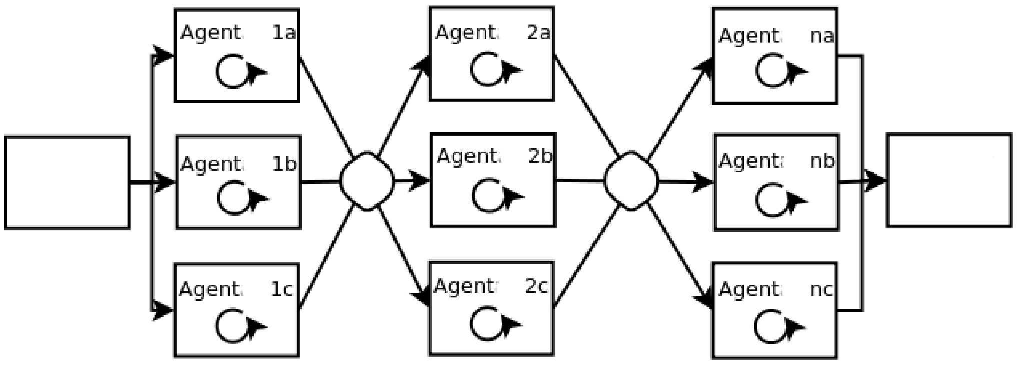


Figure 2.13: Parallel flow of software, using MPI.

A task parallelization with the MPI is depicted in figure 2.13. First of all the pre-defined number of threads are were executed. Then it is waited until they finish. Then a new set of threads are being executed. A task parallelization with multi-processing is depicted in figure 2.14.

21 experiments were executed:
With Python multiprocessing:

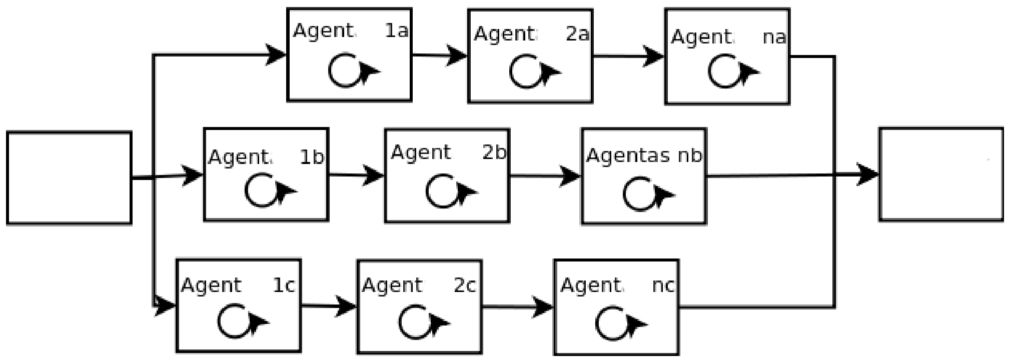


Figure 2.14: Parallel flow of software, using multiprocessing.

1. using 64 cores – 1 experiment (because only 8 cores were used),
2. using 16 cores – 4 experiments (but only 8 cores were used),
3. using 4 cores – 4 experiments,
4. personal computer with 2 cores – 4 experiments.

With Open MPI:

5. using 64 cores – 4 experiments,
6. using 16 cores – 4 experiments,
7. using 4 cores – 4 experiments,
8. personal computer with 2 cores – 4 experiments.

All figures uses following marking:
A – 100 agents, 100 iterations,
B – 100 agents, 1000 iterations,
C – 1000 agents, 100 iterations,
D – 100 agents, 10000 iterations.
These datasets were chosen to find out how speedup differs when a task completion time increases for an agent (A, C, D datasets). Furthermore, if a task completion time differs when more agents with fewer iterations are used, or fewer agents with more iterations are used (B and C datasets).

Personal computer:
Hardware:
Intel Core 2 Duo CPU P8600@2.40GHz
2990 MiB of RAM
Software:
Ubuntu 10.04, 2.6.32 Linux kernel
Python 2.6
OpenMPI 1.5.4
mpi4py 1.2.2

Cluster:
Hardware:
8 Sun Blade X6275,
each one has 2 x 4 Intel Xeon cores
E5520 2.27GHz processors,
64 cores in total.
Software (for multiprocess experiment):
Python 2.6.5
Numpy 1.5.1
Software (for MPI experiment):
Python 2.4.3
Numpy 1.2.1
mpi4py 1.2.2

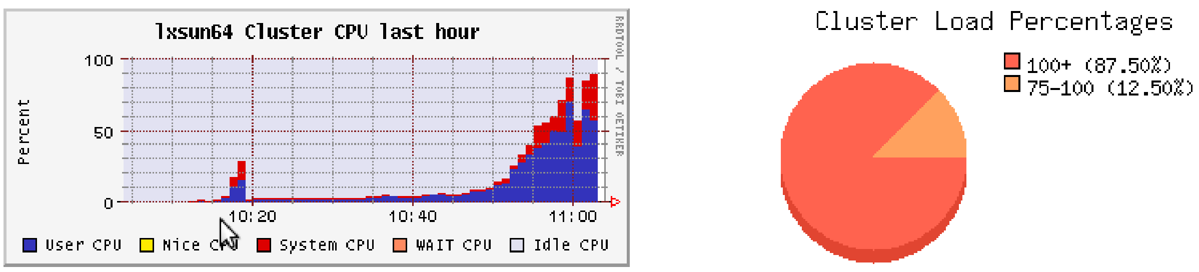


Figure 2.15: Cluster load using MPI parallelization, 64 CPUs

Figures 2.16, 2.17, 2.18 and 2.19 depict the speedup results in multiprocessing system. It must be noted, that parallel computing network was used with 8 CPUs per cluster. No significant speedup was achieved using more than 8 threads. It also can be seen in a cluster load graph (Figure 2.15).

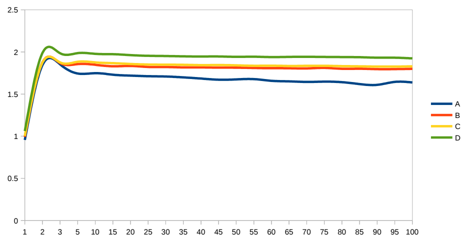


Figure 2.16: Multiprocessing parallelization on a standard personal computer with 2 CPU cores. x axis denotes number of cores, y axis – speedup.

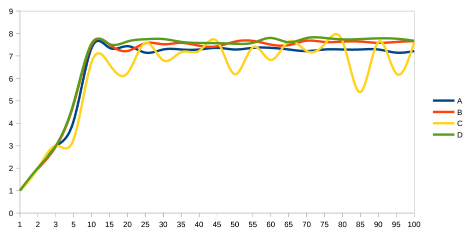


Figure 2.17: Multiprocessing parallelization on a cluster, using 8 CPU cores. x axis denotes number of cores, y axis – speedup.

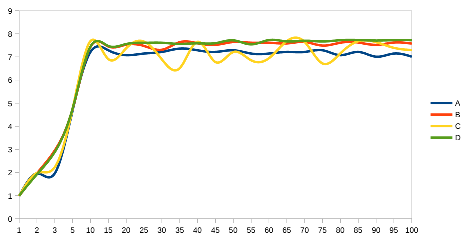


Figure 2.18: Multiprocessing parallelization on a cluster, using 16 CPU cores. x axis denotes number of cores, y axis – speedup.

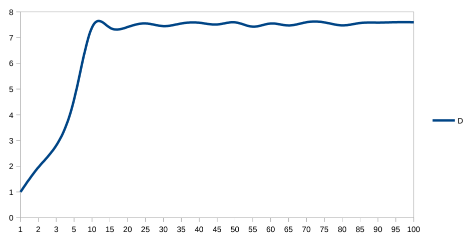


Figure 2.19: Multiprocessing parallelization on a cluster, using 64 CPU cores. x axis denotes number of cores, y axis – speedup.

While using the MPI (Figures 2.20, 2.21, 2.22, 2.23) speedup was achieved beyond 8 CPUs limit. Increased delay to send data over the network can be seen. The network instability (probably dependent on a other tasks being executed on cluster) can be witnessed. Figure 2.24 depicts a cluster load while dealing with MPI tasks.

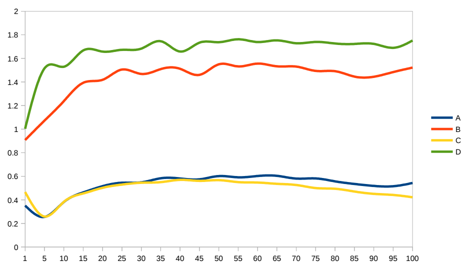


Figure 2.20: MPI parallelization on a standard personal computer with 2 CPU cores. x axis denotes number of cores, y axis – speedup.

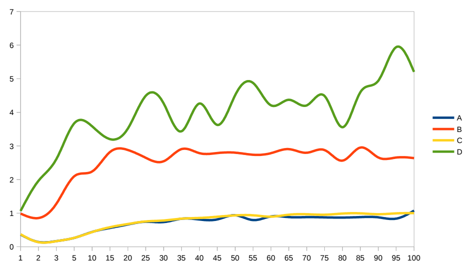


Figure 2.21: MPI parallelization on a cluster, using 4 CPU cores. x axis denotes number of cores, y axis – speedup.

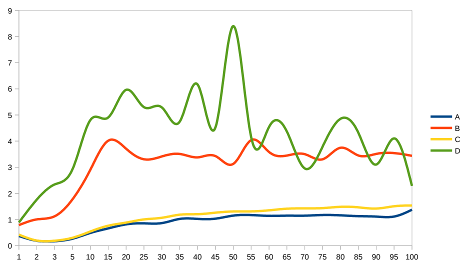


Figure 2.22: MPI parallelization on a cluster, using 16 CPU cores. x axis denotes number of cores, y axis – speedup.

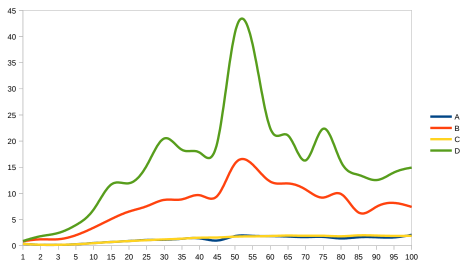
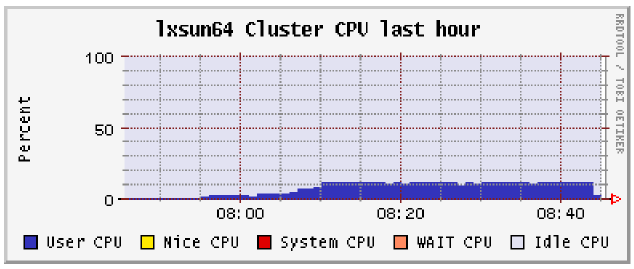


Figure 2.23: MPI parallelization on a cluster, using 64 CPU cores. x axis denotes number of cores, y axis – speedup.



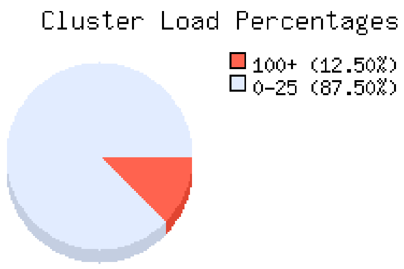


Figure 2.24: Cluster load using multiprocess parallelization, 64 CPUs

Some interesting phenomena can be observed in figure 2.21, then speedup exceeds a number of the nodes being used. This can be explained by non-deterministic nature of the task.

2.3.2 Arm Part Length Evolution

Multi-agent evolving systems were used to simulate the evolution process, observed in nature. Organisms developed in nature with similar length of two main arm parts: femur and tibia. A simple experiment was set up. 2-degrees-of-freedom robotic arm was simulated to reach points in 2D space. 100 agents were created. Each agent had initial random proportions of femur and tibia. After each epoch only 10 best agents survived. A new agent was created by varying the best ones by 10 percents. After 100 epochs femur and tibia lengths converged to the same lengths and the minimum length, required to accomplish a given task (see Figure 2.25).

Multi-agent systems are relatively simple to parallelize. They exploit parallel

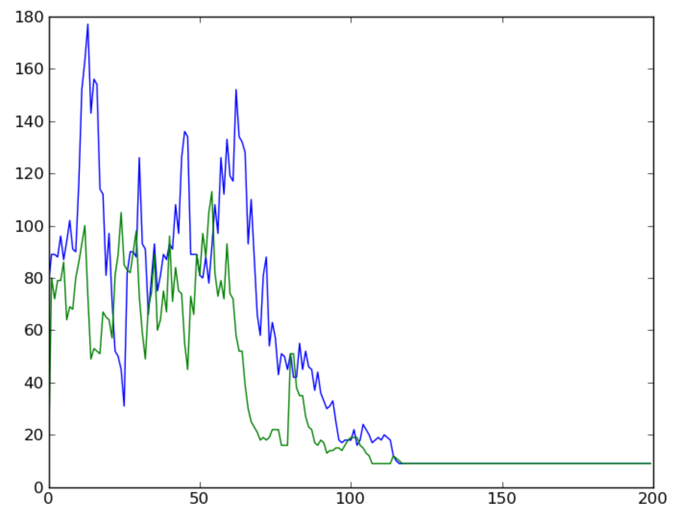


Figure 2.25: Arm part (femur and tibia) lengths. x axis denotes epochs, y axis - length.

processes happening in nature. The architecture of the systems must be taken in account when creating a parallel system. Multi-threading cannot access more CPUs than one node of the system has. To make a system more parallel, different techniques, such as the MPI, must be used. When using the MPI an additional network transfer delay is included. Multi-agent systems can be used to model evolution efficiently.

2.3.3 Experiment Results

Experiment results are displayed in figures 2.8, 2.9, 2.10, 2.11.

Agents are not allowed to die from old age, so the only way, an agent is discarded is when it performs worse than mean error of other agents. New agents are created either by cloning the best agent of the group (figures on the left), or by creating an agent with random initial parameters (figures on the right). This allows analysis of the age of an agent to impact the performance of the agent.

All agents were able to learn first task. Second task was of a similar complexity to the first one, but not all agents were able to adapt to solve it. As displayed in the figures 2.8, 2.9, 2.10, 2.11 the inability to learn, due to saturation of activation function can be observed in all non-linear activation functions.

The initial weights selection of multi-layer perceptrons plays an important role in population performance. The population of agents with linear activation functions were more capable to adapt to the changing task environment. Multi-layer peceptrons using linear activation functions showed better results when new agents were inherited from the best agents, rather when they were initialized with random values. Population of agents with non-linear activation functions benefits from new totally random agents.

Multi-layer perceptrons with the sine activation function failed to retrain at all, though this function was successfully used with single layer perceptrons [82] in other similar research of the author. Multi-layer perceptrons with logistic and hyperbolic tangent functions showed good results. They solved the first task, but when task changed the agent's ability to adapt was subject to randomly selected initial parameters. It must be noted that all these figures are a mean error of 100 simulations. And it is clearly visible, that an error of the single best agent was lower than an error of 1/10 of the best agents. Furthermore, the flat line on the total age of logistic function agents indicates that the population was constantly discarding agents, and trying to find better ones.

2.4 Conclusions

Single layer perceptrons were used to solve the inverse kinematics problem when a task changes. Results confirm learning rapidity theory, discussed in Chapter 1. Less trained perceptron, with non-saturated activation functions adapts to changed tasks. To solve the inverse kinematics problem with more precision, task space must be clustered into smaller regions. Different perceptrons have to be trained for each region. Multi-layer perceptron experiments showed impact of activation function to learning rapidity. While the sigmoid function was chosen for hidden layers, linear activation functions in the output layer prooved to suit the best for the task. The multi-agent system of multi-layer perceptrons was paralelized. The task showed to be non-deterministic and in some cases (e.g. Figure 2.21) the speedup exceeded the processor count by a small margin. In other cases the speedup was achieved close to the processor count, while MPI sollutions were slowed by network transfer delay.

Chapter 3

Practical Experiments with Walking Hexapod Robot

Classical 18-degrees-of-freedom hexapod robot was constructed and used in experiments. To get feedback from legs of the robot piezoelectric force sensors were proposed.

Walking robots show much greater ability to traverse terrain than wheeled or tracked robots. Although wheeled robots have higher speed and simple control methods, walking robots would are preferred for difficult terrain missions such as planetary exploration, underground operations or catastrophic environments [21]. Each of the mentioned situations requires either a rough terrain traversability or a very stable and careful movement. All wheeled robots can only overcome roughness that is smaller than the radius of their wheel. Due to this problem the use of all wheeled (also tracked) platforms are limited. That is why all walking robots has attracted so much attention over the last few years [59].

However, walking robots are still far away from being researched and produced. Problems like high energy consumption, difficult gait selection, complex control and feet placement arise from moving through rough terrain. Each problem must be considered and eliminated before using robots in any real-life situations. In this chapter an multi-layer perceptron based method is proposed for reducing deviations of the legs of the hexapod robot that appear when applying to geometric inverse kinematics method.

Having precise inverse kinematics calculations is very important, because trajectory planning requires inverse kinematics solutions, and it is also important, when executing delicate tasks. Relatively small deviations when calculating the joint space coordinates often cause unacceptable deviations in Cartesian space coordinates.

There are three traditional methods used to solve the inverse kinematics [38]: geometric [22, 41], algebraic [20, 30, 46, 57] and iterative [39] methods. These methods can become complex and time consuming when used in complex systems. Also these methods only give solution to a fixed geometrical configuration of kinematic chain and if kinematics change, for example, a robot's leg is damaged, it is necessary to find a new inverse kinematics solution.

A lot of different intelligent methods (based on Neural Networks, Fuzzy Logic, Reinforcement Learning, etc.) were proposed to solve the inverse kinematics problem of different robotic systems [2, 50, 52, 73]. These methods proved to overcome problems that traditional methods have and also provide a good basis of adaptive kinematic solutions that are not constrained by kinematics of a robot.

There are still many reasons as to why the deviations of feet coordinates or leg trajectory appear. When a number of manipulator degrees of freedom increases, and structural flexibility is included, analytical modeling becomes almost impossible [13]. Even simple real world rigid structures display errors compared to modeled structures. Furthermore, mechanical wear and tear, and structural flexibility adds more errors to the system. After some usage, the system requires recalibration to perform as expected. Analytic methods of inverse kinematics problem solving becomes insufficient.

Another possible source of problem is a feet slippage [49]. Moosavian et. al. have developed a dynamics model for hexapod robot which consists only of narrow pack of equations. This model includes feet interaction with the ground and a force distribution model to find the required friction forces. Even though experimental results showed minimum slippage, it is still obvious that the deviation is not fully avoided.

Several scientists have proposed [19, 70] that leg trajectory errors might occur

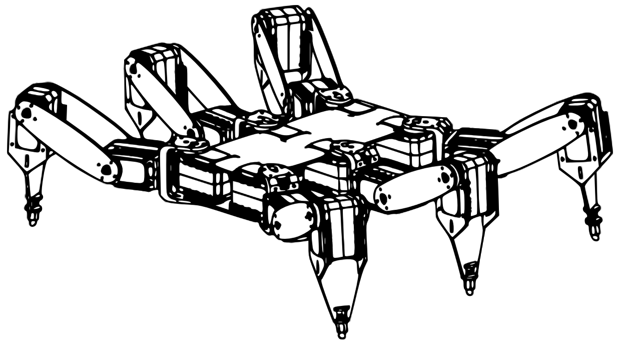


Figure 3.1: Hexapod robot CAD model.

due to a poorly developed inverse kinematics model. Legged robots are very complex systems consisting of a large number of actuators, joints and other elements. The complexity of a kinematic model and control increases with more parts. Therefore, the dynamics model and control algorithms must be carefully organized. Moreover, an accurate control of all actuator is very important. As the speed increases, completing various operations and useful workspace of leg/manipulator reduces [80]. In addition to actuator speeds, frictions and inertias, as well as, some external disturbances sometimes are not know [69]. Having taken all these conditions into account, eliminating feet positioning deviations becomes a very difficult task.

3.1 Hexapod Robot Description and Leg Kinematics

For our experiments a typical hexapod robot with a total of 18-degrees-of-freedom was used. Each leg has three Dynamixel AX-12+ servomotors (giving each leg 3-degrees-of-freedom). AX-12+ actuators support 10 V input voltage and 900 mA maximum current. The working angle is $300°$ with a resolution of $0.35°$. The robot is relatively small: body length L_1 = 160 mm and width L_2 = 90 mm. Leg dimensions are as follows: l_1 = 80 mm, l_2 = 106 mm and l_3 = 68 mm (see Figure 3.2). The total weight of the robot is around 1.5 kg, which consists mostly of eighteen actuators, each of which weighs 55 g.

The analysis of the kinematic model of the robot can be simplified by considering each leg as an individual system [19]. Leg design is very similar to legs of a bug,

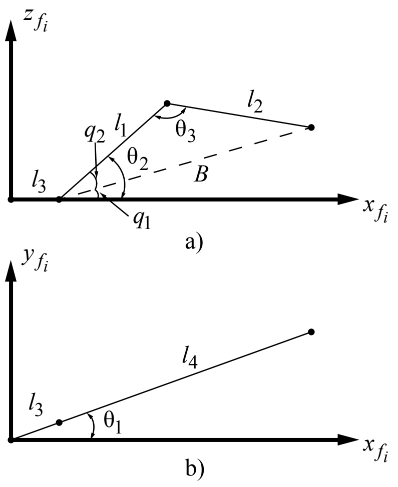


Figure 3.2: Robot leg projections into xy (a) and xz (b) axes

explained in detail in paper [11]. As each leg of the hexapod robot has 3-degrees-of-freedom inverse kinematics can be easily calculated using simple geometrical approach. Two-leg projections are needed in order to derive servomotor angles, projection into xy (Figure 3.2 (a)) and xz (Figure 3.2 (b)) planes.

In a case of forward kinematics, angles of the hexapod robot legs are known, and are $\theta_1, \theta_2, \theta_3$, as are the lengths l_1, l_2, l_3. The coordinates of the endpoint of the leg are marked as x, y and z, and can be calculated as follows:

$$B = \sqrt{l_1^2 + l_2^2 - 2 l_1 l_2 \cos(\theta_3)}, \tag{3.1}$$

$$q_2 = \arccos\left(\frac{l_1^2 - l_2^2 + B^2}{2 l_1 B}\right), \tag{3.2}$$

$$q_1 = Q_2 - q_2, \tag{3.3}$$

$$z = B \sin(q_1), \tag{3.4}$$

$$l_4 = \sqrt{B^2 - z^2} + l_3, \tag{3.5}$$

$$x = l_4 \cos(Q_1), \tag{3.6}$$

$$y = l_4 \sin(Q_1). \tag{3.7}$$

Opposite to forward kinematics, in inverse kinematics coordinates x, y, z of the endpoint of the legs of the hexapod robot are the lengths l_1, l_2 and l_3 of the leg parts. The angles $\theta_1, \theta_2, \theta_3$ are obtained in the following manner:

$$B = \sqrt{x^2 + y^2 + z^2}, \tag{3.8}$$

$$q_1 = \arccos\left(\frac{l_1^2 - l_2^2 + B^2}{2 l_1 B}\right), \tag{3.9}$$

$$q_2 = \arcsin\left(\frac{z}{B}\right), \tag{3.10}$$

$$\theta_1 = \arctan\left(\frac{y}{x + l_3}\right), \tag{3.11}$$

$$\theta_2 = q_1 + q_2 = \arccos\left(\frac{l_1^2 - l_2^2 + B^2}{2 l_1 B}\right) + \arcsin\left(\frac{z}{B}\right), \tag{3.12}$$

$$\theta_3 = \arccos\left(\frac{l_1^2 + l_2^2 - B^2}{2 l_1 l_2}\right), \tag{3.13}$$

where:
θ_1 - angle of coxa servomotor,
θ_2 - angle of femur servomotor,
θ_3 - angle of tibia servomotor,
B - subsidiary imaginary line, connecting the beginning of femur and the end of tibia,
l_1 - length of femur,

l_2 - length of tibia,
l_3 - distance between coxa and femur axes,
l_4 - length of B projection to xy plane.

3.2 Piezoelectric Force Sensors

To get feedback from a robot leg, some kind of a sensor must be used. We propose a piezoelectric force sensor as a feedback sensor from the leg of the robot. There are many different sensors that could be used as robot's force sensing: barometers, pressure sensors, tactile sensors, load cells, silicon based sensors. In this work we chose to upgrade the feet of the hexapod robot with piezoelectric sensors (Figure 3.3). Unlike silicon based sensors, that are relatively small and brake at overload [6], piezoelectric sensors are of the required size and can withstand high pressure.

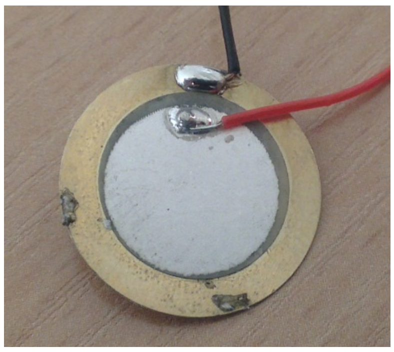


Figure 3.3: Piezoelectric sensor.

Main advantages of these sensors are:

1. Low cost.
2. High resolution (deflections can be of a size of a micrometer).
3. Wide measuring range.
4. Signal can be easily re-produced.

5. High-temperature resistance.
6. Insensitive to external electric and magnetic fields.

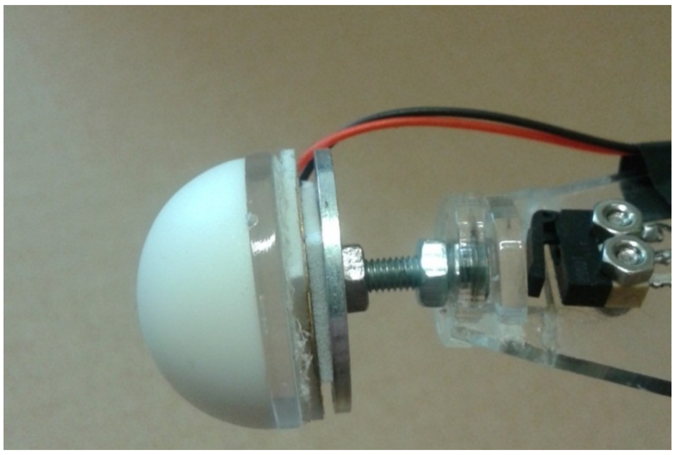


Figure 3.4: Hexapod robot foot with an integrated piezoelectric sensor.

Another reason why we chose to use piezoelectric sensors is that they can be easily applied on the feet of a robot (Figure 3.4). In order not to break these sensors or wires we glued the hemisphere on the bottom and a small metal plate on the top. The whole robot with piezoelectric sensors is shown in Figure 3.4.

Characteristics of piezoelectric sensors were not available, which is why the experiment was made using oscilloscope to obtain the voltage dependence on pressure force $U = f(F)$. We performed the experiment with regard to the parameters of the robot movement, so that the characteristics would be as useful as possible. Results of the experiment are shown in Figure 3.5.

Looking at visible points displayed in Figure 3.5, it appears that voltage dependence on pressure force is linear. Using Origin 8 program, the linear proximation is done to obtain characteristics:

$$U = 1,04F + 4.78. \tag{3.14}$$

There are three main reasons why force sensing is used: to adapt to the roughness of terrain, to equally distribute force between feet [71] and to know the

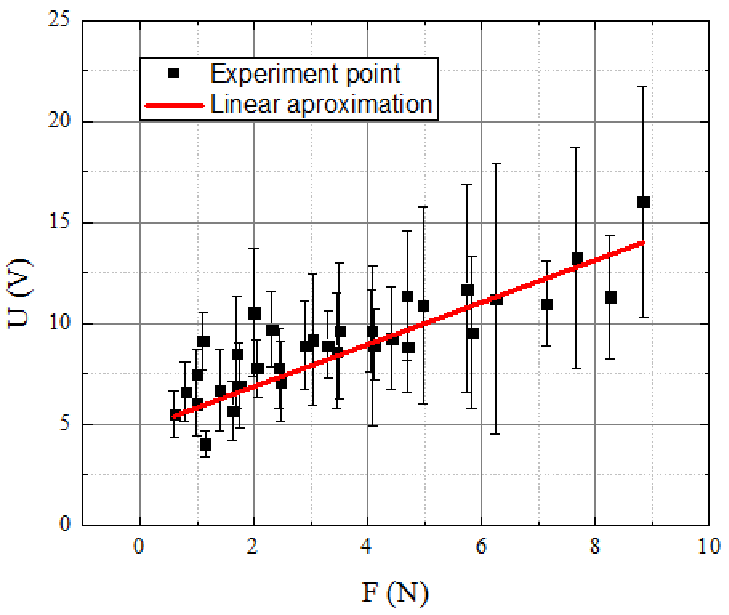


Figure 3.5: Experimental characteristics of piezoelectric sensors $U = f(F)$.

exact moment a foot reaches the ground. In our case it lets us to simulate different types of motions. Otherwise, it is impossible to control the foot pressure against the ground which might damage the robot, the surface or the cargo, if it is fragile. A good example could be an experiment made with robotic foot and its interaction with terrain [16]. The experiment was done using a robotic foot constructed separately from the whole robot. Force distribution was calculated theoretically and tested experimentally, but no sensors were used. Three different categories were distinguished: hard foot on deformable terrain, deformable foot on hard terrain and deformable foot on deformable terrain. Results clearly showed that foot placement deforms either the foot or the terrain depending on the softness of a material. But without force sensors it is impossible to develop a good algorithm that would not allow the robot to press feet hard against the surface. Furthermore, there is no discussion about the possible adaptability to an irregular terrain. Another work concerning rough terrain traversability is described in detail in [9]. Using terrain templates (different height maps under foothold) makes it easier to adapt to a rough terrain that has not been previously seen. Although the results are promising and the robot can successfully overcome rough terrain, sensing or force distribution between feet is not discussed. It still remains unknown, whether or not the robot damages the surface or its

Table 3.1: Leg abbrevations.

Abbrevation	Name of the leg	Comment
RF	Right front	Used during tripod and tetrapod gait experiments.
RM	Right middle	Used in all experiments as additional supporting leg which is always positioned on ground while tansfer legs are swinging motion.
RH	Right hind	Used only during tripod gait experiment.
LF	Left front	Were not used in any of the experiments.
LM	Left middle	Used in all experiments.
LH	Left hind	Were not used in any of the experiments.

own legs. More robust and compliant locomotion was acquired by Buchli J., Kalakrishnan M. et. al. by using force sensors [9]. Experiments were carried out on 3 irregular types of terrain. Results showed that a robot can surpass irregular terrain faster if it has force sensors. Still, no results indicated the actual size of force on each foot.

3.2.1 Piezoelectric Sensors Experiment

For our experiment we used three most common hexapod gaits: tripod, tetrapod and 6 wave gait. In each case, we monitored all transfer legs and one additional support leg using a four channel oscilloscope. Abbreviations of each leg are described in Table 3.1.

It is of great importance to emphasize the RM leg which was used in all cases. By monitoring the additional support leg it becomes possible to distinguish the start and the end of each gait. Moreover, it is much easier to see the exact moment the robot raises and places its legs on the surface.

Hexapod robot's voltage time diagrams for tripod and wave gaits are shown in Figure 3.6. A smoothing filter with polynomial order of 2 to reduce to unwanted noise was added. As we can see, there is an exponential voltage decrease. This is because after the impact, piezoelectric sensors reduce force to zero. In our case, only the voltage peaks matter, and lower cut-off frequency is not important. All voltage and recalculated force peaks are presented in Table 3.2. Given values

Table 3.2: Force peak values for different gaits.

Gait	Tripod				Tetrapod			Wave	
Robot's leg	RF	LM	RH	RM	RF	LM	RM	LM	RM
F, N	0,28	2,75	1,72	2,17	0,38	3,06	3,93	3,78	5,47

were measured during the moment of collision between feet and the ground. It is obvious that when the robot is moving using a tripod gait, legs are pressed against the surface with the least force. This is because the force is distributed among three feet. When using wave gait, robot's legs are pressed with the most force because only one leg is pressed against the surface.

It is also noticeable that voltage time diagrams of the tripod gait have less fluctuations than those of a wave gait. This is mainly because the wave gait has six phases and the tripod gait only has two. In our case, the more phases the gait has, the faster the legs are transferred.

Also, Figure 3.6 shows negative voltage values. This is because piezoelectric sensors show high-pass behavior. It means that after the first impact we have positive values. After the release of pressure, values become negative. This way we can distinguish the moment when the robot raises its legs (negative peaks) and the moment of impact with the ground (positive peaks).

Force distribution dependence on different gaits was observed in this experiment. Tripod, tetrapod and wave gaits were used to monitor the actual size of the force on each foot during the impact with the ground. Experiments were carried out on en ven terrain. Results clearly state that during the tripod gait, the feet of the robot are pressed against the ground with the least force. And during the wave gait, the robot presses its feet with the most force. This is explained as force distribution between transfer feet. Distribution occurs upon the moment of impact with the surface. The more legs are pressed against the ground, the less the force. Having the current results, one of the following solutions should be applied for the hexapod robot to eliminate feet pressure:

- Special leg placement algorithm could be developed to slower leg speed before the moment of collision with the surface.

- The program of the robot could be upgraded with force indicator, which would not allow the robot to press feet harder than the given force. This method requires different type of force sensors.

Piezoelectric sensors are used only to monitor the impact forces. After the impact, the force is exponentially reduced to zero again. It is only possible to observe the moment of collision. Furthermore, when working on piezoelectric characteristics, we encountered a number of additional problems. Verification of the appropriate voltage dependence on force is a matter of another topic, due to complexity of the experiment and dependence on various parameters (temperature, point of impact, acceleration). Future aim is to monitor force distribution for a longer period of time. For that we intend to build unique pressure sensors for the hexapod robot. It will also be possible to observe the force and energy consumption dependence with different loads. This will establish information about the size of the maximum weight a robot can carry, and thus determined what real world operations it can be used in.

3.2.2 Experiment Results

Hexapod robot's voltage time diagrams for tripod and wave gaits are shown in Figure 6. A smoothing filter was added with polynomial order of 2 to reduce unwanted noise. As we can see, there is an exponential voltage decrease. This is because after the impact piezoelectric sensors reduce the force to zero. In our case, only the voltage peaks matter, so lower cut-off frequency is not important. All voltage and recalculated force peaks are presented in Table 2. Given values were measured upon the moment of collision between a feet and the ground. It is obvious that when the robot is moving using the tripod gait, legs are pressed against the surface with the least force. This is because the force is distributed between three feet. And when using a wave gait, robot legs are pressed with the most force because only one leg is pressed against the surface.

It is also noticeable that the voltage time diagrams of the tripod gait have less fluctuations than those of wave gait. This is mainly because the wave gait has six phases and tripod gait has only two. In our case, the more phases the gait has, the faster the legs are transferred.

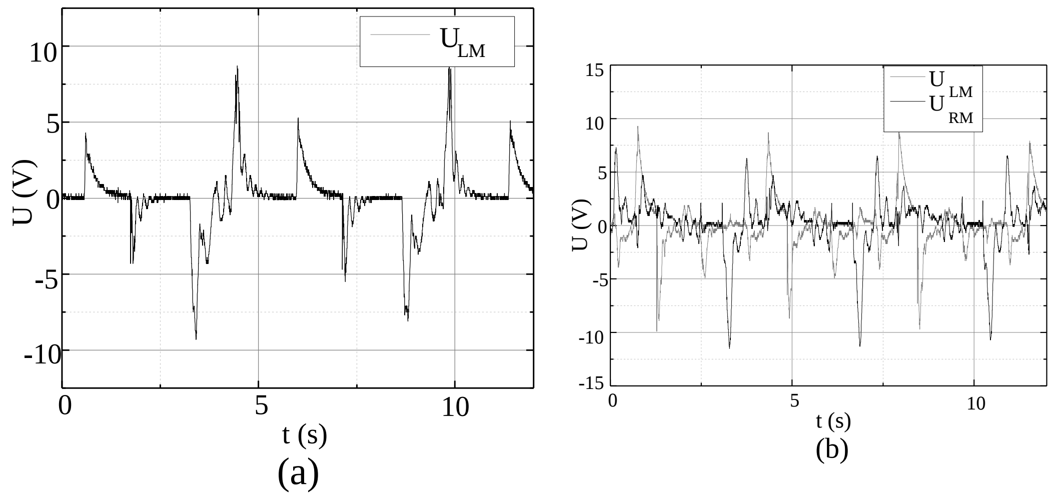


Figure 3.6: Voltage time diagrams for: (a) tripod gait, and (b) wave gait.

Figure 3.6 also shows negative voltage values. This is because piezoelectric sensors show a high-pass behavior. It means that after the first impact we have positive values. After the release of pressure the values become negative. This way we can distinguish the moment the robot raises its legs (negative peaks) and the moment of impact with the ground (positive peaks).

Force distribution dependence on different gaits was observed in this paper. Tripod, tetrapod and wave gaits were used to monitor the actual size of the force on each foot during the impact with the ground. Experiments were carried out on even terrain. Results clearly state that during the tripod gait, feet of the robot are pressed against the ground with the least force. And during the wave gait, the robot presses its feet with the most force. This can be clarified as force distribution between the transfer feet. Distribution occurs during the moment of impact with the surface. The more legs are pressed against the ground, the less the force. Having the current results, one of the following solutions should be applied for the hexapod robot to eliminate feet pressure:

1. Special leg placement algorithm could be developed to slower the leg speed before the moment of collision with the surface.
2. The program of the robot could be upgraded with a force indicator, which

would not allow the robot to press feet harder than the given force. Although this method requires different type of force sensors.

Piezoelectric sensors are used only to monitor the impact force. After the impact, the force is exponentially reduced to zero again. It is only possible to observe the moment of collision. Furthermore, when working on piezoelectric characteristics, we encountered a number of additional problems. To verify the appropriate voltage dependence on the force is a matter of another topic due to the complexity of experiment and the dependence on various parameters (temperature, point of impact, acceleration).

The future aim is to monitor force distribution for a longer period of time. For that we intend to build unique pressure sensors for the hexapod robot. It will also be possible to observe the force and energy consumption dependence with different loads. This will give information about the size of the maximum weight a robot can carry, and determine what real world operations it can be used in.

3.3 Inverse Kinematics Correction by Multi-layer Perceptron Experiment

In order to test the foot's positioning accuracy and evaluate positioning errors only one leg of the robot was used in experiments. The leg was connected to the computer via USB to UART converter (Figure 3.7). As no feedback from servomotors was used, only the Tx line was used to transmit commands. Throughout the experiments the servo motor speed was always set to 5 RPM. This ensured less jerky leg motions. The coordinates of the foot were measured in foot coordinate system in two dimensional space, xy plane. In order to measure the actual foot coordinates, they were marked on a millimeter paper (Figure 3.8) after sending a command for a leg to position it to a certain coordinates. This way we were able to measure actual coordinates that the foot was positioned at. By comparing the desired coordinates that were sent to the leg and the actual measured coordinates we were able to evaluate foot coordinate deviations and positioning errors.

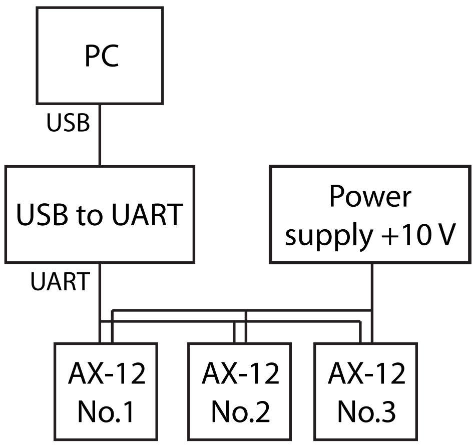


Figure 3.7: Diagram showing robot's leg connection to a PC.

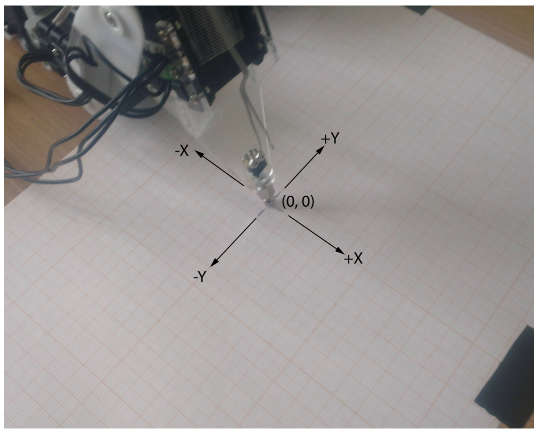


Figure 3.8: Experiment setup with shown initial foot position (0, 0) and positive, and negative foot movement directions along x and y axes (+X, -X, +Y, -Y).

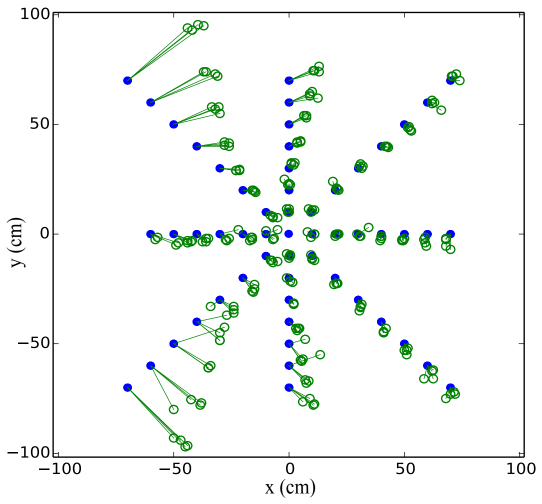


Figure 3.9: Initial experiment results that show foot positioning deviations in different leg's workspace areas.

In order to measure foot positioning deviations the following experiment was done. The foot of the robot was positioned to 55 different positions in its workspace. 27 of these positions were along x and y axes, while one of the coordinates remain equal to 0. Other 28 positions were along diagonal axes, when both coordinates x and y change (Figure 3.9 blue dots). The leg was positioned to each point 5 times. Measurement results are shown in Figure 3.9 as green dots. This gave the general information on what the positioning deviations are in different workspace areas. It can be noticed that along the positive x axis (when $y = 0$) and along diagonal axes in positive x axes (I^{st} and IV^{th} quadrants) deviations are very small. Deviantions increased a bit along the negative x axis (when $y = 0$) and along y axis in both positive and negative directions. Deviations are largest in diagonal directions in negative x axes side (II^{nd} and III^{rd} quadrants). The most important thing here is that as required position coordiantes increase, deviations also increase but only in the negative x axis side and slightly in both y axis sides. This suggests that there may be problems for leg movement in the negative direction that may be caused by the fact that this direction is towards the body of the robot and the position is getting close to mechanical limits of the leg.

Problem may arise from coxa's servo because it is responsible for the motion of the leg along y axis. One of the explanations may be that the PID controller configuration may be wrong for the said servo or even all three servos. If this is the case, each servo controller must be configured which may be a time consuming and tedious work. This also shows that even simple geometric inverse kinematic solution for 3-degrees-of-freedom leg may have problems. In this paper we focus only on eliminating deviations that may appear for any known or unknown reason with neural network. Nevertheless, the mentioned problems that potentially influence these deviations are interesting topics for future research.

Multi-layer Perceptron was chosen in order to compensate errors that occur while positioning the foot of the robot. To successfully solve this problem, a suitable multi-layer perceptron was constructed (as described in section 3.3.2). Training and testing data was gathered from a physical robot (as described in section 3.3.2). Additional features were extracted, by transforming points in the Cartesian coordinate space to polar coordinate space. The data was normalized, as described in section 3.3.1. To verify the results experiments were carried out with a physical robot model.

3.3.1 Data Preparation

The desired foot position coordinates (x, y) were used as input parameters of the multi-layer perceptron. The multi-layer perceptron output parameters are calculated as joint angles $(\theta_1, \theta_2, \theta_3)$ needed to reach this desired position (x, y). After commanding the leg of the robot to reach the desired position, foot coordinates were measured as described in Section 3.3. To increase accuracy, additional features were constructed, by transforming the desired position in the Cartesian coordinate space (x, y) to the polar coordinate space (r, φ). Polar coordinates are interpreted in a more natural way, as it represents the distance and the angle of the desired position. Experiments showed, that by using both the Cartesian and the polar coordinates as input parameters the MSE converges to 0.1: only Cartesian 0.57 MSE, and only polar 1.42 MSE. The learning speed was also improved significantly, meaning that less iterations were needed for the multi-layer perceptron to converge. The construction of additional features from existing data is widely used in neural networks, e.g. using polynomial inputs. Dataset of 4 input vectors $X = (x, y, r, \varphi)$ and 3 output vectors $Y = (\theta_1, \theta_2, \theta_3)$ was constructed. Finally, each input and output vector was normalized by standard normalization (equation 3.15, where x denotes vector is normalized, μ - mean, σ - standard deviation of dataset, and z - normalized vector).

$$z = \frac{x - \mu}{\sigma}. \tag{3.15}$$

3.3.2 Constructing Multi-layer Perceptron

It is well known, the multi-layer perceptron exists with one hidden layer that is capable of estimating an arbitrary non-linear function with any desired accuracy [12, 32]. This is valid for training data. Our experiment also confirmed this, as a training error was close to the mechanical accuracy of the robot.

The multi-layer perceptron was constructed with 4 inputs, 1 hidden layer with 6 nodes and 3 outputs (Figure 3.10) (as described in section 3.3.1). The number of nodes in a hidden layer was determined by an experiment (Figure 3.12). The mean squared error was measured by the training data: running multi-layer

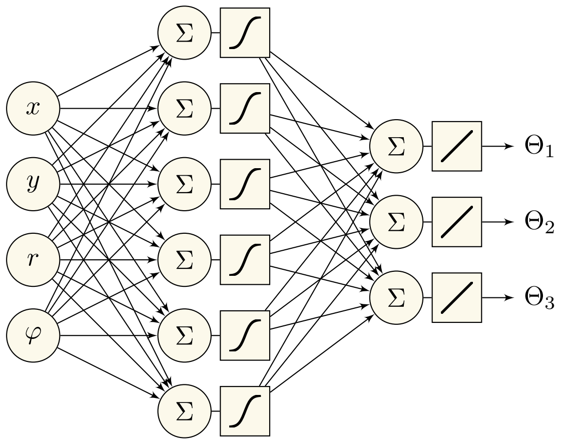


Figure 3.10: The multi-layer perceptron architecture. 4 inputs (x, y, r, φ) represent coordinates in the Cartesian and the polar coordinate space, 3 outputs $(\theta_1, \theta_2, \theta_3)$ represent 3 joint angles. 6 nodes in a hidden layer with sigmoid activation functions, and 3 output nodes with linear activation functions are used.

perceptron 100 times with different random weights from interval $[-0.2; 0.2]$. The standard deviation of 100 results was also measured to evaluate the stability of multi-layer perceptron configuration.

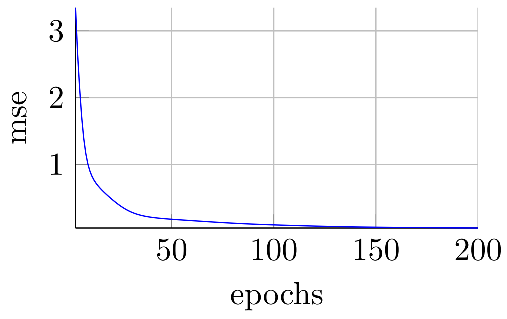


Figure 3.11: MLP training error.

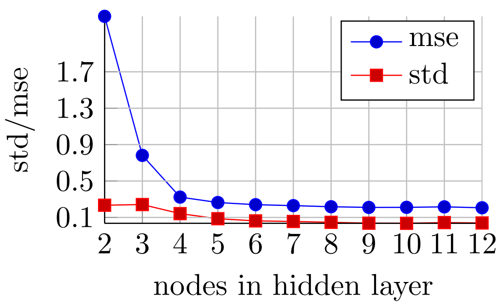


Figure 3.12: Nodes count in the hidden layer of multi-layer perceptron impact on performance

The multi-layer perceptron was trained for 200 epochs (Figure 3.11), with a learning rate (η) of a value of 0.2 and a momentum factor (α) of 0.4. These values were chosen by means of experiment and were not critical. The multi-layer perceptron was continuously able to produce similar results (Figure 3.12). By increasing the epochs count, smaller errors could be achieved in the training data, but the testing data was barely affected, especially having in mind, that the hardware of the robot only accepts integer numbers as joint position parameters. Further increase of accuracy in training data was not needed.

Sometimes using sigmoid, stepper, soft-max or a similar activation function in output layer can be beneficial (e.g. to force the output values to normalize between 0 and 1), but the linear activation function was prefered in this experiment, because the continuous output was needed and data normalization was done separately. Sigmoid activation function was used in the hidden layer of the multi-layer perceptron.

Multi-layer perceptron was trained using a delta rule (equations 3.16 and 3.17).

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n), \tag{3.16}$$

$$\delta_j(n) = e_j(n) \varphi'_j(v_j(n)), \tag{3.17}$$

$$\varphi_j(v_j(n)) = \frac{1}{1 + exp(-v_j(n))}. \tag{3.18}$$

3.3.3 Experiment Results

After training the multi-layer perceptron testing was performed on the physical robot. Deviations (D) were measured as distance (in mm) from the desired position ($coord_{desired}$) to the actual position ($coord_{actual}$): $D = |coord_{desired} - coord_{actual}|$, where $coord$ is either x or y coordinate. For this experiment two positioning squares were used: one ini a positive quadrant (Ist) and one in a negative quadrant (IIIrd) (see Figure 3.13). Square in the Ist quadrant represents the least inaccurate foot positions, as results of previous experiments showed minimum foot errors in this zone. While square in the IIIrd quadrant represents the least accurate zone (see Figure 3.9). Only these two quadrants were investigated because they represent data inaccuracies in other two quadrants. That means that quadrants I and IV are symmetric and quadrants II and III are symmetric in

Table 3.3: Experimental result comparison

	Expr. 1		Expr. 2	
	x, mm	y, mm	x, mm	y, mm
Analytic method deviations	2.09	1.53	12.4	19.6
Error	±0.21	±0.29	±0.17	±0.41
MLP deviations	1.13	1.34	0.919	1.28
Error	±0.16	±0.24	±0.15	±0.28

accordance with positioning accuracy. So using all four quadrants gives redundant information. For testing inverse kinematics only robot was programmed to move its foot to certain positions, depicted as green points (shown in Figure 3.13). After training the multi-layer perceptron robot was reprogrammed to position its foot to different locations, presented as red points (Figure 3.13).

Testing results can be seen in Figures 3.14 to 3.17. Experimental results with inverse kinematics (expr. 1) are shown in Figures 3.14 and 3.16. It is clear that the foot's position deviates from the desired position more in the negative space (the IIIrd quadrant) than in the positive one. This also proves our earlier presumption. In the positive space the average deviation is 2.09 mm for x coordinate and 1.53 mm for y coordinate and errors are ±0.21 and ±0.29 mm accordingly (Table 3.3). In the negative space the deviations are noticeably larger. Average deviations are 12.4 mm for x axis and 19.6 mm for y axis with errors ±0.17 and ±0.41 mm accordingly (Table 3.3).

Figures 3.15 and 3.17 show experimental results with a trained neural network (expr. 2). It is obvious that deviations decrease drastically and average deviations are 1.13 mm for x axis, 1.34 mm for y axis in positive space and 0.919 mm for x axis, 1.28 mm for y axis in negative space. Average errors are ± 0.16, ± 0.24 mm in the positive and ± 0.15, ± 0.28 mm in the negative spaces accordingly. As deviations with trained MLP remain relatively the same in the positive and the negative spaces it is safe to say that the multi-layer perceptron has learned to compensate unnecessary foot deviations. Measurement errors also decreased although very little and remained almost the same allowing us to say that experimental results are reliable.

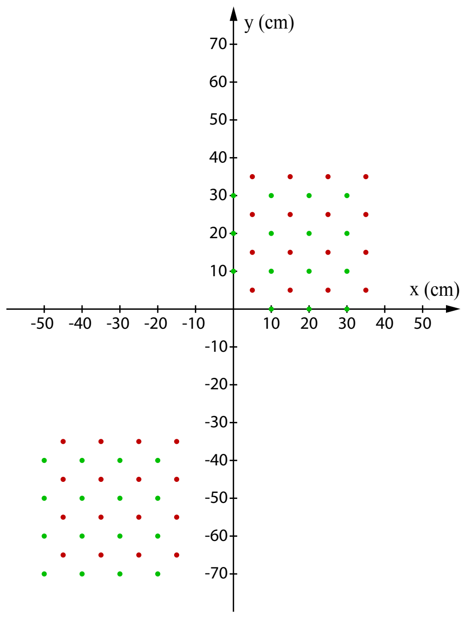


Figure 3.13: Coordinate system with test points for analytical method (green points) and for MLP (red points)

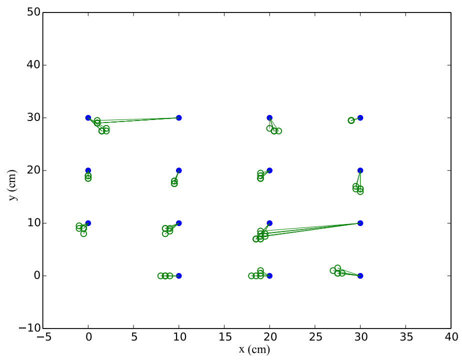


Figure 3.14: Experimental results in the positive xy plane, the I st quadrant, using the geometric inverse kinematics method. Blue dots represent desired positions, green dots – actual measured positions.

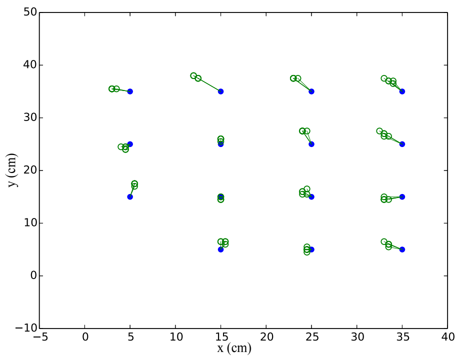


Figure 3.15: Experimental results in the positive xy plane, the I st quadrant, using the multi-layer perceptron method. Blue dots represent desired positions, green dots – actual measured positions.

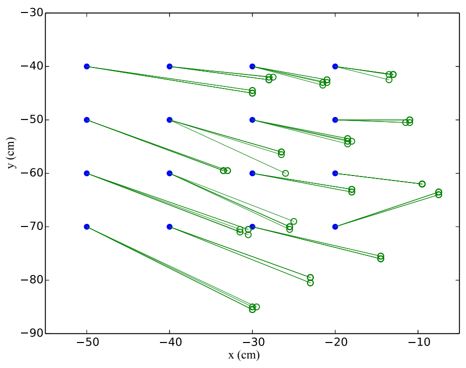


Figure 3.16: Experimental results in the negative xy plane, the IIIrd quadrant, using the geometric inverse kinematics method. Blue dots represent desired positions, green dots – actual measured positions.

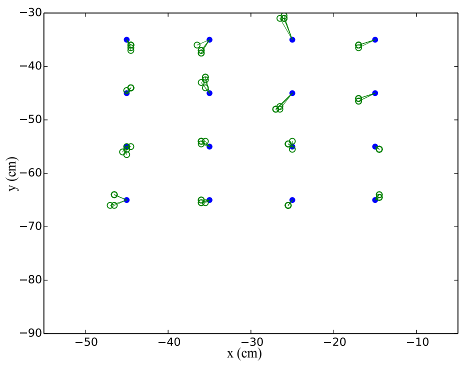


Figure 3.17: Experimental results in the negative xy plane, the IIIrd quadrant, using the multi-layer perceptron method. Blue dots represent desired positions, green dots – actual measured positions.

Table 3.3 shows multi-layer perceptron based inverse kinematics deviations in comparison with analytic method.

Theoretical calculations and experiments with a physical hexapod robot showed, that using multi-layer perceptrons can be beneficial to calculate the inverse kinematics, as analytic methods cannot adapt to inaccuracies and structural flexibility of robot mechanics. Practical experiments showed that the foot of the robot averagely deviates 2.09 ± 0.21 mm on x and 1.53 ± 0.29 mm on y axes in the positive xy plane. And foot averagely deviates 12.4 ± 0.17 mm on x and 19.6 ± 0.41 mm on y axes in negative xy plane. Such deviations appear due to mechanical inaccuracies and using analytic inverse kinematics method for calculating servomotor angles. After calculating the servomotor angles using the neural network, these deviations reduce to: $x = 1.13 \pm 0.16$ mm, $y = 1.34 \pm 0.24$ mm in the positive xy plane and $x = 0.919 \pm 0.15$ mm, $y = 1.28 \pm 0.28$ mm in the negative xy plane.

The multi-layer perceptron configuration was not critical, and showed similar results when enough nodes in the hidden layer were used (Figure 3.12). On the other hand, data normalization was essential, and showed the best results, when each data vector was normalized by standard normalization.

One of the problems with the suggested method is that it is difficult to generate an automated feedback of the foot of the robot. However, this method cannot be used for autonomous robots without external supervisor. So future work would include the development of a system with external foot position supervision to be used as feedback for the multi-layer perceptron. Furthermore, future research should be to determine the cause of the mentioned foot deviations by first changing the configuration of the PID controlers for each servo and determining how a different controller configuration influences foot positioning accuracy. If this would reduced foot deviation we would also compare it to our suggested method.

3.4 Inverse Kinematics Solution by Multi-Layer Perceptron Experiment

The foot positioning accuracy was measured on one leg of the robot. Position commands for this leg were sent from the computer via USB to UART converter. Only transmission line was used, as no feedback from servos was needed. Servo speed was set to 5 RPM to avoid any unnecessary jerky movement which could influence the measurement of accuracy. We assumed that the robot moves on a flat surface (xy plane), so the vertical coordinate was constant with the value of $z = -31$ (the body of the robot is above the surface by 31 mm). To simulate the movement of the hexapod robot leg, $n = 220$ sets of coordination values were generated. By using inverse kinematics defined in section 3.1, the exact values of angles $\theta_1, \theta_2, \theta_3$, matching the chosen coordinates were calculated. The calculated angles were sent to the hexapod leg and the actual coordinates x_r and y_r were measured by marking them on a millimeter paper after the leg was positioned (figure 3.8). This allowed us to evaluate foot deviations. Generated and actual coordinates comparison is shown in figure 3.18.

It is noticeable from Figure 3.18, that the difference between actual coordinates and the calculated ones is larger on the negative x axis side and especially in diagonal $-xy$ and $-x - y$ directions. In order to measure the overall error we calculated the mean square error of calculated coordinates by the following

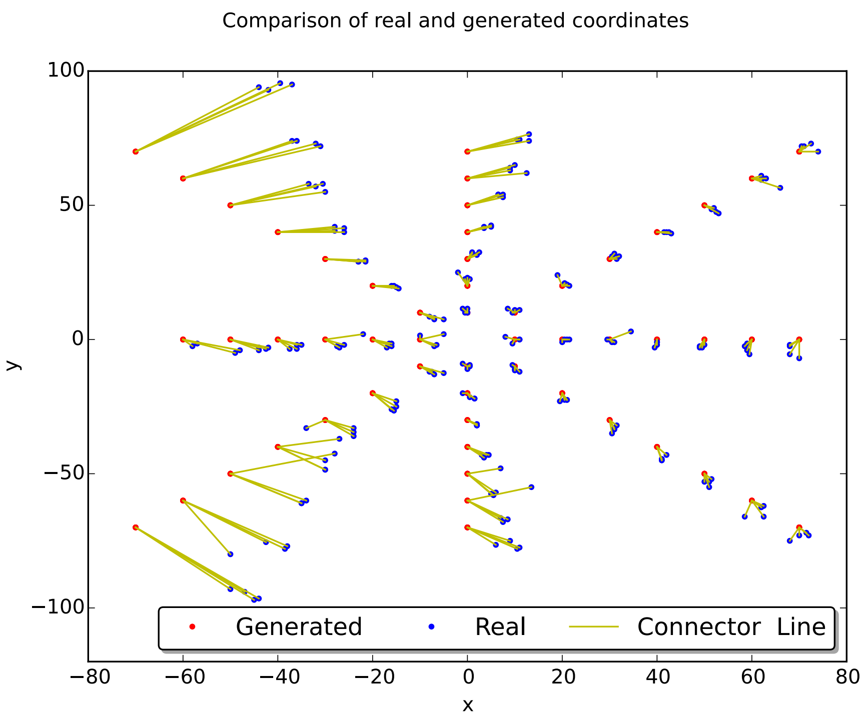


Figure 3.18: The comparison of desired and actual coordinates of hexapod leg.

formula:

$$MSE\left(x,y\right)=\frac{1}{2n}\sum_{i=1}^{n}\left(x_i'-\mathrm{x}_i\right)^2+\left(y_i'-\mathrm{y}_i\right)^2, \tag{3.19}$$

where x_i', y_i' are coordinates we obtained with multi-layer perceptron, and x, y – coordinates we wanted to obtain, and $n = 220$ is the number of generated parameter queues. The mean square error was $MSE(x, y) = 61.17954$.

3.4.1 Experiment Setup

Multi-layer perceptrons were used to solve the problem of a large difference between the calculated and the actual coordinates. The inputs and outputs of two multi-layer perceptrons are displayed in Table 3.4.

In both cases (generated and experimental data) of inverse kinematics calculations multi-layer perceptrons are used with one hidden layer, which uses a

Table 3.4: Input and output for two initial neural networks, where r, β, r_r, β_r are the radius and the angle of the polar coordinate system for generated and actual data, accordingly.

Generated data		Experimental data	
Input	Output	Input	Output
$x, y, r, \beta.$	$\theta_1, \theta_2, \theta_3.$	$x_r, y_r, r_r, \beta_r.$	$\theta_1, \theta_2, \theta_3.$

Table 3.5: Input and output for two improved neural networks.

Generated data		Experimental data	
Input	Output	Input	Output
$x, y, r, \beta, \theta_{1s}, \theta_{2s}, \theta_{3s}.$	$\theta_1, \theta_2, \theta_3.$	$x_r, y_r, r_r, \beta_r, \theta_{1r}, \theta_{2r}, \theta_{3r}.$	$\theta_1, \theta_2, \theta_3.$

sigmoid function for the activation of neurons. The linear function is used for the neurons in the output layer.

By using the multi-layer perceptrons, new angles ($\theta_1 s, \theta_2 s, \theta_3 s$ for the generated data and $\theta_{1r}, \theta_{2r}, \theta_{3r}$ for the actual data) were calculated for the same input parameter values as in Table 3.4. The results are displayed in Table 3a.

3.4.2 Improved Training of Multi-layer Perceptrons

To further improve the obtained results, two multi-layer perceptrons of similar configuration as in Section 3.4.1 were introduced, which take the following inputs and outputs for training:

By using the improved multi-layer perceptron training in a way described in Table 3.5, we obtained new output angles: $\theta^*_{1s}, \theta^*_{2s}, \theta^*_{3s}$ - for the generated data, and $\theta^*_{1r}, \theta^*_{2r}, \theta^*_{3r}$ - for the actual data. The mean square error of the calculated and the actual angles in improved multi-layer perceptrons are displayed in Table 3.6, where N stands for the neurons in the hidden layer, η – training rate, α – momentum term (or the momentum factor), itr – iteration count. Multiple realistically possible variations of N and η were tested, and the best results were achieved with the values presented in Table 3.6.

By training the initial multii-layer perceptron, the obtained mean square error (3.19) was $MSE_{g1} = 0.00107$ for the generated data, and $MSE_{r1} = 0.00367$

Table 3.6: The parameters of MLP training and the corresponding results of training presented in Section 3.4.1 (a) and Section 3.4.2 (b), where c = g (generated), r (actual).

a) MLP training results of inverse kinematics						b) Improved MLP training results				
Data type	N	η	α	itr	$MSE_{c1}(x,y)$	N	η	α	itr	$MSE_{c2}(x,y)$
Generated	11	0.6	0.16	50	0.00107	12	0.3	0.09	500	0.00027
Real	10	0.7	0.05	50	0.00367	11	0.4	0.09	500	0.00231

for the actual data, giving the relative difference of 3.43. For the improved multi-layer perceptron, we obtained $MSE_{g2} = 0.00027$ and $MSE_{r2} = 0.00231$ for the generated and actual data, respectively. The relative difference for the improved multi-layer perceptron is larger – 8.55. One can notice that improved multi-layer perceptrons are giving more accurate results for the generated data ($MSE_{g1}/MSE_{g2} = 3.962$) than the actual one ($MSE_{r1}/MSE_{r2} = 1.588$).

To measure the mean square error of the robotic leg angles, the following formula was used:

$$MSE\left(\theta_j\right) = \frac{1}{n}\sum_{i=1}^{n}\left(\theta'_{i,j} - \theta_{i,j}\right)^2, j = 1, 2, 3, \tag{3.20}$$

where j is angle number, $\theta'_{(i,j)}$ is angle obtained through the multi-layer perceptron, and $\theta_{(i,j)}$ is the angle we wanted to obtain. The mean square errors of both types of data (generated and actual) are displayed in Tables 3.7 and 3.8. As one can see, for both types of multi-layer perceptron and for both types of used data the most accurate angle was θ_2, while the least accurate was – θ_3. It is noticeable, that the best improvement (measured as a relative and an absolute difference) is for the least accurate angle (θ_3).

Forward kinematics with generated data can be solved either by geometric means, or by multi-layer perceptrons. With actual data forward kinematics geometric solution is unknown, and forward kinematics will be solved with multi-layer perceptron in both cases. Both cases uses initial angles (3 inputs) and coordinates to be reached (2 outputs) to teach multi-layer perceptron. Forward kinematics learning parameters and accuracy are displayed in Table 3.6

The multi-layer perceptron was trained for the forward kinematics formula by

Table 3.7: The accuracy of the angles for generated data after the initial multi-layer perceptron calculations (MSE_{g1}) and improved ones (MSE_{g2}) for the generated data

Angle	$MSE_{g1}(\theta_j)$	$MSE_{g2}(\theta_j)$	Difference	
			relative	absolute
θ_1	0.55565	0.16430	3.38192	0.39135
θ_2	0.42421	0.08623	4.91951	0.33798
θ_3	3.08235	0.24977	12.34075	2.83258
Avg	1.35407	0.16677	8.11938	1.1873

Table 3.8: The accuracy of the angles for actual data after the initial multi-layer perceptron calculations (MSE_{r1}) and improved ones (MSE_{r2}) for the actual data.

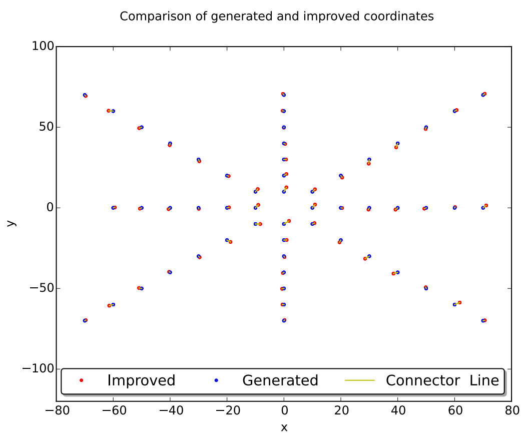
Angle	$MSE_{r1}(\theta_j)$	$MSE_{r2}(\theta_j)$	Difference	
			relative	absolute
θ_1	1.10783	0.96349	1.14980	0.14434
θ_2	1.06909	0.93485	1.14359	0.13424
θ_3	5.45186	3.42236	1.59301	2.0295
Avg	2.54293	1.77357	1.43379	0.76936

obtained angles $\theta_{1t}, \theta_{2t}, \theta 3t$ and new coordinates of x_n and y_n were obtained. The newly obtained coordinates are displayed in Figure 3.19. One can see that there is a large improvement between Figure 3.18 and Figure 3.19b – instead of using inverse kinematics (described in section 3.1) with unacceptably high errors between the obtained and the desired data (see Figure 3.18), multi-layer perceptron usage increases the accuracy of coordinate prediction. This claim is affirmed by results presented in Table 3.10.

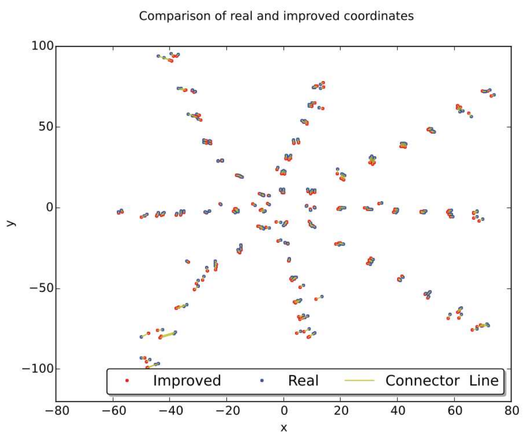
Averages of mean square errors for the initial coordinates and coordinates

Table 3.9: Parameter values of the forward kinematics formula and corresponding accuracy, for initial Neural Network (NN = 1) and for improved Neural Network (NN = 2).

Data type	NN	N	η	α	itr	MSE_{ak}
Calculated	1	8	0.3	0.08	30	0.00002
Real	1	8	0.4	0.01	30	0.00311
Calculated	2	7	0.2	0.13	30	0.00006
Real	2	7	0.7	0.14	30	0.00009



a



b

Figure 3.19: Comparison of initially generated (a) and actual (b) coordinates against the ones obtained with the improved neural network.

Table 3.10: The comparison of initial (MSE) (Figure 3.18) calculations, initial NN ($MSE1$) and improved NN ($MSE2$) coordinate mean square error.

Data type	$MSE(x,y)$	$MSE_1(x,y)$	$MSE_2(x,y)$	Difference	
				relative	absolute
Calculated	-	2.89506	0.88991	-	-0.88991
Real	61.17954	4.84741	1.39515	43.85	59.78439

obtained with the multi-layer perceptron of improved training are displayed in Table 3.10.

3.4.3 Experiment Results

Hexapods robot inverse kinematics were improved by using the multi-layer perceptron. By using the multi-layer perceptron to correct errors of robot's leg deviations, results of the simulation can be increased by approximately 43 times as described in this paper. Mean square error between the desired and the obtained coordinates depends on the training error of the MLP. The accuracy of results decreases by increasing the training error. By using improved multi-layer perceptrons described in Section 3.4.2, the most and the least accurate angles remain the same, θ_2 and θ_3 respectively. The training error directly correlates with the complexity of the formula: a more complex formula leads to the larger

MSEs. Such problem can be solved by using more parameters in training of neural network – it proved to be successful in case of study, as described in Section 3.4.2.

3.5 Conclusions

A walking hexapod (six-foot) robot was constructed to verify theoretical and simulation results. The theory of non-ideal real world robotic system was confirmed, as deviations of inverse kinematics calculation between the analytic model and experiment results were found. A multi-layer perceptron was successfully used to minimize the deviations. Piezoelectric force feedback sensors prooved to be useful to get feedback from legs of the robot and simulate different types of motions. Different walking patterns of a hexapod robot were analysed.

Conclusions

Motion consists of finding the inverse kinematics solution, path finding, trajectory planning and motion execution. A number of statistical, geometrical and iterative algorithms can be used to solve the inverse kinematics problem. Single layer and multi-layer perceptrons were chosen to investigate the inverse kinematics solution in the changing task environment, and their ability to adapt to robotic system changes. Methods to retrain already trained perceptrons were proposed and investigated. Multi-layer perceptrons were used to calculate inverse kinematics of a hexapod robot. A method to improve the hexapod robot's inverse kinematics solution (calculated by analytical means) using multi-layer perceptrons was introduced.

A rapidly changing task environment requires different learning techniques and different tuning of neural networks. The lack of research on neural network controlled robotic systems in the changing task environment was observed. Nature inspired methods to solve the inverse kinematics and trajectory planning problems were chosen to be investigated in depth. Single layer perceptron and multi-layer perceptron algorithms have an advantage, as by using different learning techniques speed can be traded of to accuracy. Multi-agent systems were created to simulate the changing task environment. Analytic methods are not able to adapt to environmental and robotic system changes (e.g. a broken leg, or uncalibrated servos). Multi-agent systems were used to mimic the natural evolution. Parameters of the robotic system (e.g. manipulator geometry) can be improved by evolutionary algorithms.

A method to use motion trajectory primitives was introduced in order to improve the planning of trajectory of the hexapod robot. As experiments with piezoelectric force feedback sensors showed, parabolic trajectories are not sufficient in a different environment.

Piezoelectric force sensors were proposed to improve the hexapod's legs by adding a feedback mechanism. Different kind of motions, observed in nature, can be simulated by the varying feedback amount and the needed inverse kinematics precision. Piezoelectric force sensors offer a cheap and simple solution to add feedback to the system. These sensors have an advantage, as they can determine, if a manipulator is moving down, or up, based on the surface.

A real-life hexapod robot was used to verify these results.

Following conclusions were obtained:

1. A number of techniques potentially perspective to increase the learning speed of neural network used for robot foot movements control and retraining in changing environments were examined both theoretically and by simulation. It was demonstrated while traning single layer and multilayer perceptrons to solve inverse kinematic problem that simpler and less over-trained networks (when the magnitude of weights are small) adapt to the changed task in fewer iterations. Over-trained networks (when activation functions are saturated) fail to adapt. In this case, most effective approach, however, appeared training of the novel network starting from small random initial parameters. Other alternatives are noise injection and/or reduction of targets.

2. Analytic methods are insufficient to solve the inverse kinematics problem in real robotic systems. Multi-layer perceptrons can be used to aid the analytic method in minimizing foot deviations of the hexapod robot (six footed robot). The enhanced analytic method resulted in 2 to 15 times better precision than the analytic solution alone in some cases. Highest increase of precision was achieved when hexapods robot joints were operated at close to maximum angles. These positions caused larger deviations of the analytic solution and were corrcted with multi-layer perceptron.

3. Neural networks require different training to cope with different kind of motions. It was experimentally showed, that piezoelectric force sensors allows to get feedback from robotic systems, to be used with neural network based methods. These sensors can distinguish the moment a robot raises its legs and the moment of their impact with the ground.

4. After constructing the multiagent system of the multilayer perceptrons to solve inverse kinematics problem, speedup up to 40 times was achieved using 64 MPI threads, when solved task was complex. MPI better works with more complex tasks, because when the task is simple the network transfer time becomes similar to the execution time. Simulations demonstrated that the multi-agent system can be used to find the best parameters for the multi-layer perceptron, solving inverse kinetatics problem, by simmulating many different configurations. Experiments with real hexapod robot

showed that multi-agent systems can be used to simulate evolution, by calculating the acceptable configuration of the robotic arm to accomplish the given task.

Future research will address the selection of motion primitives in hexapods and other walking robots. Nature-based methods to minimize energy consumption while learning joint angles speed and acceleration parameters will be investigated. Different types of motions will be integrated in one learning system. Methods will be scaled up to higher-degrees-of-freedom serial manipulators.

Bibliography

[1] Aggogeri, Pellegrini, and Adamini. A fuzzy algorithm to study the inverse kinematics problem of a serial manipulator. volume 783, pages 77–82, 2015.

[2] Alavandar, Srinivasan, and Nigam. Neuro-fuzzy based approach for inverse kinematics solution of industrial robot manipulators. *International Journal of Computers, Communications & Control*, 3(3):224–234, 2008.

[3] Barany, Della-Maggiore, Viswanathan, Cieslak, and Grafton. Feature interactions enable decoding of sensorimotor transformations for goal-directed movement. *The Journal of Neuroscience*, 34(20):6860–6873, 2014.

[4] Bayati. Using cuckoo optimization algorithm and imperialist competitive algorithm to solve inverse kinematics problem for numerical control of robotic manipulators. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 2015.

[5] Becker and Le Cun. Improving the convergence of back-propagation learning with second order methods. In *Proceedings of the 1988 connectionist models summer school*, pages 29–37, 1988.

[6] Beebe, Denton, Radwin, and Webster. A silicon-based tactile sensor for finger-mounted applications. *Biomedical Engineering, IEEE Transactions on*, 45(2):151–159, 1998.

[7] Bingul, Ertunc, and Oysu. Comparison of inverse kinematics solutions using neural network for 6r robot manipulator with offset. In *Computational Intelligence Methods and Applications*, pages 5–pp, 2005.

[8] Biswal, Jha, and Sahu. Inverse kinematic solution of robot manipulator using hybrid neural network. *International Journal of Materials Science and Engineering*, 2015.

[9] Buchli, Kalakrishnan, Mistry, Pastor, and Schaal. Compliant quadruped locomotion over rough terrain. pages 814–820, 2009.

[10] Chandran and Anand. Forward kinematic solution for a five joint robot

using artificial neural network. pages 1704–1709, 2014.

[11] Chen, Ren, Zhang, and Wang. Smooth transition between different gaits of a hexapod robot via a central pattern generators algorithm. *Journal of Intelligent & Robotic Systems*, 67(3-4):255–270, 2012.

[12] Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[13] Daya, Khawandi, and Akoum. Applying neural network architecture for inverse kinematics problem in robotics. *Journal of Software Engineering and Applications*, 3(03):230, 2010.

[14] Deng, Huang, Gao, Zhan, and Zhu. Development of an improved genetic algorithm for resolving inverse kinematics of virtual human's upper limb kinematics chain. pages 189–211. 2014.

[15] Van der Hoek and Wooldridge. Multi-agent systems. *Foundations of Artificial Intelligence*, 3:887–928, 2008.

[16] Ding, Gao, Deng, Song, Liu, Liu, and Iagnemma. Foot–terrain interaction mechanics for legged robots: Modeling and experimental validation. *The International Journal of Robotics Research*, 32(13):1585–1606, 2013.

[17] Doynei, Packard, and Perelson. The immune system, adaptation, and machine learning. *Physica D: Nonlinear Phenomena*, 22(1):187–204, 1986.

[18] Driscoll. Comparison of neural network architectures for the modeling of robot inverse kinematics. In *Southeastcon 2000*, pages 44–51, 2000.

[19] Duan, Chen, Yu, and Liu. Tripod gaits planning and kinematics analysis of a hexapod robot. In *Control and Automation*, pages 1850–1855, 2009.

[20] Duffy. *Analysis of Mechanisms and Robot Manipulators*. 1980.

[21] Ettlin and Bleuler. Rough-terrain robot motion planning based on obstacleness. pages 1–6, 2006.

[22] Featherstone. Position and velocity transformations between robot end-effector coordinates and joint angles. *The International Journal of Robotics Research*, 2(2):35–45, 1983.

[23] Feng, Yao-nan, and Yi-min. Inverse kinematics solution for robot manipulator based on neural network under joint subspace. *International Journal of Computers Communications & Control*, 7(3):459–472, 2014.

[24] Ferber. *multi-agent systems: an Introduction to Distributed Artificial Intelligence*, volume 1. 1999.

[25] Fu and Gonzalez. *Robotics: Control, Sensing, Vision, and Intelligence*. 1987.

[26] Galbraith, Guenther, and Versace. A neural network-based exploratory learning and motor planning system for co-robots. *Frontiers in neurorobotics*, 9, 2015.

[27] Gardner and Dorling. Artificial neural networks (the multilayer perceptron)–a review of applications in the atmospheric sciences. *Atmospheric environment*, 32(14):2627–2636, 1998.

[28] Gasparetto, Boscariol, Lanzutti, and Vidoni. Path planning and trajectory planning algorithms: A general overview. pages 3–27. 2015.

[29] Gentili, Oh, Huang, Katz, Miller, and Reggia. Towards a multi-level neural architecture that unifies self-intended and imitated arm reaching performance. pages 2537–2540, 2014.

[30] Gonzalez and Fu. Robotics: Control, sensing, vision, and intelligence. *Editora Hardcover*, 1987.

[31] Haykin. *Neural Networks and Learning Machines*, volume 3. 2009.

[32] Hornik and White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[33] Huangi, Lai, Keung, Nakamori, and Wang. Forecasting foreign exchange rates with artificial neural networks: a review. *International Journal of Information Technology & Decision Making*, 3(01):145–165, 2004.

[34] Jain, Mao, and Mohiuddin. Artificial neural networks: A tutorial. *IEEE computer*, 29(3):31–44, 1996.

[35] Kang, Chanal, Dai, and Ray. Comparison of numerical and neural network methods for the kinematic modeling of a hybrid structure robot. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 2014.

[36] Kar, Issac, and Jayarajan. Gaits and energetics in terrestrial legged locomotion. *Mechanism and machine theory*, 38(4):355–366, 2003.

[37] Köker. Reliability-based approach to the inverse kinematics solution of robots using elman's networks. *Engineering Applications of Artificial Intelligence*, 18(6):685–693, 2005.

[38] Köker, Öz, Çakar, and Ekiz. A study of neural network based inverse kinematics solution for a three-joint robot. *Robotics and Autonomous Systems*, 49(3):227–234, 2004.

[39] Korein and Badler. Techniques for generating the goal-directed motion of articulated structures. *IEEE Computer Graphics and Applications*, 2(9):71–81,

1982.

[40] Kuncheva. Classifier ensembles for changing environments. In *Multiple Classifier Systems*, pages 1–15. 2004.

[41] Lee. Robot arm kinematics, dynamics, and control. *Computer*, pages 62–80, 1983.

[42] Leshno, Lin, Pinkus, and Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867, 1993.

[43] Lian and Ruey-Jing. Adaptive self-organizing fuzzy sliding-mode radial basis-function neural-network controller for robotic systems. *Industrial Electronics*, 61(3):1493–1503, 2014.

[44] Tomas Luneckas. *Sesiakojo Roboto Judejimo Nelygiu Pavirsiumi Tyrimas*. 2013.

[45] MacQueen. Some methods for classification and analysis of multivariate observations. pages 281–297, 1967.

[46] Manocha and Canny. Efficient inverse kinematics for general 6r manipulators. *Robotics and Automation*, 10(5):648–657, 1994.

[47] McCulloch and Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.

[48] Moody. Economic forecasting: Challenges and neural network solutions. *OHSU Digital Commons*, 1995.

[49] Moosavian, Ali, and Dabiri. Dynamics and planning for stable motion of a hexapod robot. In *Advanced Intelligent Mechatronics*, pages 818–823, 2010.

[50] Morris. Finding the inverse kinematics of manipulator arm using artificial neural network with lookup table. *Robotica*, 15(06):617–625, 1997.

[51] Muaz and Amir. Agent-based computing from multi-agent systems to agent-based models: a visual survey. *Scientometrics*, 89(2):479–499, 2011.

[52] Nanda, Panda, and Subudhi. A novel application of artificial neural network for the solution of inverse kinematics controls of robotic manipulators. *International Journal of Intelligent Systems and Applications*, 4(9):81, 2012.

[53] Narayana, Perugu, and Ramana. Predection of inverse kinematics solution of a puma manipulator using anfis. *Int. J. Adv. Engg. Res. Studies/III/II/Jan.-March*, 84:88, 2014.

[54] Oyama, Agah, MacDorman, Maeda, and Tachi. A modular neural network architecture for inverse kinematics model learning. *Neurocomputing*, 38: 797–805, 2001.

[55] Oyama, Gan, MacDorman, and Tachi. Inverse kinematics learning for robotic arms with fewer degrees of freedom by modular neural network systems. In *Intelligent Robots and Systems*, pages 1791–1798, 2005.

[56] Pachikara, Kehoe, and Lind. A path-parameterization approach using trajectory primitives for 3-dimensional motion planning. In *Proceedings of the 2009 AIAA Guidance, Navigation, and Control Conference*, 2009.

[57] Paul and Shimano. Kinematic control equations for simple manipulators. In *Decision and Control Including the 17th Symposium on Adaptive Processe*, pages 1398–1406, 1979.

[58] Petra and De Silva. Inverse kinematic solutions using artificial neural networks. volume 534, pages 137–143, 2014.

[59] Pongas, Mistry, and Schaal. A robust quadruped walking gait for traversing rough terrain. pages 1474–1479, 2007.

[60] Raj, Raglend, and Anand. Inverse kinematics solution of a five joint robot using feed forward and elman network. In *Circuit, Power and Computing Technologies*, pages 1–5, 2015.

[61] Raudys. An adaptation model for simulation of aging process. *International Journal of Modern Physics*, 13(08):1075–1086, 2002.

[62] Raudys and Amari. Effect of initial values in simple perception. 2:1530–1535, 1998.

[63] Raudys and Mitašiūnas. Multi-agent system approach to react to sudden environmental changes. In *Machine Learning and Data Mining in Pattern Recognition*, pages 810–823. 2007.

[64] Raudys and Žliobaitiė. Prediction of commodity prices in rapidly changing environments. In *Pattern Recognition and Data Mining*, pages 154–163. 2005.

[65] Rescorla. The role of information about the response-outcome relation in instrumental discrimination learning. *Journal of Experimental Psychology: Animal Behavior Processes*, 16(3):262, 1990.

[66] Rescorla and Wagner. A theory of pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement. *Classical conditioning II: Current research and theory*, 2:64–99, 1972.

[67] Ridderström. *Legged Locomotion: Balance, Control and Tools-from Equation to Action*. 2003.

[68] Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386, 1958.

[69] Rossomando, Soria, and Carelli. Sliding mode neuro adaptive control in trajectory tracking for mobile robots. *Journal of Intelligent & Robotic Systems*, 74(3-4):931–944, 2014.

[70] Roy and Pratihar. Kinematics, dynamics and power consumption analyses for turning motion of a six-legged robot. *Journal of Intelligent & Robotic Systems*, 74(3-4):663–688, 2014.

[71] Schmucker, Schneider, Rusin, and Zavgorodniy. Force sensing for walking robots. pages 3–93, 2005.

[72] Schumacher. Multi-agent systems. *Objective Coordination in Multi-Agent System Engineering: Design and Implementation*, pages 9–32, 2001.

[73] Shah, Rattan, and Nakra. Kinematic analysis of 3-dof planer robot using artificial neural network. *IAES International Journal of Robotics and Automation*, 1(3):145–151, 2012.

[74] Shah, Svec, Bertaska, Klinger, Sinisterra, von Ellenrieder, Dhanak, and Gupta. Trajectory planning with adaptive control primitives for autonomous surface vehicles operating in congested civilian traffic. pages 2312–2318, 2014.

[75] Siciliano, Bruno, Khatib, and Oussama. *Springer Handbook of Robotics*. 2008.

[76] Springer, Gattringer, and Stöger. A real-time nearly time-optimal point-to-point trajectory planning method using dynamic movement primitives. pages 1–6, 2014.

[77] Sprunk. Planning motion trajectories for mobile robots using splines. *University of Freiburg*, 2008.

[78] Srikant and Rao. Application of backpropagation neural networks in calculation of robot kinematics. *Case Studies in Intelligent Computing: Achievements and Trends*, page 379, 2014.

[79] Tahami, Jafari, and Fallah. Learning to control the three-link musculoskeletal arm using actor–critic reinforcement learning algorithm during reaching movement. *Biomedical Engineering: Applications, Basis and Communications*, 26(05), 2014.

[80] Tarokh and Zhang. Real-time motion tracking of robot manipulators using adaptive genetic algorithms. *Journal of Intelligent & Robotic Systems*, 74(3-4): 697–708, 2014.

[81] Toshani and Farrokhi. Real-time inverse kinematics of redundant manipulators using neural networks and quadratic programming: A lyapunov-based

approach. *Robotics and Autonomous Systems*, 62(6):766–781, 2014.

[82] Valaitis. Learning motion patterns of robotic arm. *Numerical Computations: Theory and Algorithms*, page 138, 2013.

[83] Vanek, Faigl, and Masri. multi-goal trajectory planning with motion primitives for hexapod walking robot. In *Informatics in Control, Automation and Robotics*, volume 2, pages 599–604, 2014.

[84] Wang and Tso. Path error compensation of a two-link flexible robot arm based on integrated laser transducers. volume 4, pages 3786–3790, 1997.

[85] Weng, McClelland, Pentland, Spornsi, Stockman, Mrigankai, and Thelen. Autonomous mental development by robots and animals. *Science*, 291 (5504):599–600, 2001.

[86] Xu and Wunsch. Survey of clustering algorithms. *Neural Networks, IEEE Transactions on*, pages 645–678, 2005.

Publications by the Author

Periodic

1. T. Luneckas, M. Luneckas, D. Udris, V. Valaitis. Hexapod Robot Foot Deviation Correction Using Multilayer Perceptron. *International Journal of Advanced Robotic Systems.*, pp. 294-297, 2015 [ISI Web of Science].

2. M. Luneckas, T. Luneckas, V. Gavelis, V. Valaitis, D. Udris. Piezoelectric Force Sensors for Hexapod Transportation Platform. *Transport (Special Issue on Smart and Sustainable Transport).*, 30(3), 8 p., 2015 [ISI Web of Science].

Reviewed Conference Publications

5. Š. Raudys, V. Valaitis, Ž. Pabarškaitė, G. Biziulevičienė. A Price we Pay for Inexact Dimensionality Reduction. *Bioinformatics and Biomedical Engineering.*, 289-300, 2015 [ISI Web of Science].

6. V. Valaitis. Learning inverse kinematics problem in changing task environment. *The 12th Scandinavian AI conference*, 257:299-302, 2013 [ISI Web of Science].

7. V. Valaitis. Learning motion patterns of robotic arm. *Proceedings of the international conference "Numerical computations: theory and algorithms"*, 138, 2013.

8. V. Valaitis. Judesiai gamtoje ir dirbtinėse sistemose. *Informacinės technologijos 2012*, 57-60, 2012.

Other publications

9. S. Peldžius, S. Ragaišis, V. Valaitis. Seeking Process Maturity with DSDM Atern. *Computational Science and Techniques*, 2:193-204, 2013 [Periodic].

Conference talks

10. The 3rd IEEE Workshop on Bio-Inspired Signal and Image Processing, 5 May, 2014, Vilnius, Lithuania. V. Valaitis, Multi-agent Neural Network Approach on Inverse Kinematics Problem in Changing Task Environment.
11. The 12th Scandinavian AI conference, 20-22 November, 2013, Aalborg, Denmark. V. Valaitis, Learning inverse kinematics problem in changing task environment.
12. Kompiuterininkų dienos - 2013, 19 - 21 September, 2013, Šiauliai, Lithuania. V.Valaitis, Gamta paremti judesiai dirbtinėse sistemose.
13. Numerical Computations: Theory and Algorithms, 17 - 23 June 2013, Falerna, Italy. V.Valaitis, Learning Motion Patterns of Robotic Arm.
14. Informacinės technologijos 2012, 20 April, 2012, Kaunas, Lithuania. V.Valaitis, Judesiai gamtoje ir dirbtinėse sistemose.

Curriculum Vitae

Vytautas Valaitis was born in Vilnius on 6th of September 1986. He graduated from Vilnius University receiving BSc degree in Software Engineering in 2009 and MSc degree in Informatics in 2011. From 2011 to 2015 he was enrolled into Informatics PhD study program at Vilnius University.

During 2011-2016 academic years he was a lecturer in Faculty of Mathematics and Informatics and prepared lectures for Software Systems Testing and Computer Technics. Served as an assistant for a number of lectures and as a supervisor for a number bachelor and master thesisis.

Since 2006 V. Valaitis has occupied positions of a programmer in several companies.

Since 2012 V. Valaitis participated in 3 projects: High-dimensionality and small data size problems in classification of biomedical and financial data (results of which are used in the thesis), LituanicaSAT-1 - first Lithuanian space satellite, Unpiloted hybrid spacecraft "Kolibris" for defense purpose development.

Vocabulary - Žodynėlis

artificial neural network (ANN)	dirbtinis neuroninis tinklas
data source	duomenų šaltinis
classification	klasifikavimas
clustering	blokinių kūrimas
degree of freedom (DOF)	laisvės laipsnis
forward kinematics	tiesioginė kinematika
hexapod robot	šešiakojis robotas
inverse kinematics	atgalinė kinematika
multi-agent system	daugiaagent ė sistema
multi-layer perceptron (MLP)	daugiasluoksnis perceptronas
parallel computing	lygiagretieji skaičiavimai
single layer perceptron	vienasluoksnis perceptronas
spline	tolydžioji kreivė
supervised learning	mokymas su mokytoju
unsupervised learning	mokymasis be mokytojo

Summary in Lithuanian (Santrauka)

Tiriamojo darbo objektas yra gamta pagrįsti algoritmai roboto judesių planavimui ir valdymui. Patobulintas atgalinės kinematikos skaičiavimo metodas, naudojant vienasluoksnį ir daugiasluoksnį perceptroną. Skaičiavimai buvo pritaikyti greitai besikeičiančioms aplinkos sąlygoms. Rezultatai išbandyti su realiu šešiakoju robotu. Darbe pasiūlytas metodas judesio planavimui remiantis judesio primityvais ir tolydžiosiomis kreivėmis. Pasiūlyta naudoti piezoelektrinius daviklius kaip roboto kojų grįžtamojo ryšio daviklius.

Vytautas Valaitis

ROBOTO JUDESIŲ GERINIMAS NEURONINIAIS TINKLAIS

Daktaro disertacija

Fiziniai mokslai (P000),

Informatika (09 P)

Redaktorė Monika Žemgulytė

Vytautas Valaitis

NEURAL NETWORKS BASED ROBOT MOTION IMPROVEMENT

Doctoral Dissertation

Physical Sciences (P000),

Informatics (09 P)

Editor Monika Žemgulytė