Lappeenrannan teknillinen korkeakoulu
Lappeenranta University of Technology

Asko Rouvinen

USE OF NEURAL NETWORKS IN ROBOT POSITIONING OF LARGE FLEXIBLE REDUNDANT MANIPULATORS

Acta Universitatis
Lappeenrantaensis 88

ISBN 978-952-214-877-3 (PDF)

Lappeenrannan teknillinen korkeakoulu
Lappeenranta University of Technology

Asko Rouvinen

USE OF NEURAL NETWORKS IN ROBOT POSITIONING OF LARGE FLEXIBLE REDUNDANT MANIPULATORS

Thesis for the degree of Doctor of Technology to be presented with due permission for public examination and criticism in the Auditorium of the Student House of the Student Union at Lappeenranta University of Technology, Lappeenranta, Finland on the 6th of October, 1999, at 12 noon

Acta Universitatis
Lappeenrantaensis
88

ISBN 951-764-368-3
ISSN 1456-4491
Lappeenrannan teknillinen korkeakoulu
Monistamo 1999

Asko Rouvinen	Use of neural networks in robot positioning of large flexible redundant manipulators
UDK	681.532.1 : 004.032.26 : 621.865.8
Key words:	hydraulic manipulator, neural networks, deflection compensation, robot positioning, inverse kinematics

ABSTRACT

Deflection compensation of flexible boom structures in robot positioning is usually done using tables containing the magnitude of the deflection with inverse kinematics solutions of a rigid structure. The number of table values increases greatly if the working area of the boom is large and the required positioning accuracy is high. The inverse kinematics problems are very nonlinear, and if the structure is redundant, in some cases it cannot be solved in a closed form. If the structural flexibility of the manipulator arms is taken into account, the problem is almost impossible to solve using analytical methods.

Neural networks offer a possibility to approximate any linear or nonlinear function. This study presents four different methods of using neural networks in the static deflection compensation and inverse kinematics solution of a flexible hydraulically driven manipulator. The training information required for training neural networks is obtained by employing a simulation model that includes elasticity characteristics.

The functionality of the presented methods is tested based on the simulated and measured results of positioning accuracy. The simulated positioning accuracy is tested in 25 separate coordinate points. For each point, the positioning is tested with five different mass loads. The mean positioning error of a manipulator decreased from 31.9 mm to 4.1 mm in the test points. This accuracy enables the use of flexible manipulators in the positioning of larger objects. The measured positioning accuracy is tested in 9 separate points using three different mass loads. The mean positioning error decreased from 10.6 mm to 4.7 mm and the maximum error from 27.5 mm to 11.0 mm.

PREFACE

This study was carried out at the Laboratory of Machine Automation at Lappeenranta University of Technology as a part of the Boom–2000 research project. The project belongs to the Advanced Heavy Machinery technology program funded by the Finnish Technology Development Center (TEKES).

I am indebted to Professor Heikki Handroos who supervised my work. I would also like to thank Professor Aki Mikkola whose always optimistic attitude has been very important, Professor Jouko Lampinen and Professor Heikki Kälviäinen for their help in the neural networks part of the study, the members of the Laboratory of Machine Automation for their help, my family and my motorcycling friends, especially at MC–Club of Kitee, for keeping me motivated.

I also like to thank the Lahja and Lauri Hotinen Foundation, the Foundation of Technology in Finland and the Rector of Lappeenranta University of Technology for their financial support.

Special thanks to Maikki.

Asko Rouvinen

September 1999
Lappeenranta, Finland

CONTENTS

NOMENCLATURE

A	Area of the beam
A^i	Rotation matrix of body i
A^i_4	Homogeneous transformation matrix of body i
B_{di}	Mapping from independent coordinates to generalized coordinate
C	Kinematic constraints of multibody system
C^i	Structural damping matrix of body i
C_q	Jacobian of kinematic constraints
C_{qi}	Jacobian of independent kinematic constraints
C_{qd}	Jacobian of dependent kinematic constraints
C_{di}	Mapping from independent coordinates to dependent coordinates
d	Output of the combined neural networks
d_i	Desired output value of neuron i
d_j	Desired output value of neuron j
d_{pj}	Desired output value of neuron j, related to input–output pair p
e	Vector of network errors
E	Young's modulus
E_b	Sum squared output error calculated in batch mode
E_p	Sum squared output error calculated in patch mode
E_s	Sum squared output error
f	Activation function of a neuron
h_j	Output of hidden neuron j
H	Hessian matrix
$i_{1...3}$	Unit vectors along fixed global axis
$i^i_{1...3}$	Unit vectors along axis of body i
I	Area moment of inertia
I_I	Identity matrix
I_p	Denotes the p–dimensional unit hypercube
J	Jacobian matrix of the network errors
k	All neurons in layers above neuron j
K	Stiffness matrix of multibody system

K^i	Stiffness matrix of flexible body i
m_i	Mass of node point
M	Number of neurons in hidden layer
M^i	Mass matrix of body i
n	Number of generalized coordinates of multibody system
n_b	Number of bodies of multibody system
n_o	The total number of output neurons
N_I	Size of input layer
N_H	Number of neurons in hidden layer
N_O	Size of output layer
N_T	Size of training data set
N_W	Total number of synaptic weights
p	Total number of input–output pairs or number of input nodes
p_b	Force per unit length
q	Total vector of generalized coordinates of multibody system
q^i	Generalized coordinates of body i
q_i	Set of independent generalized coordinates
q_d	Set of dependent generalized coordinates
q_j	Generalized coordinates of multibody system
Q	Vector of generalized forces
Q^i	Generalized forces associated with the generalized coordinates of the body i
Q^j	Generalized forces of multibody system
Q^i_e	Externally applied forces in body i
Q^i_v	Quadratic velocity vector of body i
Q_e	Externally applied forces of multibody system
Q_v	Quadratic velocity vector of multibody system
r^i	Position vector of point P^i in global reference
$r^i_{1...3}$	Components of vector r^i
r^i_4	Position vector of point P^i in global reference using 4x4 transformation
R^i	Position of origin of local coordinate system i in global reference

R_e	Vector including external and stiffness forces of multibody system
$\overline{R}$	Residual force vector
$S1$	Lift cylinder stroke
$S2$	Jib cylinder stroke
$S3$	Extension cylinder stroke
t	Time
T	Total kinetic energy of multibody system
T^i	Kinetic energy of body i
u^i	Position of point P^i in global reference
$u^i_{1...3}$	Components of vector u^i in global reference
$\overline{u}^i$	Position of point P^i in body reference
$\overline{u}^i_{1...3}$	Components of vector $\overline{u}^i$ in body reference
$\overline{u}^i_4$	Position vector of point P^i in body reference using 4x4 transformation
w	Weight matrix
w_b	Bending displacement of a beam
w_i	Neuron weights
w_{ij}	Weight of input neuron j or hidden neuron i
$w^{(1)}_{jk}$	Weight of neuron j for input value k
W^i	Work of externally applied forces acting on body i
W^i_s	Work of forces cause by stiffness element on body i
$X_{1...3}$	Fixed global axis
$X^i_{1...3}$	Body axis of body i
x	Input vector
x_b	Position along x–axis of the beam
x_i	Output value of neuron i or an input
x_j	Output value of neuron j
x_k	Input value of neuron k
y	Output value of a perceptron
y_i	Actual output value of neuron i
y_j	Actual output value of neuron j
y_{pj}	Actual output value of neuron j, related to input–output pair p

0_3	Null vector
a	Moment term
$\overline{\alpha}^i$	Angular acceleration vector of point P^i in body reference
$\overline{\alpha}^i_{1..3}$	Components of vector $\overline{\alpha}^i$
α_l	Coefficient constant
γ	Learning rate
δ_j	Error term for neuron j
$\delta^{(1)}_j$	Error term for hidden neuron j
$\delta^{(2)}_i$	Error term for output neuron i
ε	Fraction of errors permitted in test
Φ	Function
η	Gain term
λ	Lagrange multiplier
μ	Scalar value
ρ	Mass density of a beam
θ	Bias value of a neuron
θ^i	Set of rotational coordinates
θ_{ij}	Bias value of input neuron j or hidden neuron i
$\overline{\omega}^i$	Angular velocity vector of point P^i in body reference
$\overline{\omega}^i_{1..3}$	Components of vector $\overline{\omega}^i$
Δq_i	Vector of Newton differences

1 Introduction

1.1 General

In robot positioning the robot controller calculates reference values for the joint angles or actuator strokes of the robot. These values are calculated using a mapping from the Cartesian space to the joint or actuator space so that for a certain coordinate point in the Cartesian space there is responding joint angle combination that accomplishes the positioning of the end–effector to the point. The mapping is called inverse kinematics. The inverse kinematics problem can usually be solved if the manipulator structure is assumed to be rigid.

The flexibility of the manipulator structure not only causes difficulties in inverse kinematics but also decreases the positioning accuracy. Due to the demand for structural stiffness, the effective mass load of a manipulator, payload, is very small compared to the dead load. If the compensation of the deflection caused by structural flexibility is carried out, the efficiency of manipulators can be increased and lighter structures can be used as manipulators. Deflection compensation can be carried out using tables, which include correction terms for coordinate points.

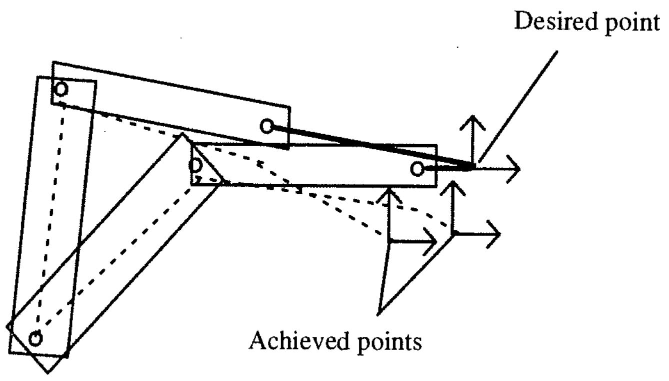


Figure 1. 1 Effect of redundancy and deflection.

The correction term is added to the desired coordinate value and the required actuator strokes or joint angles are calculated by using inverse kinematics of a rigid structure. Redundant and large structures demand very large tables because deflection depends on the positions of the manipulator links. Separate links have different elasticity characteristics, and if a certain coordinate point can be achieved with several link position combinations, the magnitude of the deflection varies. In figure 1.1 the desired coordinate point is theoretically achieved with two separate link position combinations and the dotted lines show schematically how different elasticity characteristics affect the magnitude of deflection.

Since the achieved coordinate point varies depending on the link position combination used in the positioning, the magnitude of the deflection must be declared depending on the actuator strokes or the joint angles and the amount of the mass load of the manipulator.

One solution to the problem is to solve the inverse kinematics of a flexible manipulator. Unfortunately the flexibility causes big problems in direct and inverse kinematics calculations. The modelling of structural flexibility is quite a complicated process and in the solution the knowledge of the affecting forces is required. The dynamic analysis of manipulators can solve the problem but the required computational capacity for real–time applications is great.

1.2 Problem statement and work definition

The deflection and the inverse kinematics of robots are typically highly nonlinear problems. Analytical handling of the inverse kinematics of a redundant mechanism can result in either multiple solutions, degeneration [48], or possibly in no solution at all [5]. The main problem in the degeneration of inverse kinematics is the interdependency of joint angles. If one joint angle can be specified the rest can be defined as functions of that angle, which can lead to several joint angle combinations that accomplish the same position of the end–effector. The analytical solution methods can be divided to two classes: algebraic and geometric methods. The algebraic solution methods are based on link parameters and transformation matrices while geometric solutions try to decompose the spatial geometry of the manipulator into several plane geometry problems [5].

Numerical or optimization methods are often used for solving the inverse kinematics problem but some problems may arise because of the demand of computation time [5], [48]. In [3] several traditional optimization methods are used in solving inverse kinematics of redundant and nonredundant robots. Genetic algorithm is utilized to solve the inverse kinematics of a nonredundant boom structure in [41]. In [36] obstacle avoidance using modified genetic algorithm is considered.

The deflection of a beam can be calculated from a partial differential equation of the deflection curve

$$\frac{\partial^2}{\partial x_b^2}\left[EI(x_b)\frac{\partial^2 w_b(x_b,t)}{\partial x_b^2} \right] + \rho(x_b)\ddot{w}_b(x_b,t) = p_b(x_b,t) \qquad (1)$$

where $EI(x_b)$ is the bending rigidity of the cross section of the beam, $\rho(x_b)$ is the linear mass density of the beam, $w_b(x_b,t)$ is the bending displacement of the beam, and $p_b(x_b,t)$ is the force per unit length. Solving the deflection in its closed form can be difficult because every term in the equation is a variable of the arm length or position. The arms of boom structures may include complicated geometry that affects the bending rigidity of the cross section and the mass density. The force per unit length depends on the position of the arm and the amount and location of mass load.

In [49] is presented a trajectory planning algorithm for large flexible manipulators which takes deflection into account as a correction term in joint positions. The method is based on a nonlinear kinematics model, which involves second order deformation terms. It gives good results but requires lots of computation power in real–time applications. An iterative method based on the use of elasticity matrices is presented in [51]. In [8] an iterative scheme for the end–effector regulation of a flexible robot is presented. The amount of deflection due to gravity is measured using an optical sensor. A method based on the algebra of rotations for motion planning of a flexible manipulator is presented in [53]. The method divides the motion of the robot end–effector to two parts: the motion of the arm that retains the initial deflection and the variation in the deflection due to the change in manipulator configuration. Hiller [16], [44] has studied the modelling, simulation and control of large redundant manipulators with structural flexibility utilizing kinetostatic transmission elements describing mechanical components. The method also includes the coupling with hydraulic actuators.

The use of neural networks offers a possibility to approximate any linear or nonlinear mapping [11], [1]. The use of neural networks in the solution of inverse kinematics of rigid non–redundant manipulators has been considered in [50], [23] and [24]. In these studies the structure under investigation has been a two–link manipulator and either the task space or joint angles have been limited. In [50] neural networks are used to map the manipulator end coordinates from the Cartesian space to joint space. In [23] and [24] neural networks are used for solving the relations of both the positions and velocities from the Cartesian space to the joint space. The functionality of the used methods has been tested using circular test trajectory. Zurada [55] considers the use of neurocontrollers in the forward and inverse kinematics of a rigid two degree of freedom manipulator and the comparison between four different neural network architectures is also presented in the case of forward kinematics. The use of neural networks in the solution of inverse kinematics and trajectory planning of rigid redundant manipulators is studied in [17]. The solution of inverse kinematics of a redundant robot using global regularization is studied in [9]. The structure under investigation in both [9] and [17] has been a three–link plane manipulator. The method presented in [52] uses recurrent neural network in the calculation of the pseudoinverse of the Jacobian matrix. In [17] a Hopfield network is used for solving the inverse kinematics and to optimize the trajectory planning. A sub–neurocontroller is also designed to track the trajectory. The use of neural network in robot positioning by solving the inverse kinematics directly based on the machine vision information is studied in [43] and [47]. The more general use of different types of neural networks as learning controllers for industrial robots is studied in [2]. The use of self–organizing maps in robot positioning utilizing machine vision system and in robot navigation is discussed in [20]. The references show that neural networks are capable of solving the inverse kinematics of rigid manipulators in many different ways accurately enough to be used in robot positioning control.

This thesis shows how neural networks can be utilized in the inverse kinematics solution and deflection compensation of a large redundant flexible manipulator in a static robot positioning situation. The training information for neural networks is obtained using commercial and generally available software as is done in the training and use of neural networks. By using neural networks it is possible to approximate mappings with less computational efficiency than by using the exact solution. The methods presented in this thesis are more suitable for control applications than for solving the deflection as a function. This thesis considers only one particular boom structure because the aim of the work is to clarify the functionality of different neural network architectures.

1.3 Overview of the dissertation

This thesis observes the theoretical background of statics of multibody systems and the theory of neural networks. The first part of chapter 2 presents the basics of the kinematics of rigid systems. A short introduction to frames and transformations needed in kinematics is presented. Generalized coordinates and the kinematic constraint required in the statics of a rigid body are shown. Modelling the structural flexibility using the lumped mass approach is studied. The statics of rigid multibody system are presented. The second part of chapter 2 is dedicated to neural networks. The basic types of neural networks are presented. The most popular neural network, the multi -layer perceptron and the closely related learning algorithms, back–propagation and Levenberg–Marquardt, are presented. Choosing the correct size of a neural network is also considered as well as the solution of the inverse function of a neural network.

Chapter 3 presents the log crane under investigation and the used simulation model of the crane. The use of the model in the calculation of needed training and testing information is discussed.

Simulated results using different methods are shown in chapter 4. It also includes the results measured from an existing structure. Chapter 4 also presents the use of neural networks in the studied methods.

Conclusions about the suitability of different methods in robot positioning of flexible manipulators are drawn in chapter 5. Some discussion about the future research is also included.

1.4 Contribution of the dissertation

The solution to the presented problem is searched by combining existing technologies. The use of simulation models in generation of training information for neural networks is a general method. Neural networks have been used to solve direct and inverse kinematics of robot structures. Most solutions for deflection and inverse kinematics problems have been based on analytical techniques or iterative methods but they demand great computational efficiency.

The original contributions developed in this dissertation are:

1. A method of using a neural network in deflection compensation of large redundant flexible manipulators in robot positioning. The neural network is used together with an existing robot controller system to produce correction terms in order to avoid the effects of deflection due to payload.

2. Methods of using neural networks in the inverse kinematics solution of large redundant flexible manipulators. The methods use neural networks:

a) to model the direct kinematics of the flexible manipulator, and the inverse kinematics are solved by computing the inverse function of the trained network

b) to model the inverse kinematics of a flexible manipulator.

2 Theoretical background

2.1 Theoretical aspects of the kinematics of multibody systems

Kinematics is the study of motion, quite apart from the forces that produce the motion. More particularly, kinematics is the study of position, velocity and accelerations in a system of bodies that make up a mechanism [37]. Multibody systems, figure 2.1, consist of deformable components that are interconnected to each other. The components can be described as rigid or deformable depending on the required modelling accuracy. The connections are done using different types of joints that kinematically constrain the motions of the components with respect to each others.

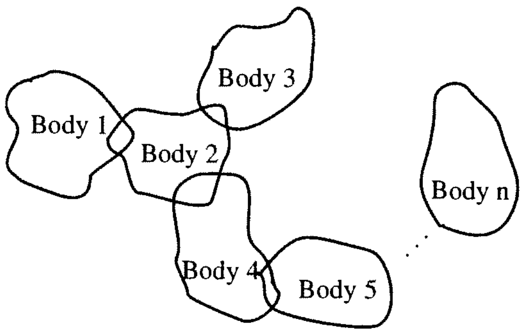


Figure 2.1 Multibody system.

In a rigid body the distance between two of its particles remains constant so there is no difference between the kinematics of the body and the kinematics of its reference coordinate system. The motion of a rigid body in space can be described completely using six coordinates, three translational and three rotational ones [37]. On the other hand two particles on a deformable body can move relative to each other. Consequently one reference coordinate system is not capable of describing the kinematics of a deformable body [46]. The following chapters present the kinematics of rigid multibody systems. The static analysis of a multibody system is also introduced. A short introduction to frames and transformations needed in kinematics modelling is given first, however.

2.1.1 Frames and transformations

The kinematics of a multibody system can be described by using part positions and orientations and their derivatives. These are vector quantities and have to be considered in the proper reference frame or coordinate system. A frame can be described by using three orthogonal axes, which are fixedly connected to the origin point of a reference. There are two types of frames: one that is fixed in time is called global or inertial frame of reference. The other type is called local or body reference and it is attached to each component of the multibody system. In figure 2.2 a body reference and global frame of reference are presented.

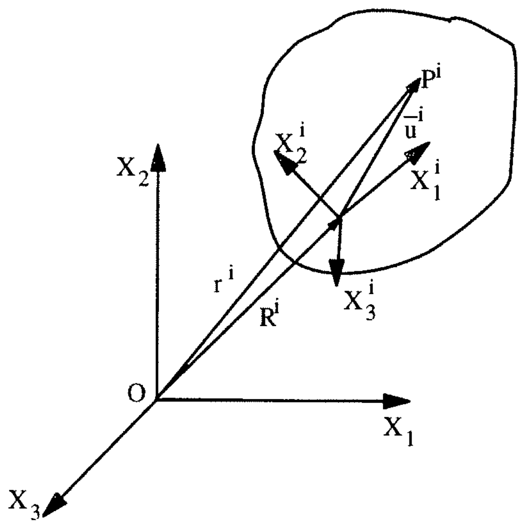


Figure 2.2 Body reference and global frame of reference.

Position vector of the point P^i, can be defined in the body reference as:

$$\bar{u}^i = \bar{u}_1^i i_1^i + \bar{u}_2^i i_2^i + \bar{u}_3^i i_3^i \qquad (2)$$

where i^i_1, i^i_2 and i^i_3 are unit vectors along body axes X^i_1, X^i_2 and X^i_3, and $\bar{u}^i_1$, $\bar{u}^i_2$ and $\bar{u}^i_3$ are the components of the vector $\bar{u}^i$ in the local frame.

In the global reference the same vector can be defined as:

$$u^i = u^i_1 i_1 + u^i_2 i_2 + u^i_3 i_3 \tag{3}$$

where i_1, i_2 and i_3 are unit vectors along fixed global axes X_1, X_2 and X_3, and u^i_1, u^i_2 and u^i_3 are the components of the vector u^i in the global frame.

Hence there are two separate representations for a single point, one in the local reference and the other in the global frame. Because the definition of a point is usually easier to do in local reference and we are mostly interested in positions and orientations in the global frame there is a need to combine these representations. This combination is called mapping [5] and in the case of pure rotation it can be defined as [5], [29], [46]:

$$u^i = A^i \bar{u}^i \tag{4}$$

where A^i is the 3x3 rotation matrix.

The rotation matrix A^i can be defined using several different approaches. The most common are the four Euler parameters that depend on each other, or the Euler angles that describe the rotations relative to the moving system. The Bryant angles [37] consider rotations about axes other than those for the Euler angles, the Rodrigues parameters and the direction cosines that consist of three independent variables [46]. The use of Euler angles, Bryant angles, or Rodrigues parameters may cause singularities at certain body orientations [46]. The singularities are caused by denominators of parameters approaching zero or parameters approaching infinite. Using Euler angles the rotation matrix can have twelve different forms depending on the order and the axis of rotations in the right–hand coordinate system [5].

In the case of rotation and translation the position of the point P^i can be defined as:

$$r^i = R^i + A^i \bar{u}^i \tag{5}$$

where R^i is the position of the origin of the local coordinate system in the global frame, figure 2.2. Using a 4x4–transformation matrix, which includes both rotation and translation, the mapping is:

$$r^i_4 = A^i_4 \bar{u}^i_4 \tag{6}$$

where r^i_4 is $[r^i_1\ r^i_2\ r^i_3\ 1]^T$, $\bar{u}^i_4$ is $[\bar{u}^i_1\ \bar{u}^i_2\ \bar{u}^i_3\ 1]^T$ and the homogeneous transformation matrix A^i_4 is [29], [46]:

$$A_4^i = \begin{bmatrix} A^i & R^i \\ 0_3 & 1 \end{bmatrix} \qquad (7)$$

where 0_3 is the null vector [0 0 0]. The homogenous transformation matrix is frequently used in describing the kinematics of robot systems.

2.1.2 Kinematics of rigid multibody system

The motion of a rigid body in space can be completely described using three Cartesian and three rotational coordinates and their derivatives. In the case of moving body the vectors r^i and R^i are time–variant components, which have to be defined as function of time. Differentiating equation 5 with respect to time and remembering that the components of the position vector of the point in the local coordinate system are constant as a function of time yields:

$$\dot{r}^i = \dot{R}^i + \dot{A}^i \overline{u}^i \qquad (8)$$

where $\dot{r}^i$ is the absolute velocity vector of point P^i and $\dot{R}^i$ is the absolute velocity of the origin of the local coordinate system and

$$\dot{A}^i \overline{u}^i = A^i(\overline{\omega}^i \times \overline{u}^i) \qquad (9)$$

where $\overline{\omega}^i$ is the angular velocity vector $[\overline{\omega}^i_1 \; \overline{\omega}^i_2 \; \overline{\omega}^i_3]$ defined in the local coordinate system. Equation 8 can now be rewritten as:

$$\dot{r}^i = \dot{R}^i + A^i(\overline{\omega}^i \times \overline{u}^i) \qquad (10)$$

Differentiating equation 10 with respect to time leads to the acceleration of a rigid body:

$$\ddot{r}^i = \ddot{R}^i + A^i[\overline{\omega}^i \times (\overline{\omega}^i \times \overline{u}^i)] + A^i(\overline{\alpha}^i \times \overline{u}^i) \qquad (11)$$

where $\dot{\overline{u}}^i = 0$ and $\ddot{\overline{u}}^i = 0$ are used. The term $\ddot{R}^i$ is the absolute acceleration of the origin of the local coordinate system and $\overline{\alpha}^i$ is the angular acceleration vector $[\overline{\alpha}^i_1 \; \overline{\alpha}^i_2 \; \overline{\alpha}^i_3]$ defined in the local coordinate system. The second and third terms are respectively the normal and the tangential components of the acceleration vector [46].

2.1.3 Kinematic constraints

The joints between the separate bodies of a multibody system impose certain conditions on the relative motions between the bodies. From the mathematical point of view mechanical joints require constraint equations which can be written in vector form [37], [46]:

$$C(q_1, q_2, \ldots, q_n, t) = C(q, t) = 0 \tag{12}$$

where q is the set of generalized coordinates defined as:

$$q = [q^1, q^2, \ldots, q^{n_b}] \tag{13}$$

where n_b is the number of the bodies of a multibody system and

$$q^i = [R^{iT} \ \theta^{iT}]^T \tag{14}$$

where R^i is a set of Cartesian coordinates that define the location of the body reference of body i and θ^i is a set of rotational coordinates that define the orientation of the body reference of body i [37], [46].

The generalized coordinates, which must satisfy the constraint equations of the joints, are said to be dependent. Similarly, coordinates that are free to vary arbitrarily are said to be independent. The constraints reduce the number of degrees of freedom in a system. If the set of constraints is dependent on coordinates and time, the constraints are said to be holonomic constraints. If the set of constraints contains inequalities or relations between velocity components that can not be integrated in a closed form, they are said to be nonholonomic constraints [37]. If the time does not appear explicitly, the system is said to be scleronomic [46].

2.2 Structural flexibility in multibody systems

Although it is difficult to provide a generally applicable approach to assigning elastic behavior to a body in the model, it can be said that one body of the mechanism is always elastic. Selecting which bodies of the mechanism are to be assigned flexibility and how many degrees of freedom the mechanism should be assigned to achieve satisfactory results is more or less a question of engineering skill. A flexible body must be modelled using a specific approximation method that reduces the partial differential equation that defines the structural deformation into a set of ordinary differential equations. Finite element [4], [30] and assumed modes [6], [30] are the most commonly used approximation methods [31].

The assumed modes method produces a generalized parameter model of a continuous system that approximates the behavior of the flexible system. The approximation is done using one or more functions of spatial variables and time to describe the deformation of the system. These functions are called admissable functions or shape functions [6].

In the finite element method the deformable body is discretized by dividing it into small regions called elements. The admissable functions are not defined for the entire system but for elements which allows the use of very simple functions [30]. The elements are interconnected by node points. The lumped mass or consistent mass formulation can be used to define the mass of the deformable body.

This study used the lumped mass approach where the total mass of the deformable body is distributed on the node points, which can thus be treated as mass points. The relative motion of the body, the deformation of the body, in respect to its reference motion is defined by the elasticity of the interconnecting elements. The stiffnesses of these elements are defined by matrices, which are described locally. The global stiffness matrix is not needed when using reference coordinate systems which move with the mass points. Figure 2.3 highlights the idealization of the deformable body while using the lumped mass approach.

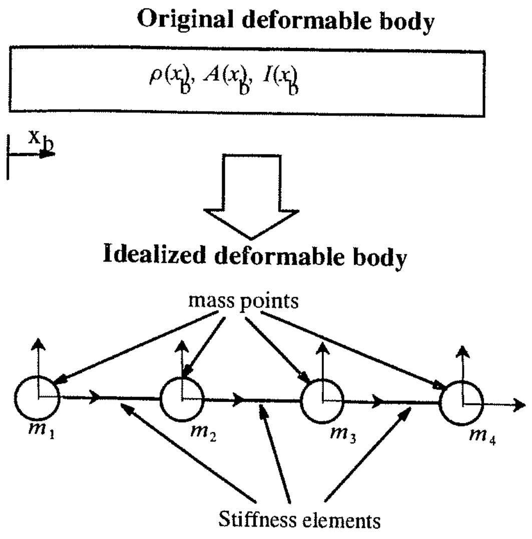


Figure 2.3 The idealization of the deformable body using the lumped mass approach [31].

The response of a deformable body can be determined by solving the dynamics for each mass point. The rigid body equations of motion can be used in the mass points. In this case the forces acting within the stiffness elements need to be taken into account as external forces acting on the mass points. The virtual work of the forces caused by the stiffness elements can be obtained as:

$$\delta W_s^i = -q^{iT} K^i \delta q^i \tag{15}$$

where K^i is the stiffness matrix of the elements to which a mass point is connected. Similarly the damping of a flexible body can be taken into account. The total virtual work of external forces acting on a mass point can be written as:

$$\delta W^i = -\dot{q}^{iT} C^i \delta q^i - q^{iT} K^i \delta q^i + Q_e^i \delta q^i \tag{16}$$

where Q_e^i is the vector of the external forces applied on the rigid body and C^i is the structural damping matrix of an element. Equation 16 can also be expressed in form:

$$\delta W^i = Q^{iT}\delta q^i \tag{17}$$

where Q^i is the vector of generalized forces associated with the generalized coordinates of the body i

$$Q^i = -K^i q^i - C^i \dot{q}^i + Q_e^i \tag{18}$$

In order to achieve a decent computational accuracy a flexible body must be divided to several separate rigid bodies which leads to a large number of generalized coordinates and, unavoidably, to large matrices. For this reason, the computing capacity required for applying this method is large.

2.3 Static analysis of multibody systems

Static equilibrium analysis is used for assigning correct values to the coordinates that describe the state of static equilibrium of the system. This is often used for initializing the positions of the bodies of a system for dynamic analysis. In this study several thousands of static analyses of a rigid multibody system with idealized stiffness properties were used to determine the positions of the flexible bodies of the structure in the static robot positioning situation. The static equilibrium analysis is a special case of the dynamics of a multibody system. The dynamics of a multibody system can be described using, for example, the Lagrangian method, which is an energy–based approach. The use of D'Alembert's principle, virtual work, generalized force and generalized coordinates leads to Lagrangian equation:

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{q}_j}\right) - \frac{\partial T}{\partial q_j} - Q^j = 0, \qquad j = 1,2,\ldots,n \tag{19}$$

where Q^j is the component of the generalized force associated with the coordinate q_j, n is the number of generalized coordinates and T is the total kinetic energy of the system:

$$T = \sum_{i=1}^{n_b} T^i = \sum_{i=1}^{n_b} \frac{1}{2}\dot{q}^{iT} M^i \dot{q}^i \tag{20}$$

where M^i is the mass matrix of body i and q^i is the vector of generalized coordinates of that body. The potential energy of the system, caused by gravitation, is included into the generalized forces vector Q^j. The substitution of the kinetic energy into Lagrangian equation leads to:

$$\frac{d}{dt}(M^i \dot{q}^i) - \frac{\partial T}{\partial q_j} = Q^j \tag{21}$$

Differentiating the first term of equation 21 with respect to time we obtain the equation of unconstrained motion of a rigid body:

$$M^i \ddot{q}^i + \dot{M}^i \dot{q}^i - \frac{\partial T}{\partial q_j} = Q^j \tag{22}$$

Using notation

$$Q_v^i = -\dot{M}^i \dot{q}^i + \frac{\partial T}{\partial q_j} \tag{23}$$

where Q_v^i is the quadratic velocity vector. Taking into account the kinematic constraints by utilizing the Lagrange multipliers in the equation 22 the constrained motion of a rigid body can be expressed in compact matrix form:

$$M\ddot{q} + C_q^T \lambda = Q + Q_v \tag{24}$$

where λ is the vector of Lagrange multipliers and C_q is the constraint Jacobian matrix:

$$C_q = \frac{\partial C}{\partial q} \tag{25}$$

Taking into account equation 18, the equation 24 can be written as:

$$M\ddot{q} + C\dot{q} + Kq + C_q^T \lambda = Q_e + Q_v \tag{26}$$

In a general case, the mass matrix M^i of body i contains elements, the value of which is not zero, outside the diagonal. The non–diagonal elements define the inertial coupling between the translation and rotation of the body and are functions of time. However, the non–diagonal elements of the mass matrix become zero if the body reference is attached to the body center of mass as a consequence of which a simplified system of the equations of the motion can be obtained. The location of the body reference has also another influence. The quadratic velocity vector includes the centrifugal effect caused by the rotation of the body into the body coordinate system. If the body reference and the body center of mass are in the same point the centrifugal forces do not exist and the quadratic velocity vector is zero which also simplifies the equations of the motion of the system.

In the static equilibrium the velocities and accelerations of a rigid body are zero. Using notation

$$R_e = Q_e - Kq \tag{27}$$

where R_e is a vector including the forces acting in stiffness elements and the external forces, the equation 26 for static analysis can be rewritten as:

$$C_q^T\lambda - R_e = 0 \tag{28}$$

For the virtual change in the generalized coordinates of the system, the equation 28 leads to:

$$(C_q^T\lambda - R_e)^T\delta q = 0 \tag{29}$$

The constraint equation 12 for static analysis depends only on the vector of generalized coordinates:

$$C(q) = 0 \tag{30}$$

The vector of generalized coordinates can be written in partitioned form using independent and dependent coordinates:

$$q = [q_i^T \quad q_d^T]^T \tag{31}$$

The independent coordinates are coordinates, the values of which can be integrated from an associated set of differential equations. The dependent coordinates relate to the independent coordinates by kinematic constraints and can be determined using the kinematic relations [46].

Utilizing virtual work and coordinate partitioning in the constraint equation, the virtual change in the dependent system coordinates in terms of the virtual change of the independent system coordinates is:

$$\delta q_d = C_{di}\delta q_i \tag{32}$$

where C_{di} is the matrix

$$C_{di} = -C_{q_d}^{-1}C_{q_i} \tag{33}$$

where C_{q_d} and C_{q_i} relate to dependent and independent constraint Jacobian matrix respectively. The vector of virtual change δq can now be written as:

$$\delta q = \begin{bmatrix} \delta q_i \\ \delta q_d \end{bmatrix} = \begin{bmatrix} I_I \\ C_{di} \end{bmatrix} \delta q_i = B_{di}\delta q_i \tag{34}$$

where I_I is the identity matrix. By substituting the equation 34 into equation 29, the virtual change in the generalized coordinates of the system can now be written as:

$$(C_q^T\lambda - R_e)^T B_{di}\delta q_i = 0 \tag{35}$$

The components of the generalized coordinate vector are linearly independent, which yields to:

$$(C_q^T\lambda - R_e)^T B_{di} = 0 \tag{36}$$

If generalized coordinates are estimated correctly in the static equilibrium, the equation 36 is satisfied. However, in large multibody systems it is difficult to estimate the coordinates correctly and it can be expected that the equation 36 is violated:

$$(C_q^T\lambda - R_e)^T B_{di} = \overline{R}^T \tag{37}$$

It can be shown that the matrix $C_q B_{di}$ is a null matrix so the equation 37 becomes:

$$-R_e^T B_{di} = \overline{R}^T \tag{38}$$

where $\overline{R}$ is called the vector of residual forces, which is related to the independent generalized coordinates. If the static equilibrium in equation 37 is satisfied the value of vector $\overline{R}$ is zero. The roots of the system determine the static equilibrium position of a multibody system. The roots can be solved numerically using, for example, Newton–Raphson method, in which the iterative solution for the following system is sought [46]:

$$\frac{\partial \bar{R}}{\partial q_i} \Delta q_i = -\bar{R} \qquad (39)$$

where the vector Δq_i is the vector of Newton differences.

In figure 2.4 is presented a computational algorithm to solve numerically the Newton differences. The first step is to evaluate the constraint Jacobian matrix, equation 25, using an estimate for the static equilibrium state. Independent and dependent coordinates, equation 31, are identified. In the second step the dependent coordinates are adjusted by solving equation 30 using for example Newton–Raphson algorithm. The third step is to compute the constraint Jacobian matrix and the force vector R_e from equation 27. In the fourth step the residual force vector, equation 38, is solved. The fifth step includes the evaluation of the coefficient matrix, equation 39. Finally the Newton differences are solved from equation 39 and independent coordinates are updated:

$$q_i = q_i + \Delta q_i \qquad (40)$$

If the convergence criteria is satisfied, the dependent coordinates are updated and the new set of generalized coordinates defines the static equilibrium state of the system. Otherwise the algorithm is repeated.

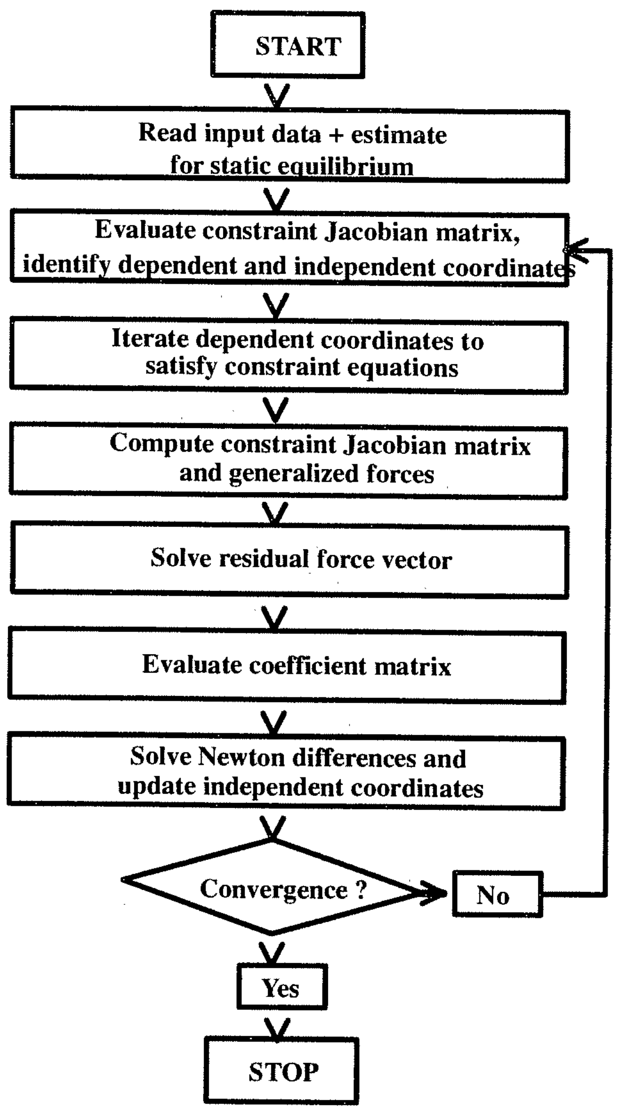

START
Read input data + estimate for static equilibrium
Evaluate constraint Jacobian matrix, identify dependent and independent coordinates
Iterate dependent coordinates to satisfy constraint equations
Compute constraint Jacobian matrix and generalized forces
Solve residual force vector
Evaluate coefficient matrix
Solve Newton differences and update independent coordinates
Convergence ?
No
Yes
STOP

Figure 2.4 *Computational algorithm for the static analysis [46].*

2.4 Theoretical aspects of neural networks

In computing, a neural network refers to a class of models which simulate learning. Neural networks can be used for assisting in detecting information, predicting outcomes, and making decisions. Neural networks offer great computing power due to their parallel structure and the ability to learn and generalize. Neural networks store the learned information with distributed encoding [22]. Typical properties for neural networks are nonlinearity, input–output mapping by learning, adaptivity and the possibility to on– or off–line training [15], [38], [54]. Nonlinearity is an important feature in modelling nonlinear mechanisms [15], [54]. Learning input–output relationships makes it possible to solve classification, prediction, control and diagnosis problems [38]. Adaptivity enables easier retraining of the neural network in the case of changes in the surrounding environment [15]. Computer models and off–line training can be used in modelling large systems which need more training time. The main application fields of neural networks are pattern recognition and diagnostics, inference, modelling and decision support, control and identification of nonlinear systems, optimization and associative memory. The following chapters give an overview to different types of neural networks. The most common neural network, multi–layer perceptron and the closely related back–propagation learning algorithm [15], [28], [38], [42], [54] are introduced. The basic element of a multi–layer perceptron, the perceptron [15], [28], [38] is presented as well as choosing the correct size of a neural network. Finally the solution of the inverse function of a neural network is discussed.

2.4.1 Basic types of neural networks

Neural network models can be divided to three basic types [38]:

- Signal transfer (feedforward) networks
- Competitive learning networks
- Dynamic state transfer (feedback) networks

The Multi–Layer Perceptron (MLP) [15], [28], [38], [54], Radial Basis Function (RBF) network [15], [28], [38], [54] and Principal Component Analysis (PCA) networks [15], [38] are signal transfer networks. The Kohonen Self–Organizing Map (SOM) [18], [38] and Learning Vector Quantizer (LVQ) [15], [19] present the competitive learning network type. The Hopfield network [15], [28], [38] and Boltzmann machine [15], which is a generalization of the Hopfield network, are examples of dynamic state transfer networks.

In supervised learning of for example MLP, each input–output relation is known. The distance between the actual and desired response is utilized as error measurement and is used for adjusting the values of the network parameters. In unsupervised learning of for example SOM, the desired response is not known and explicit error information cannot be used for adjusting the network behavior. The unsupervised learning algorithms must discover the existing patterns, regularities, separating properties, etc. by themselves [22], [55].

The signal transfer networks are often used in classification, modelling complex black box systems and in control and signal processing applications. Hence the MLP is the most popular neural network structure used [21], [38]. Competitive learning networks are suitable for the clustering of input data and for automated feature extraction [38]. Dynamic state transfer networks can be used as associative memories and in solving optimization problems [15], [38].

2.4.2 Multi–Layer Perceptron

The Multi–Layer Perceptron (MLP) network was chosen to be used in modelling the deflection and inverse kinematics of the boom in this study because they can be seen as complicated black box systems. The calculation of the correction term as well as the calculation of the actuator strokes for positioning are clearly control applications. The used input values are continuous and the training information gives a possibility to use supervised learning.

The MLP network consists of layers of parallel perceptrons and hence it is a generalization of the perceptron. The perceptron is the simplest form of a layered feedforward neuron. The input layer of a perceptron is connected only to the unit on the secondary layer where the output can be read. A single perceptron can learn to separate linearly an N–dimensional input space to two half–spaces [38]. It does this by positioning a decision surface in the form of a hyperplane between the two classes. A single–layer perceptron consists of a single neuron with adjustable synaptic weights and a threshold. From figure 2.5 can be seen that the output value of a perceptron depends on the activation function, threshold value and the weighted input value [15].

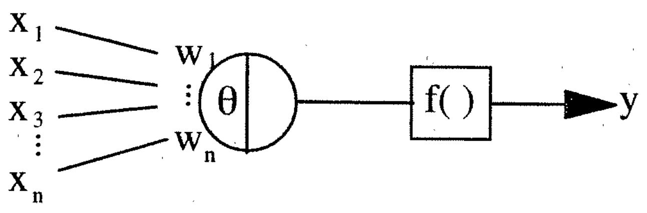


Figure 2.5 *A perceptron [15].*

Generally the output value y of a single perceptron is [15]:

$$y = f(\sum_{i=1}^{n} x_i w_i - \theta) \tag{41}$$

where $f(.)$ is the activation function of a perceptron, θ is the bias value or offset and w is the weight vector

$$w = [w_1 \ w_2 \ \cdots \ w_n]^T \tag{42}$$

and x is the input vector

$$x = [x_1 \ x_2 \ \cdots \ x_n]^T \tag{43}$$

Since the perceptron has the capability to separate the input space to half–spaces it is often used as a classifier. By using more than one neuron in the output layer it is possible to form a classifier with more than two classes. For proper action of the perceptron the classes have to be linearly separable, however. A single perceptron can also be used for performing basic logic operations NOT, OR and AND. Figure 2.6 shows two general, discrete activation functions used in classification problems and in logic operations and the linear activation function, which is used with continuous perceptrons. The linear activation function enables better control over the training procedure and the computation of the error gradient [55].

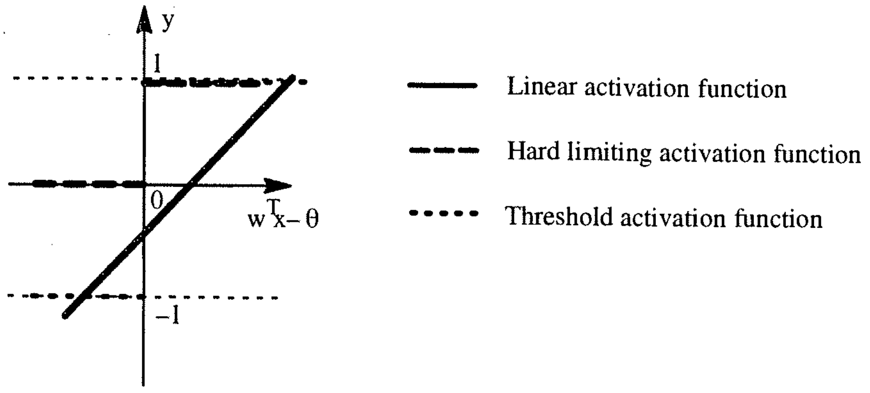


Figure 2.6 Some activation functions.

The output value of a single linear perceptron is:

$$y = w^T x - \theta \qquad (44)$$

The correct values of the weights and the bias for a given problem can be found by training if the solution exists.

By combining different amounts of different types of perceptrons, as figure 2.7 shows, it is possible to achieve desired mapping between the input and output layers of an MLP. The MLP network exhibits a high degree of connectivity. Generally the network structure is fully connected, which means that every perceptron on a certain layer is connected to all perceptrons on the preceding layer. Anyhow, if the designer has background knowledge about the inexpediencies of the parameters of a certain problem, it is possible to include this knowledge into a network by eliminating some connections [54] or constraining the values of weights in certain connections [26], [45]. An MLP network is called feedforward network if it contains no closed loops between neurons. The feedback network has closed loops between neurons or feedback pathways [22].

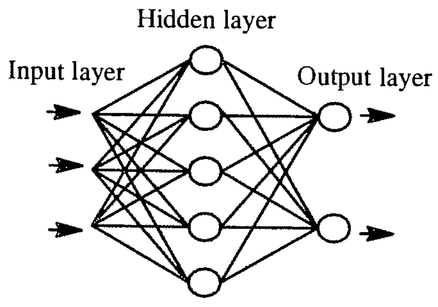


Figure 2.7 *A fully connected feedforward Multi–Layer Perceptron network with one hidden layer.*

The number of perceptron layers and the number of perceptrons in one layer depends on the complexity of the desired representation. A more complicated representation needs more layers and more perceptrons. The number of hidden layers and the number of perceptrons in one layer can freely be selected by the designer. However, the numbers must match the complexity of the problem and the amount of training information. The size of the input layer depends on the number of input parameters. Similarly the number of perceptrons in the output layer depends on the number of output parameters [38].

The activation functions of the perceptrons of an MLP network include nonlinear functions. The important point to emphasize is that the activation function must be differentiable everywhere. The nonlinearity of the activation functions makes the MLP networks differ from a single perceptron. If the MLP network has only linear activation functions its function can be reduced to a single layer perceptron [15].

There are several different kinds of nonlinear activation functions for the perceptron. The most generally used ones are a hyperbolic tangent and a logarithmic activation function, presented in figure 2.8. Also a linear activation function, figure 2.6, is used especially in the perceptrons of the output layer [15].

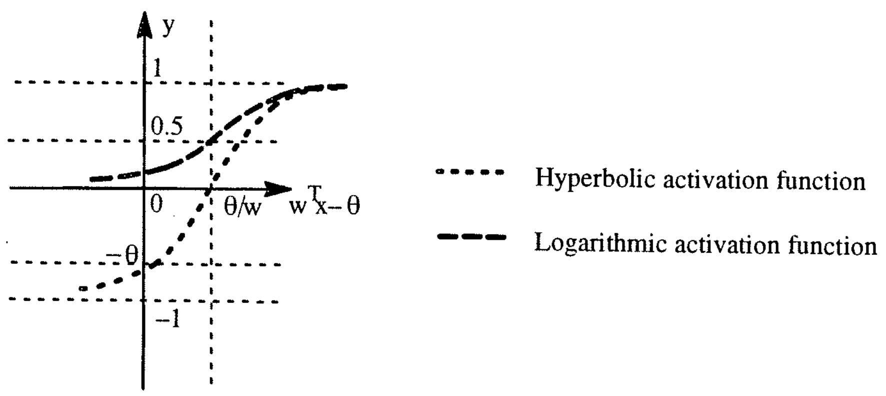


Figure 2.8 Nonlinear activation functions.

The output value of a single perceptron with the hyperbolic tangent activation function is:

$$y = tanh(w^T x - \theta) \tag{45}$$

The output value of a single perceptron with the logarithmic or sigmoidal activation function is:

$$y = \frac{1}{1 + e^{-(w^T x - \theta)}} \tag{46}$$

Besides nonlinearity, hidden layers and connectivity, the ability to learn the desired relations from given information through training enables the high computational capacity of an MLP network. The operation of the MLP network is separation of the number of weights in the network and the effective number of parameters in the network. In the beginning of the training the values of the weights are small and decrease near zero, which produces nearly linear mapping. The linear mapping has only small number of degrees of freedom independently of the complexity of the network. During the learning the values of the weights grow and the operation points of nonlinear perceptrons move to increasingly nonlinear parts of the activation functions. This increases the number of effective parameters in the MLP network [25], [35].

The capability of feedforward multilayer neural networks in approximation of continuous mappings has been studied in [11]. The universal approximation theorem that can directly be applied to multilayer perceptrons is stated as follows [11],[15]:

Let $\Phi(.)$ be a nonconstant, bounded and monotone–increasing continuous function. Let I_p denote the p–dimensional unit hypercube $[0,1]^p$. The space of continuous functions on I_p is denoted by $C(I_p)$. Then given any function f $\in C(I_p)$ and $\varepsilon > 0$, there exists an integer M and sets of real constants a_i, θ_i and w_{ij}, where $i=1,\ldots,M$ and $j=1,\ldots,p$ such that approximate realization of function $f(.)$ can be defined as

$$F(x_1,\ldots,x_p) = \sum_{i=1}^{M} a_i\phi(\sum_{j=1}^{p} w_{ij}x_j - \theta_i) \qquad (47)$$

and

$$|F(x_1,\ldots,x_p) - f(x_1,\ldots,x_p)| < \epsilon \qquad (48)$$

for all $\{x_1, \ldots ,x_p\} \in I_p$.

The logarithmic activation function is a nonconstant, bounded and monotone–increasing function, so it satisfies the criteria for the function $\Phi(.)$. The equation 47 represents the output of a multilayer perceptron described as:

- The network has p input nodes and a single hidden layer with M neurons
- The inputs of the network are $x_1, \ldots ,x_p$
- Hidden neuron i has weights $w_{i1}, \ldots ,w_{ip}$ and threshold θ_i
- The network output is a linear combination of the outputs of the hidden neurons
- $a_i, \ldots ,a_M$ define the coefficients of the combination

This theorem states that a single hidden layer is sufficient for a multilayer perceptron to compute a uniform ε approximation to a given training set of inputs $x_1, \ldots ,x_p$ and desired outputs $f(x_1, \ldots ,x_p)$ [15].

2.4.3 Back–propagation learning algorithm

The use of neural networks includes two phases: the learning phase in which the weight and bias values are estimated and the actual use phase in which the neural network is used for the desired representation. In the learning phase the weight and bias values of neurons are estimated on the basis of training information. The learning phase usually demands large calculation capacity due to a great amount of iterations. There are several different supervised learning algorithms. The most common ones used with MLP networks are the back–propagation algorithm which is based on the gradient method and the Levenberg–Marquardt algorithm [10] that is based on the gradient– and Gauss–Newton–methods. The Levenberg–Marquardt algorithm is more efficient but requires more computer memory. In both algorithms the goal is to minimize the error between training information and the result computed by the neural network.

The back–propagation learning algorithm is a gradient descent method that tries to minimize the error function. The error function is the sum of squared errors between actual and desired output values. In patch or stepwise mode the value of error function E_p is calculated after each input–output pair:

$$E_p = \frac{1}{2}\sum_{j=1}^{n_o}(d_{pj} - y_{pj})^2 \tag{49}$$

where d_{pj} is the desired output of neuron j, considering input–output pair p, y_{pj} is the actual output of neuron j using input–output pair p and n_o is the total number of output neurons. In batch mode the value of error function E_b is calculated as:

$$E_b = \sum_{p} E_p \tag{50}$$

where p is all the input–output pairs.

The back–propagation method requires that the activation function of the perceptrons is continuously derivative [15], [28]. The first step in the method is to set all weight and bias values to small random values. In the next step the output values of the neural network are calculated using all learning values. Then weights and bias values are adjusted to minimize the error according to equations 51–54. The new weight and bias values from hidden neuron *i* or from an input to neuron *j* are calculated as [28]:

$$\begin{aligned} w_{ij}(t+1) &= w_{ij}(t) + \eta\delta_j x_i \\ \theta_{ij}(t+1) &= \theta_{ij}(t) + \eta\delta_j \end{aligned} \tag{51}$$

where η is the gain term, δ_j is the error term for neuron *j* and x_i is either the output of neuron *i* or an input. If neuron *j* is an output neuron, then the error term is calculated as [28]:

$$\delta_j = y_j(1 - y_j)(d_j - y_j) \tag{52}$$

where y_j is the actual output of neuron *j* and d_j is the desired output of neuron *j*. If neuron *j* is an internal hidden neuron, then the error term is calculated as [28]:

$$\delta_j = x_j(1 - x_j)\sum_k \delta_k w_{jk} \tag{53}$$

where *k* is all the neurons in the layers above neuron *j* and x_j is the output of neuron *j*. Sometimes the system converges faster if a moment term is added to equation 51. New weight and bias values are then calculated as equation 54 shows [28]

$$\begin{aligned} w_{ij}(t+1) &= w_{ij}(t) + \eta\delta_j x_i + a(w_{ij}(t) - w_{ij}(t-1)) \\ \theta_{ij}(t+1) &= \theta_{ij}(t) + \eta\delta_j + a(\theta_{ij}(t) - \theta_{ij}(t-1)) \end{aligned} \tag{54}$$

where α is the moment term and $0 < \alpha < 1$.

The updating of the weight and bias values can be done in two basic ways depending on the processing of the training set. In the pattern or stepwise mode the updating is done after each input–output pair and in batch mode the updating is done after the presentation of all the training values. The batch mode enables more accurate estimate for the gradient vector. If the training samples are given in random order in the pattern mode it decreases the probability of trapping to a local minimum [15].

In general, the back–propagation algorithm cannot be shown to converge. Neither is there a well–defined criterion for stopping the training operation. In [15] there are four different criteria presented for the convergence of the back–propagation learning:

- The Euclidean norm of the gradient vector reaches a sufficiently small gradient threshold
- The absolute rate of change in the average squared error is sufficiently small
- The maximum value of the average squared error is equal or less than a sufficiently small error energy threshold or the Euclidean norm of the gradient vector is equal or less than a sufficiently small gradient threshold
- The generalization properties of the network are discovered to be adequate or the generalization performance of the network has peaked

In the latest method the generalization properties of a network are tested by using an independent data set that has not been included in the training set of the concerned iteration round. The test data set can be a random sequence of all training data or a separate data set used only in the performance testing. In [39] an overview of advanced supervised learning methods as well as some benchmark results of studied algorithms can be found.

2.4.4 Levenberg–Marquardt learning algorithm

In Levenberg–Marquardt learning algorithm the main advantage is that it gives second order training speed without computing the Hessian matrix. In the case of error function that has the form of the sum of squares, as in equation 49, the Hessian matrix can be approximated as [10]:

$$H = J^T J \tag{55}$$

where J is the Jacobian matrix. The Jacobian matrix contains the first derivatives of the network errors with respect to weights and bias values. The Jacobian matrix can be computed using standard back–propagation technique. The updated values are calculated as [10]:

$$w(t+1) = w(t) - [J^T J + \mu I_l]^{-1} J^T e \tag{56}$$

where e is the vector of network errors and I_l is the identity matrix. The scalar μ defines the type of the method. If μ is zero the update method is the Gauss–Newton method, if the value of μ is large the update method approximates the gradient descent method with small step size. The Gauss–Newton method is faster and more accurate near an error minimum. The objective is to reduce the value of μ and shift towards the Gauss–Newton method as quickly as possible [10].

2.4.5 Determining the correct size of MLP

Choosing the number of neuron layers and neurons in one layer of MLP correctly for a current problem is a problem itself. It has been shown that a multi–layer neural network with at least one hidden layer whose activation functions are sigmoid functions can approximate any continuous mapping [11], [1]. In classification an advantage can be achieved by using several hidden layers [13], [28]. The operation of an MLP network depends on a large number of free parameters, which are the neuron weights and bias values, offered parameters, and the number of used parameters, chapter 2.4.2. Oja [38] states that the number of neurons in the hidden layer should be approximately

$$N_H \approx \frac{N_T}{5(N_I + N_O)} \tag{57}$$

where N_T is the size of training set, N_I is the size of input layer and N_O is the size of output layer. In reference [15] the relation between the number of hidden neurons N_H and the size of training set N_T is presented as:

$$N_T \geq \frac{32N_W}{\epsilon}\ln(\frac{32N_H}{\epsilon}) \tag{58}$$

where N_W is the total number of synaptic weights in the network and ε is the fraction of errors permitted in the test. An MLP network with too few neurons is not capable of approximating the mapping in the desired way as shown in figure 2.9. This situation is called underfitting [10] or underdetermination [38]. If the MLP network includes too many neurons the mapping might look correct considering the training information. If independent test data not included in the training information are given as input the output is heavily inaccurate as figure 2.10 presents. This situation is called overfitting [10], [15] or overtraining [38].

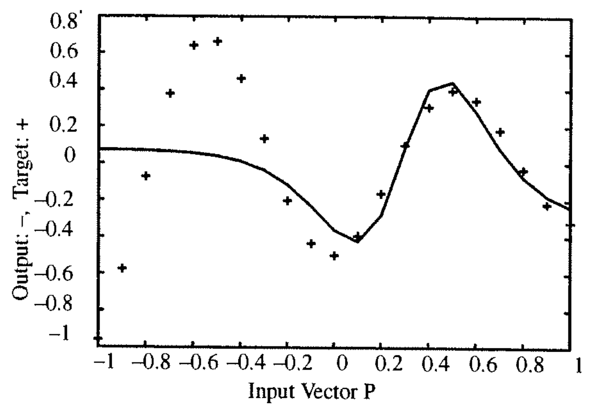


Figure 2.9 Underfitting [10].

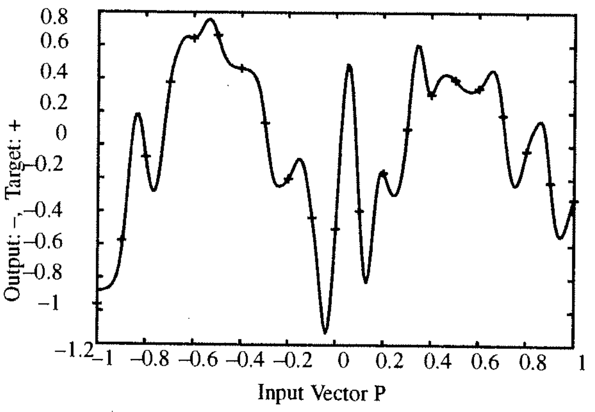


Figure 2.10 Overfitting [10].

The correct size of the MLP network for each problem can be determined by using several different sizes of networks. When the number of neurons is increased the error calculated using the training information decreases. When independent test information is used the error first decreases as well as the underfitting decreases. In some point the error starts to increase because of overfitting. The correct size of the MLP network is in the point where the error calculated using the test information is minimized. In this study that method was employed to determine the correct sizes of neural networks.

In [25] an overview of advanced methods used for determining the correct size for MLP networks can be found. In [35] is presented an analysis of the expected test set error for nonlinear learning systems.

2.4.6 Solution of inverse MLP

The inverse function of MLP is needed when the direct kinematics of the boom structure are modelled using MLP. The inverse kinematics solution of the boom can be achieved by solving the inverse function of MLP, which is calculated using the gradient descent method.

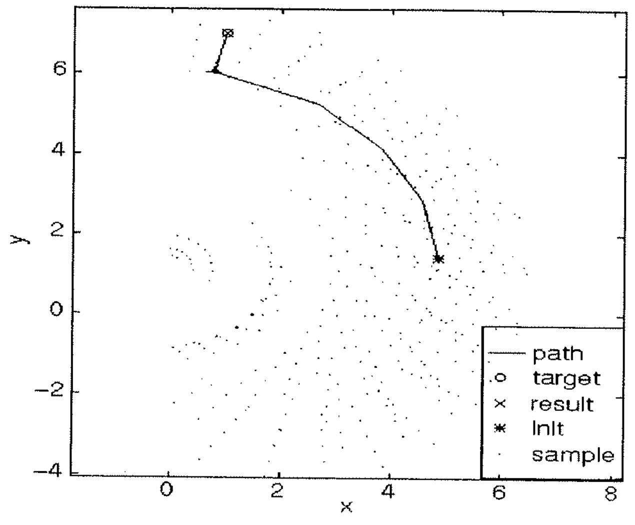


Figure 2.11 Statevector from initial state to desired state.

The selected MLP structure consists of one hidden layer with the sigmoidal activation function, equation 46, and linear function output layer, equation 44. The problem is similar to the learning of the network by the back–propagation algorithm. In this case the weights and bias values are fixed and the input must be changed. Changing the input value is done in several steps. The result is a series of statevectors that describe the change from the initial state to the desired state. This is illustrated in figure 2.11. The statevectors are calculated as the following equations present [27].

The sum squared output error E_s between the desired and the actual output is defined as:

$$E_s = \frac{1}{2}\sum_i \| d_i - y_i \|^2 \tag{59}$$

where d_i is the desired and y_i is the actual output of neuron i. The partial derivatives of output error with respect to the inputs is:

$$\frac{\partial E_s}{\partial x_k} = -\sum_j w_{jk}^{(1)} \delta_j^{(1)} \tag{60}$$

where x_k is the input value and $w_{jk}^{(1)}$ is the weight of neuron j for input value k. The deltas of the hidden layer are:

$$\delta_j^{(1)} = (1 - h_j^2)\sum_i w_{ij}^{(2)} \delta_i^{(2)} \tag{61}$$

where $(1-h_j^2)$ is the derivative of the hyperbolic tangent activation function and h_j is the output of hidden neuron j. The deltas of the linear output layer are:

$$\delta_i^{(2)} = d_i - y_i \tag{62}$$

A simple gradient descent would then produce a series of statevectors as follows:

$$x_k(t + 1) = x_k(t) - \gamma \frac{\partial E_s}{\partial x_k} \tag{63}$$

where γ is the learning rate and t is the time step.

In the calculation of the inverse function of the network some input parameters can be constants during the iteration process. If the mapping has redundant degrees of freedom the convergence to the desired state must be ensured.

3 Studied structure and simulation model

3.1 Studied structure

The structure under investigation is the PATU 655 log crane manufactured by the Finnish company Kesla OY. The mechanical flexibility of the construction is significant, and so the deflection caused by mass load can clearly be noticed. The lift arm and pillar have a varying profile but the jib arm and extension have constant profiles. The log crane and its functional dimensions are shown in figure 3.1.

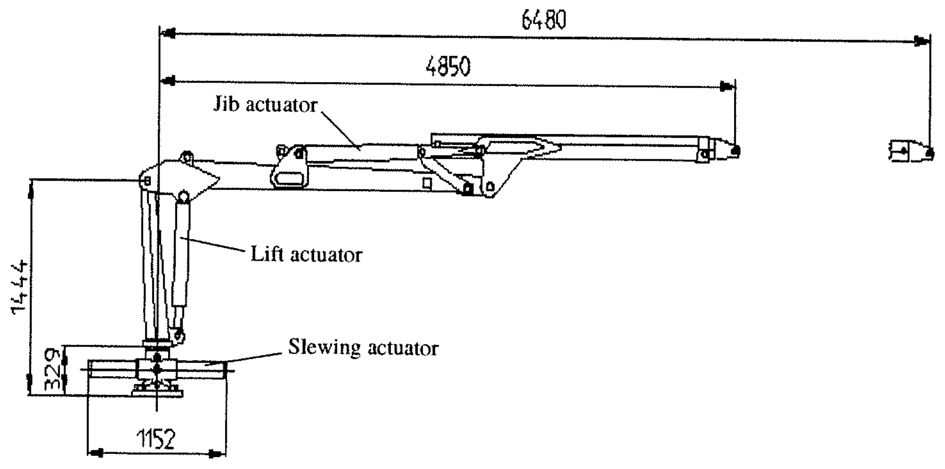


Figure 3.1 Studied log crane.

The crane is moved with lift, jib, slew and extension actuators. The motion of jib cylinder is transmitted to the jib arm via a four bar linkage. The maximum load of the crane is 500 kg with the extension fully out. In the laboratory assembly the hydraulic lift and jib cylinders were replaced using servo actuators manufactured by Mekarita KY. These actuators enable better positioning accuracy and repeatability in robot positioning. The actuators were controlled with a PC. The kinematic dimensions of the crane as well as the kinematic model are presented in appendix A.

3.2 Simulation model of the log crane

The complete simulation model was constructed in ADAMS by connecting the flexible and rigid mechanism parts to each other by joints. The equations describing the hydraulics were added using the mathematical properties of ADAMS [40].

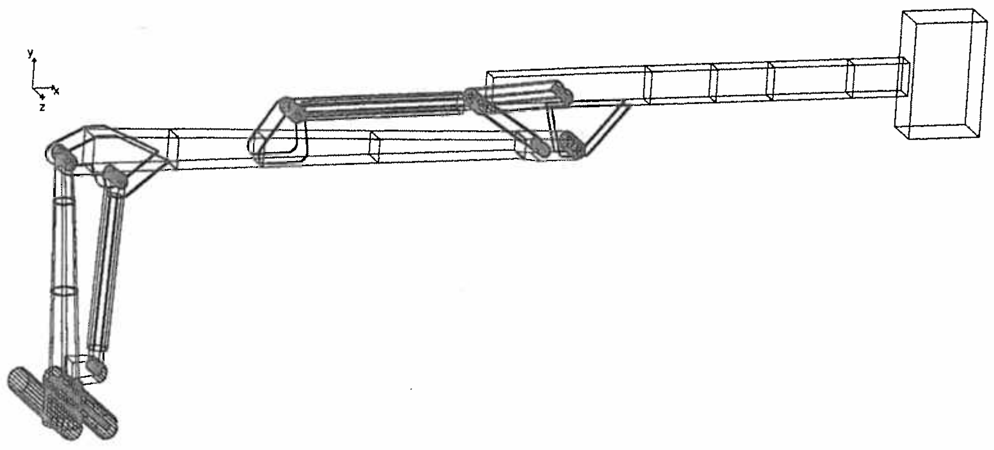


Figure 3.2 Simulation model of the log crane.

Four mechanism members, pillar, lift arm, jib arm and extension were modelled flexible using the lumped mass approach, figure 3.2. The ground connection of the boom was also modelled as flexible using a torque spring. The four bar linkage and the actuators were modelled as rigid parts. The flexible members of the mechanism were modelled in the ANSYS finite element program. Due to the simplicity of the flexible members' cross section the modelling was performed by linear 6–DOF beam elements. The pillar was modelled using three elements, the lift arm using six elements, the jib arm using five elements and the extension using four elements. The separate FE–models were imported to ADAMS using condensed stiffness and mass matrices. The correctness of the condensation was checked by comparing the eigenfrequencies of the condensated and non–condensated models.

The original hydraulic circuit of the boom was modelled utilizing the semi–empirical approach [14]. The equations describing the hydraulics are presented in [33] ,[14] and [40].

The following assumptions and simplifications were made in order to decrease the required computation time to a reasonable level. The assumptions and simplifications are discussed in detail in [31].

- Joint clearances were not modelled. Clearance can cause a loss of contact between the tin and the bearing, which causes impact forces. Analyzing the impact forces requires a large amount of computing capacity and can fundamentally be limited [7].
- The joints of the boom were modelled with no friction forces. In [32] the effect of joint friction in a one degree of freedom boom system is studied. The results show that the friction force has minor effects on the dynamics of the system.
- The linear properties of the material. The model assumes that plastic deformation only takes place in a small region in such a way that local yielding does not affect the global behavior of the crane [31].
- Distortion, warping and other deformation forms of the out–of–beam theory. These phenomena are not taken into account in the model of the crane. The model assumes that these phenomena have only minor effects on the flexibility of the structure [31].

The reliability of the model was verified by comparing the calculated results with the results measured from an existing structure. The verification of the simulation model is presented in detail in [33].

3.3 Using the simulation model in the calculation of training information

The training information for neural networks consisted of actuator strokes and coordinates of the boom end–effector in Cartesian space using different mass loads. Furthermore, for separate deflection compensation, information of change in the end–effector coordinates between flexible and rigid structures with different actuator strokes and mass loads was needed. Measuring the needed training information from existing structure would have been very slow and time consuming due to the accuracy needed. Measuring the change in end–effector coordinates caused by mass load is almost impossible. The problem was solved by using the simulation model in the generation of training information.

In the simulation model the control circuit of motions was assumed to be optimal, and thus hydraulic flexibility would not affect the positioning of actuators. Therefore it was possible to remove the hydraulic circuits from the simulation model and use motions instead of actuator forces to move the boom arms. This modification enabled faster analysis so that more information could be calculated. The training information was calculated in static equilibrium state, which also justifies the use of motions instead of actuator forces.

Calculation of the analysis was made automatically, figure 3.3, by a FORTRAN–language program that changed the actuator strokes and mass load values to the ADAMS–Solver input file. After the changes the program started the ADAMS–Solver that calculated static equilibrium analysis and wrote the end–effector coordinates to the result file. The results, actuator strokes and the amount of mass load were then collected to the final result file as one training vector.

Each vector included actuator strokes, mass load and depending on the method being trained either boom end–effector coordinates or amount of deflection due to mass load. The interval in actuator strokes and mass load between two analyses can be seen in table 4.1.

Table 4.1 Training information calculation intervals

	S1 [m]	S2 [m]	S3 [m]	mass [kg]
Minimum	0.0	0.0	0.0	50
Maximum	0.535	0.78	1.6	500
Increment	0.066875	0.0975	0.4	50
No. of values	9	9	5	10

By these numbers the number of training vectors is 4050. Anyhow, some vectors are infeasible due to mechanical constraints. The results include positions where X–coordinate was less than zero or two boom links were inside each other. When these vectors were removed the number of training vectors decreased to 3670. The settling of the training points to the operation area is shown in figure 3.4.

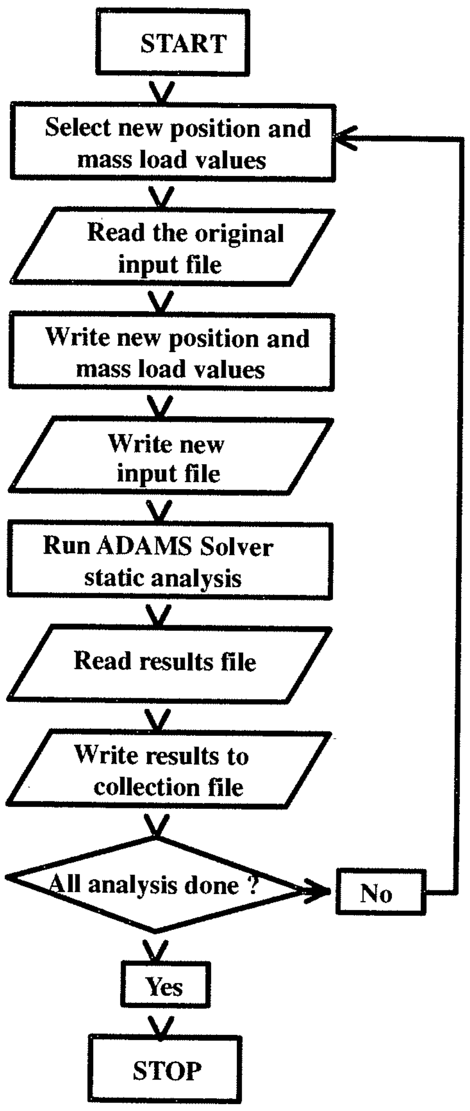

START
Select new position and mass load values
Read the original input file
Write new position and mass load values
Write new input file
Run ADAMS Solver static analysis
Read results file
Write results to collection file
All analysis done ?
No
Yes
STOP

Figure 3.3 Training information calculation algorithm.

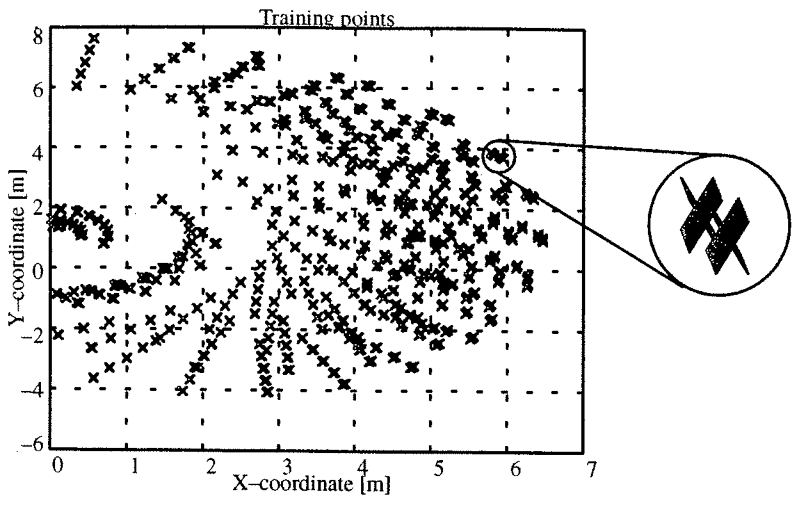


Figure 3.4 Training points in operation area.

The training information covers the operation area of the boom quite well. Furthermore the end –effector of the boom has been positioned to several points with different actuator stroke combinations. The influence of deflection to the position accuracy of the end–effector can be seen in figure 3.4. The deflection can be seen as a slight inaccuracy in the Xs that describe the coordinate points of boom end–effector as there are several coordinate points close to each other. The thickness of the Xs is directly relative to deflection.

CPU–times used in the computation of the training information are at a reasonable level; the computation of 4000 vectors took less than 25 minutes CPU–time using a DEC Alpha 600 computer.

4 Use of neural networks in robot positioning of a large flexible manipulator

There are several different methods that utilize neural networks in robot positioning. In some methods, neural networks are used for modeling the Jacobian matrices of a manipulator [23], [24] and [52]. The use of Jacobian matrices demands more mathematical background of the problem, but in dynamic control problems it is important that position, velocity and acceleration are taken into account [24]. Generally neural networks can be used in system identification as figure 4.1 presents.

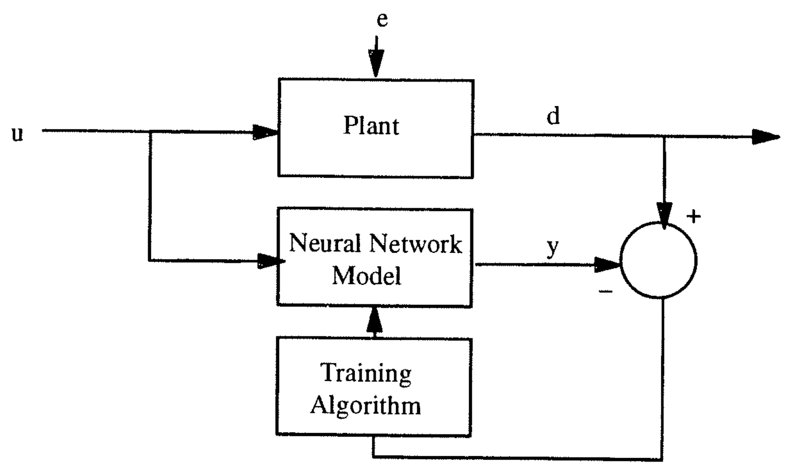


Figure 4.1 Neural network for system identification [54].

In this study, four different neural network methods were used in the robot positioning of a large flexible manipulator in static robot positioning situation. In the first method a neural network was used for modelling the magnitude of the deflection. The second method employed a neural network to model the direct kinematics of the manipulator with elasticity characteristics. The inverse kinematics solution was carried out by computing the inverse function of the neural network [27]. In the third method two neural networks, one for the direct rigid kinematics of the manipulator and the other for the deflection of the manipulator, were used for modelling the direct kinematics of the manipulator with elasticity characteristics. The inverse kinematics was achieved by computing the inverse function of the combined network [27]. In the fourth method, neural networks were used for modelling the inverse kinematics of the manipulator with elasticity characteristics.

The suitability of the different methods was tested by simulating the positioning of the studied boom structure at 25 separate coordinate points shown in figure 4.2.

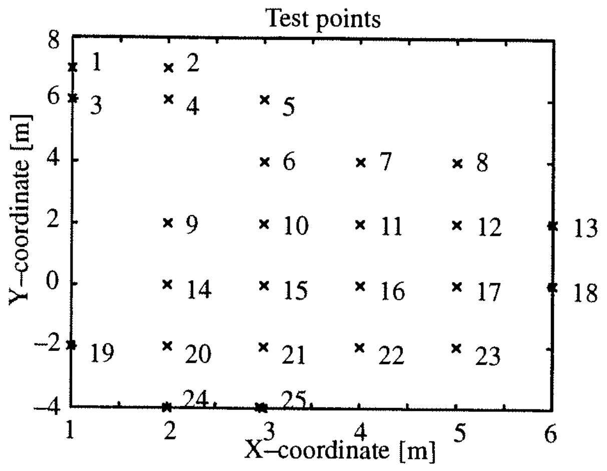


Figure 4.2 Test points in working area.

The test points were not included in the training information. The positioning of each point was tested using five different mass loads, namely 30, 120, 260, 310 and 470 kg, which were neither included in the training information. The results were simulated using the same simulation model as in the computation of the training information.

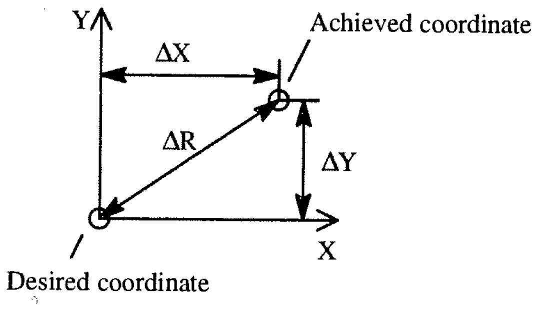


Figure 4.3 The specification of positioning errors.

In the simulation it was assumed that the controller functions optimally so that the desired actuator strokes were achieved with absolute accuracy. The results were computed as static analyses when the velocities and accelerations of the boom were zero. Figure 4.3 shows how the errors were specified. The value of position error was calculated by subtracting the achieved value from the desired value.

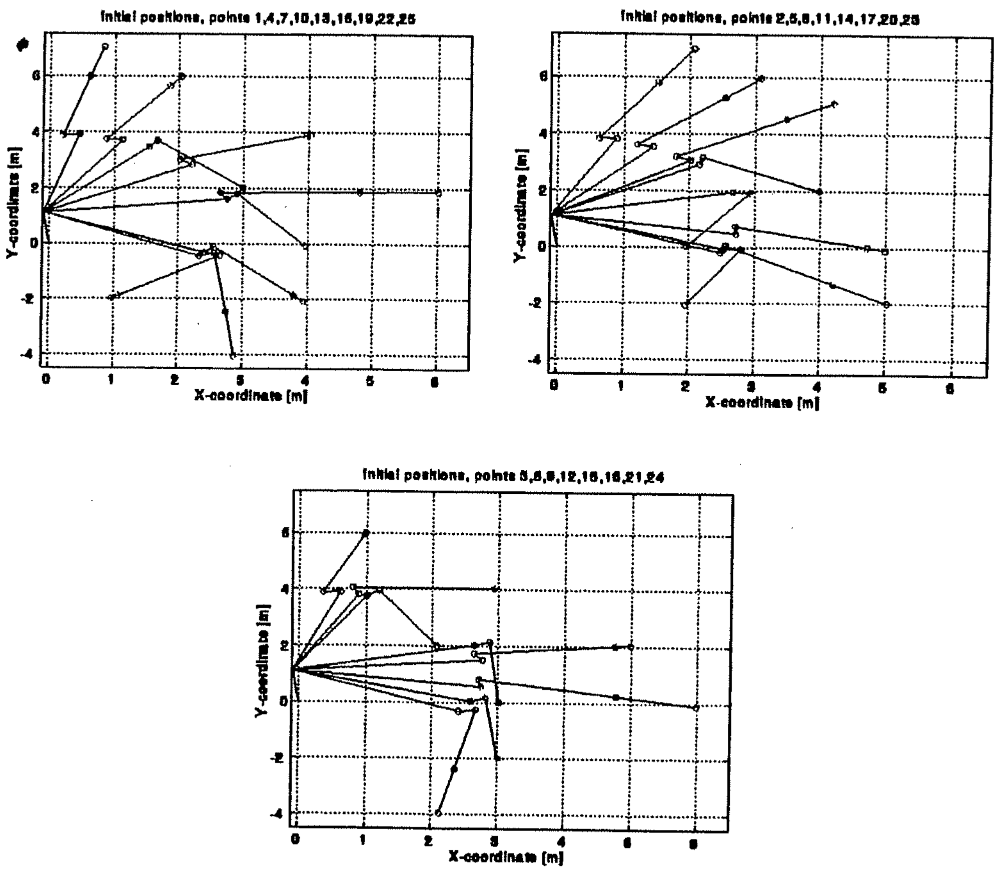


Figure 4.4 Initial positions for test points.

Because of the redundancy of the studied manipulator, each coordinate point can be achieved using several different manipulator link position combinations. To ensure that all methods studied use the same combination in each point and thus make the results comparable, a certain initial position was given to each point, figure 4.4.

The initial positions were calculated by choosing the extension cylinder stroke using the heuristic method, presented in appendix B, to reduce the degrees of freedom to a non–redundant level and then using the analytical model of the rigid manipulator. The actuator strokes were then rounded to the nearest full centimeter in order to create some noise to the position information.

4.1 Compensation by using a combined analytical kinematics solution and a neural network

Using a neural network in the modelling of the position error of a redundant manipulator requires that the position error is known as a function of the actuator strokes or joint angles and mass load. Employing a simulation model of the flexible manipulator in several different actuator strokes and with several different mass loads, the corresponding coordinates of the end–effector of the manipulator are computed. Comparison coordinates are computed using the same actuator strokes with a kinematics model of the rigid structure. A change of mass is not necessary because the amount of the mass has no effect on the kinematics of the rigid structure.

The used learning information is normalized between -1 and 1. The normalized sequences of the used actuator strokes and mass load are presented in figure 4.5.

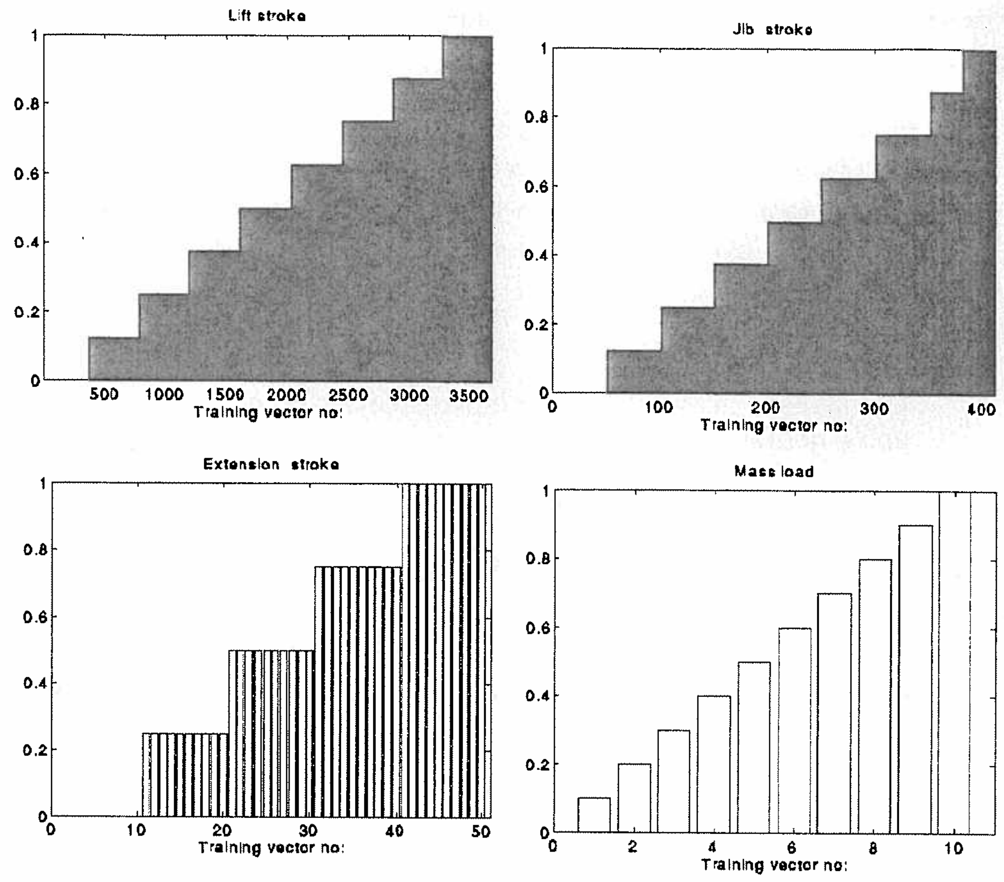
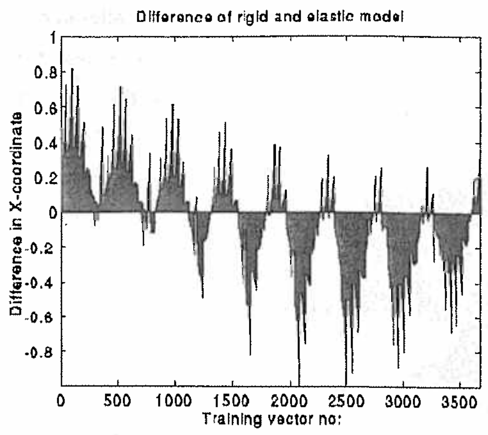


Figure 4.5 Sequences of normalized actuator strokes and mass load.

The difference in the achieved end–effector coordinates between the kinematics model of the rigid structure and the model with structural flexibility was used for finding out the position error, figure 4.6. A neural network was employed to model the error as a function of actuator strokes and mass load. By using this value as a correction term in control values, it was possible to compensate the deflection due to structural flexibility.



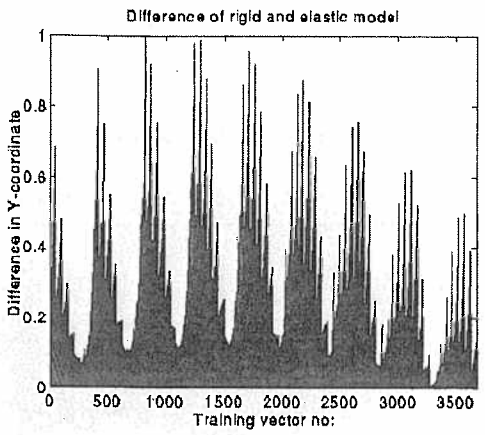


Figure 4.6 Normalized difference in achieved end–effector coordinates.

The actuator strokes to be used as initial values for the neural network were calculated using an analytical kinematics model of the rigid structure. The analytical model used genetic algorithm [12] for solving the required actuator values. The genetic algorithm was used in order to study it's capabilities in solving the inverse kinematics problem. The fitness function of the genetic algorithm was the magnitude of the position error. When the error was minimized the theoretical actuator strokes for the given end–effector coordinate point were achieved. To ensure that the genetic algorithm selects the desired combination from numerous possible combinations, two individuals of the initial population of the genetic algorithm were given so that they corresponded to the current actuator strokes. The thirteen other individuals of the initial population were randomly selected values so that the probability to achieve the new coordinate values nearby the existing solution and with minimum change in actuator strokes, is obvious when considering small changes in required coordinates. The solution was accepted if the position error was less than five millimeters. The maximum number of the populations was restricted to 1000 in order to keep the computation time at a reasonable level. The convergence of the genetic algorithm is studied furthermore in [41].

The analytical model was firstly used for calculating theoretical actuator strokes for the desired coordinate point (X,Y). The actuator strokes and the mass load were used as input values for the neural network, which computed the magnitude of the positioning error of the end–effector in both the X– and Y–coordinates. The magnitudes of the errors were added to the desired values, and the results were then used in the analytical model that was used for calculating the required actuator strokes S1, S2, S3 as shown in figure 4.7. The selected network structure had 14 sigmoidal neurons in the hidden layer and two linear neurons in the output layer. The network structure was fully connected. The training method used was the Levenberg–Marquardt algorithm but the back–propagation algorithm was also tested.

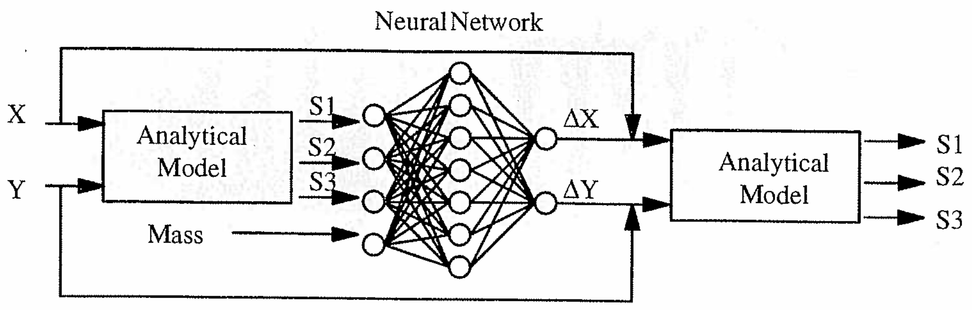


Figure 4.7 Neural network combined with analytical model.

Figure 4.8 presents the positioning error of the end–effector in test points with combined analytical model and the neural network.

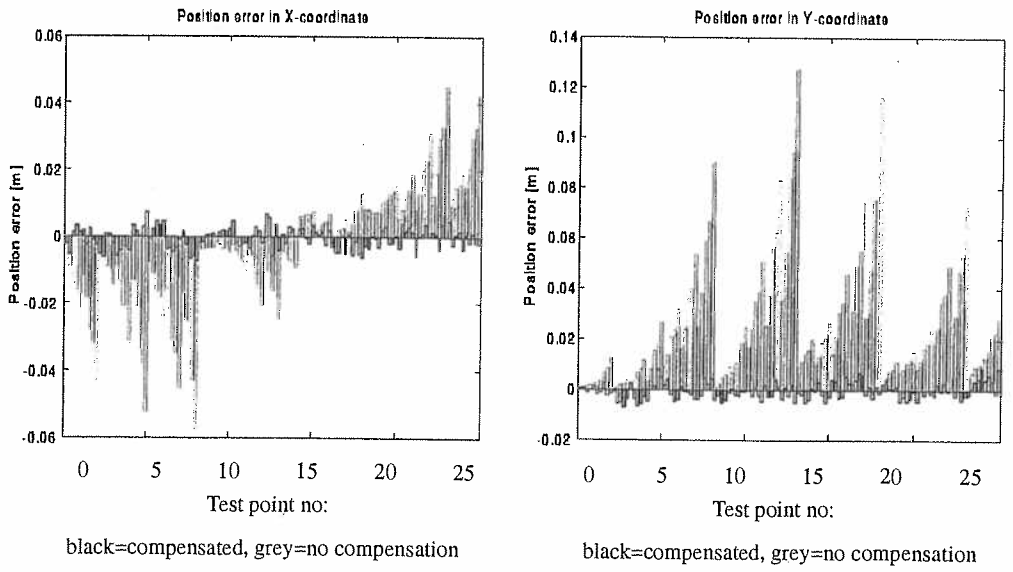


Figure 4.8 Positioning error using combined analytical model and neural network.

It can clearly be seen that the compensation reduces the amount of positioning error in both coordinates. The mean position error using this method was 4.14 mm. The maximum error was 9.21 mm in the coordinate point (2,–4) and the amount of mass load was 30 kg.

When the compensated position error is compared more accurately to the non–compensated result, it can be seen that the compensated case gives better results in all but two coordinate points. These points are point (1,6), where the error is 2.5 mm bigger using 30 kg mass load and 3.1 mm bigger using 120 kg mass load than without compensation, and point (2,2), where the error is 0.74 mm bigger using 30 kg mass load. In these points the positioning error without compensation is less than 6 mm. The point (2,2) does not have training information related closely to it, as can be seen in figure 3.4. Furthermore the smallest payload used in the training information was 50 kg, so the error might be smaller if the training information included smaller payloads. The method still works in most cases outside the training information, which proves that it is useful and stable. The result can partly be explained by remembering that the role of the neural network in positioning is only the correction of deflection. More detailed comparison of the results using different methods can be found in table 4.1, page 55.

4.2 Direct kinematics and inverse neural network

Using neural networks in robot positioning requires information of actuator strokes or joint angles, mass load and corresponding coordinates of the boom end–effector. There are two possible methods for using neural networks to model this kinematic relation. One is to use the neural network directly in the modelling of the inverse kinematics of a manipulator. The other is to use the neural network in modelling direct kinematics and then calculate the inverse function of the neural network, as shown in figure 4.9. The inverse function of the plant model neural network is used for calculating the required actuator strokes for the desired coordinate point in given mass load. The inverse function is calculated with gradient descent [27].

During the iteration process the value of the mass load is kept constant. The initial values for the actuator strokes in the practical application are the current values measured from actuators. The network has only to be iterated a few times and the probability that the combination of the actuator positions to be chosen is near the existing one increases, when small changes in the coordinates are considered . In the testing the values presented in figure 4.4 were used as initial values.

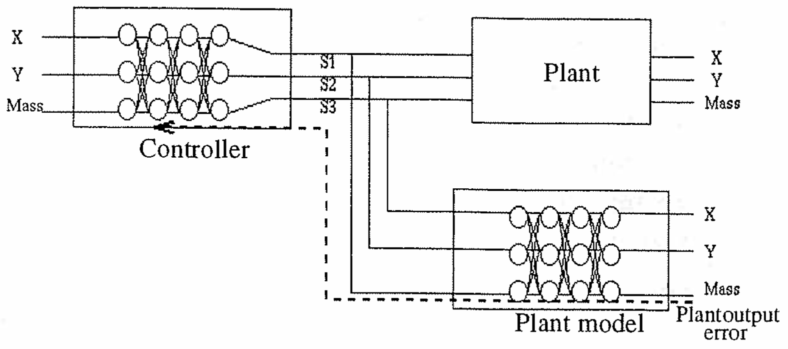


Figure 4.9 *Structure used in describing the direct kinematics of the manipulator and in the calculation of the inverse function of neural network [34].*

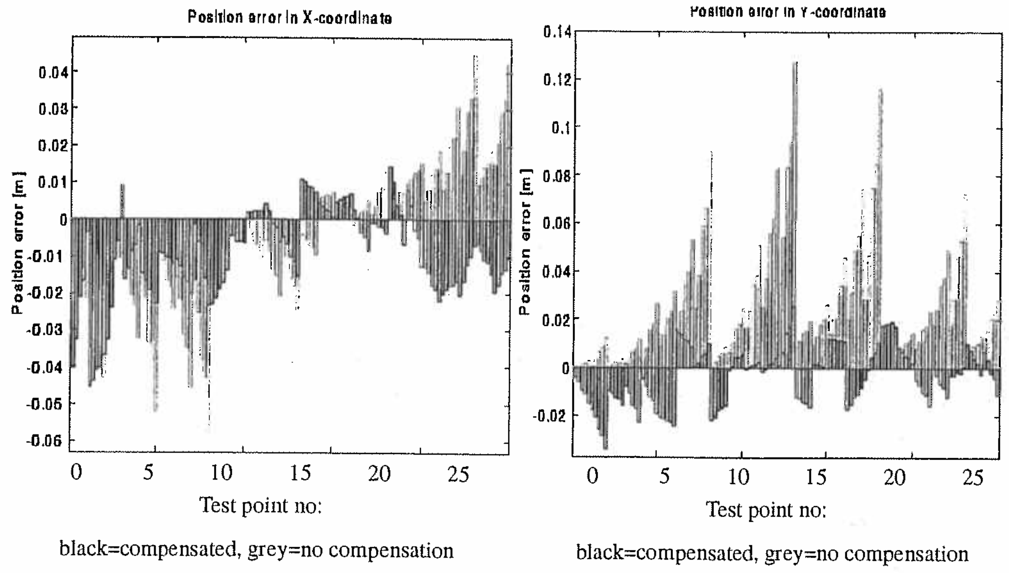


Figure 4.10 *Positioning error using direct kinematics and inverse neural network.*

Figure 4.10 presents the positioning error in test points using neural network with 18 sigmoidal neurons in the hidden layer and three linear neurons in the output layer for modelling the direct kinematics of the boom. The network structure was fully connected. The used training method was the Levenberg–Marquardt algorithm.

Using this method the mean positioning error was 16.53 mm, and the maximum error was 49.83 mm. When the results are considered more accurately, the greatest positioning errors can be found in the coordinate points (1,7) and (2,7). The training information around these points, especially the point (2,7), is quite scarce, as figure 3.4 presents. The results might be better if the learning information covered the working area better. If these two points were not included, the maximum position error would be 33.93 mm, and the mean error 14.70 mm.

This method is also quite sensitive for the error tolerance used in the iteration of the inverse function of the network. By decreasing the error tolerance from 1e–5 to 1e–7, the mean positioning error diminished 43 percents. The change of error tolerance from 1e–7 to 1e–8 improved the results only 3 percents. The results presented here are computed using error tolerance 1e–7. The mean amount of iterations using this tolerance value was 129 and the maximum amount of iterations was 232. Using tolerance value 1e–5 the values were 114 and 155 respectively. The error tolerance value 1e–8 increased the mean amount of iterations to 132 and the maximum amount to 250.

4.3 Compensation by using modular neural networks

Modular networks can be used for solving large and complex problems by dividing the problem into a set of smaller sub–problems [25]. In [54] modular networks are used for modelling the direct dynamics and inverse Jacobian of a robot. Modular and hierarchical networks are also employed in the control and navigation of mobile robots in [54].

The use of two modular networks in the modelling inverse kinematics of the manipulator with elasticity characteristics was achieved by dividing the kinematics of the flexible manipulator to the direct kinematics of the rigid manipulator and to the positioning error as shown in figure 4.11.

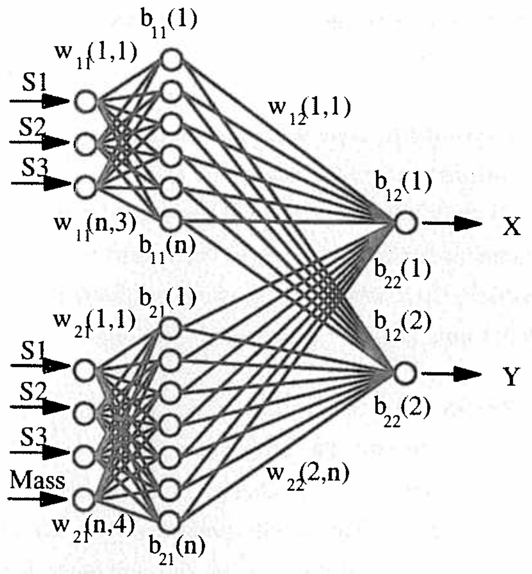


Figure 4.11 Used modular neural network structure.

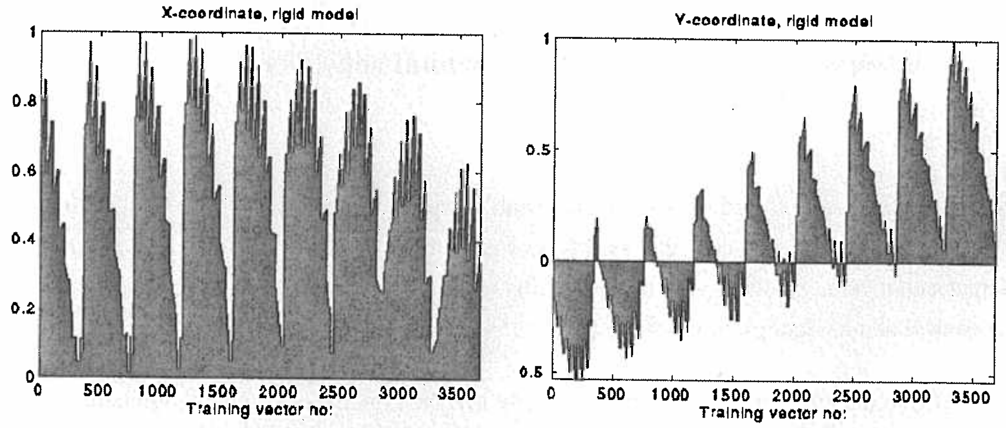


Figure 4.12 Normalized X– and Y–coordinates using a rigid model.

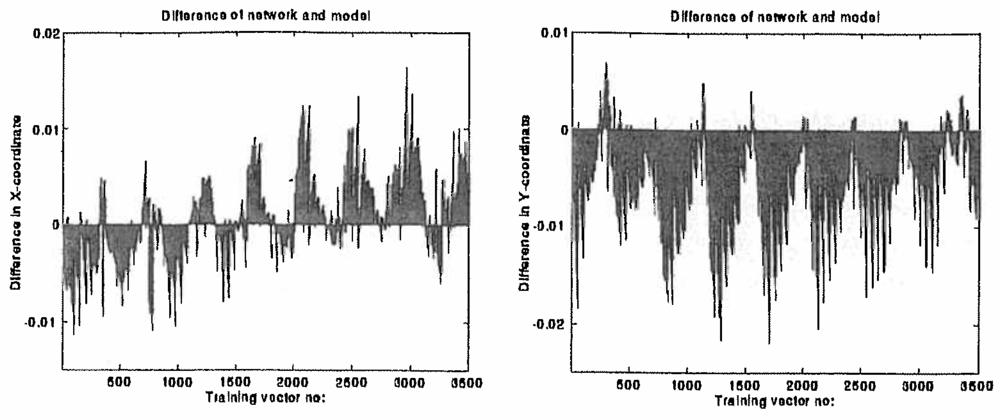


Figure 4.13 *Normalized difference of the trained neural network model and the simulation model with structural flexibility.*

Firstly one neural network ($w_{11}, b_{11}, w_{12}, b_{12}$) was trained to model the direct kinematics of the rigid structure. Figure 4.12 presents the used normalized X– and Y–coordinate learning information. The used normalized actuator strokes are presented in figure 4.5. Next, a second neural network ($w_{21}, b_{21}, w_{22}, b_{22}$) was trained for modelling the difference between the rigid neural network model and the ADAMS model with structural flexibility, figure 4.13. Therefore, the second neural network compensates also the error of the rigid neural network model. The both networks consisted of 15 sigmoidal neurons in the hidden layer and two linear neurons in the output layer. Also in this case the used training method was the Levenberg–Marquardt algorithm.

For the case of a linear activation function the combination of the two separate neural networks ($w_{11}, \theta_{11}, w_{12}, \theta_{12}$) and ($w_{21}, \theta_{21}, w_{22}, \theta_{22}$) is calculated as:

$$\begin{aligned} d &= w_{12} \times h_1 + \theta_{12} + w_{22} \times h_2 + \theta_{22} \\ &= [w_{12} \quad w_{22}] \times \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} + (\theta_{12} + \theta_{22}) \end{aligned} \tag{64}$$

where d is the output of combined networks, w_{12} and w_{22} are weight matrices, h_1 and h_2 are output vectors of hidden layers, and θ_{12} and θ_{22} are bias vectors. Output vectors h_1 and h_2 are calculated as:

$$\begin{aligned} h_1 &= f(x_1 \times w_{11} + \theta_{11}) \\ h_2 &= f(x_2 \times w_{21} + \theta_{21}) \end{aligned} \qquad (65)$$

where w_{11} and w_{21} are weight matrices, x_1 and x_2 are input vectors of neural networks, θ_{11} and θ_{21} are bias vectors and $f(.)$ is the activation function. The combined modular networks model the direct kinematics of a manipulator with elasticity characteristics. The inverse kinematics can be solved by calculating the inverse function of a combined network [27]. Figure 4.14 presents the positioning error in the test points using modular neural networks.

This method had the mean position error of 13.20 mm, and the maximum error of 34.65 mm. The greatest positioning errors can be found in the coordinate points (1,6) and (1,–2). If these two points were not considered, the maximum position error would be 32.82 mm, and the mean error 12.09 mm. The position error is smaller than when using a single neural network as was expected.

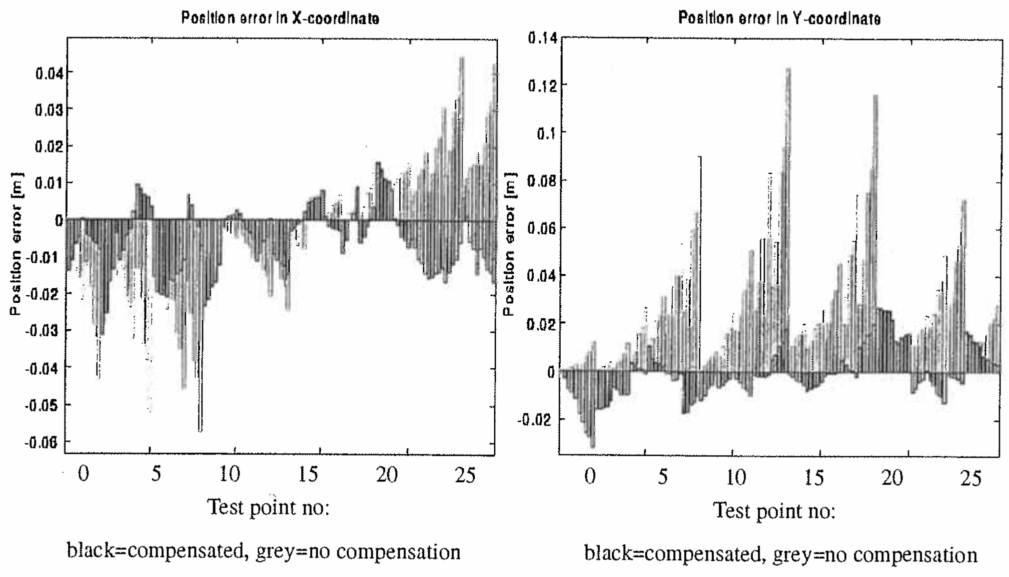


Figure 4.14 Positioning error using modular neural networks.

The change in the error tolerance had even bigger effects as the previous method: increase of the tolerance from 1e–7 to 1e–5 doubled the mean positioning error. The results presented here are computed using error tolerance 1e–7. The mean amount of iterations using higher tolerance value was 127 and the maximum amount of iterations was 163. Using tolerance value 1e–5 the values were 114 and 155 respectively. The number of iterations using higher tolerance is a bit smaller than in the previous method but the number of perceptrons is doubled so the required computing power is greater utilizing modular networks.

4.4 Inverse kinematics solution

The use of neural networks in modelling the inverse kinematics of a manipulator has been studied quite extensively. In dynamical control better accuracy can be achieved by using also the derivatives of position in trajectory planning [50], [23]. In this paper, only static positioning is considered, and thus only the inverse function of the boom end–effector position is taken into account. The inverse input– output relations are shown in figure 4.15. In the case of the redundant structure this relation is not a function, however. A certain coordinate point can be achieved using several actuator stroke combinations and there is no way to predict which combination is the output of a neural network. This can also cause some problems in the training of the network.

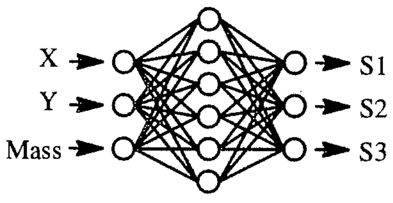


Figure 4.15 Neural network for modelling inverse kinematics.

Figure 4.16 shows the neural network structure which enables the prescribing of the desired combination by reducing the degrees of freedom using the extension actuator stroke as input value and thus converting the function of the neural network to an unambiguous function. The extension actuator stroke is computed using heuristic method presented in appendix B.

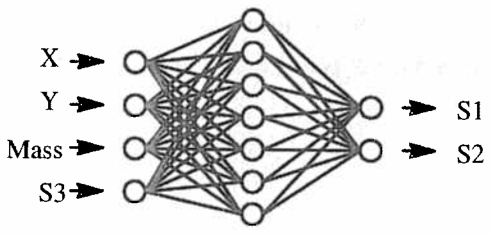


Figure 4.16 Neural network for modelling inverse kinematics with restriction.

The input vector includes the X– and Y– values of the desired coordinate point, figure 4.17, and the amount of mass load and the extension actuator stroke; the output vector consists of the required jib and lift actuator strokes, figure 4.5.

Figure 4.18 presents the positioning error in test points using a neural network for modelling the inverse kinematics of the flexible boom. The used neural network had 19 sigmoidal neurons in the hidden layer and was trained using the Levenberg–Marquardt algorithm..

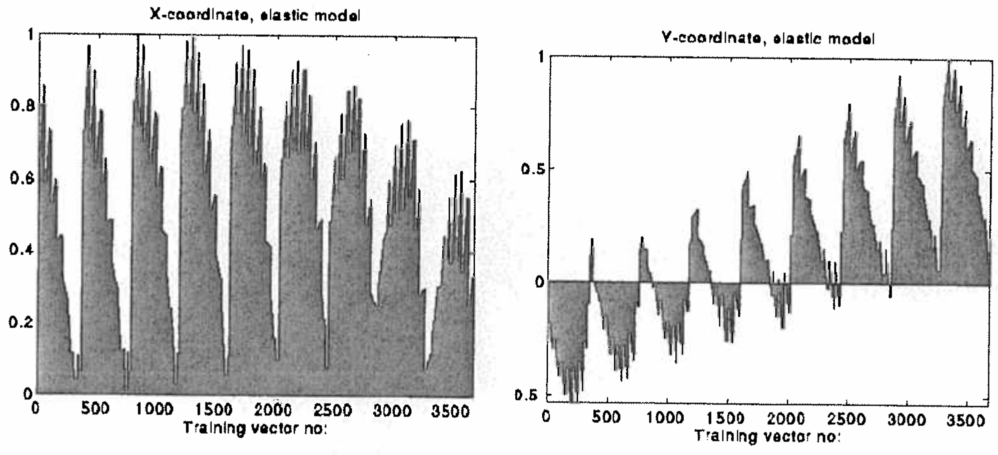


Figure 4.17 Normalized X– and Y–coordinates using a model with structural flexibility.

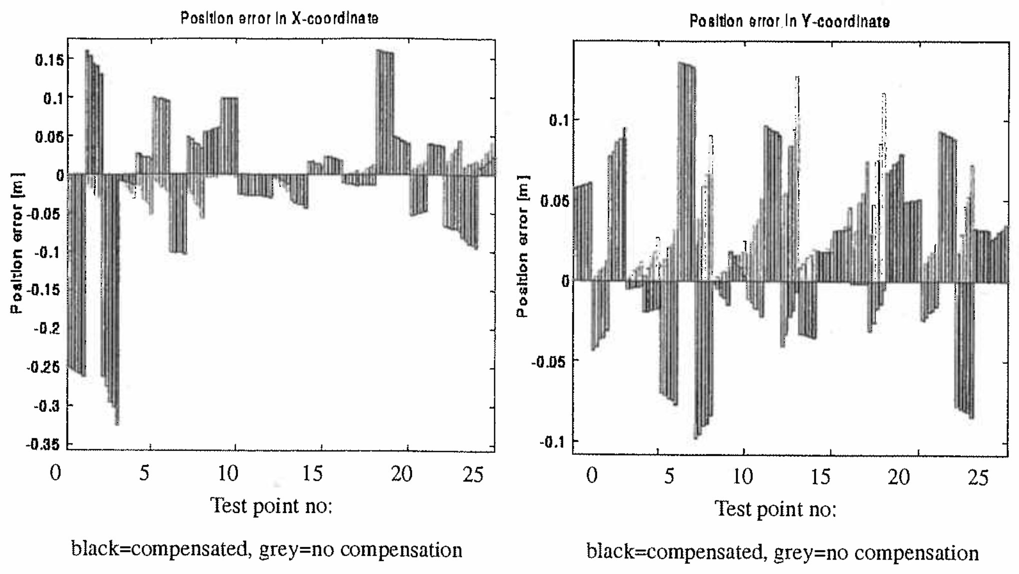


Figure 4.18 Positioning error using neural networks in modelling inverse kinematics.

Positioning using a neural network for the representation of the inverse kinematics of a flexible structure did not seem to work very well, figure 4.18. The maximum error was more than 30 cm. The main reason for this is the huge amplification of the mechanism. The change of one millimeter in actuator strokes can cause a positioning error of several centimeters in the boom end–effector. In the other methods the same problem does not exist, one method describes the amount of deflection as a function of actuator strokes and the two other methods use a very strict error tolerance in the calculation of the inverse function. The main advantage in this method is that it is computationally efficient. The computing of the actuator strokes for one coordinate point took 395 floating–point operations using MATLAB.

4.5 Simulated results

The simulated maximum and mean positioning errors without compensation and using different neural network architectures are presented in table 4.1.

Table 4.1. *Simulated positioning errors*

	No compensation	Combined analytical model and neural network	Direct kinematics and inverse function of neural network	Combination of two neural networks	Inverse kinematics
ΔX_{max} [mm]	57.4300	7.5500	45.3800	30.8700	324.7700
ΔY_{max} [mm]	127.3400	8.3700	33.6600	31.8200	135.5000
ΔR_{max} [mm]	129.6755	9.2058	49.8279	34.6512	338.3428
ΔX_{mean}[mm]	14.7157	2.6136	11.8616	8.3611	69.9901
ΔY_{mean}[mm]	25.5415	2.7380	9.8299	8.2357	45.4265
ΔR_{mean}[mm]	31.9331	4.1405	16.5295	13.2036	89.5450

The comparison of the results computed utilizing separate methods show that the best results are achieved using the analytical kinematics model of a rigid structure together with the neural network that describes the magnitude of the deflection. The mean positioning error of 4.1 mm enables the use of a large manipulator, for example, in loading or tunnel drilling applications. The methods that include the calculation of the inverse function of the neural network gave also quite good mean positioning accuracy. Especially by using combined modular networks even the maximum positioning error stayed at a reasonable level. The main advantage in these methods is that there is no need for any kind of kinematics model of the structure. The maximum error values can be diminished by distributing the training information more evenly to the working area of the manipulator. The modelling of the inverse kinematics of a flexible structure using neural network was an unsuccessful method. The main reason for that was the amplification of the studied structure. If the structure had smaller dimensions the method might work a bit better.

4.6 Experimental results

The functionality of the compensation using the combined analytical kinematics solution and neural network was tested by measuring an existing boom structure, figure 4.19.

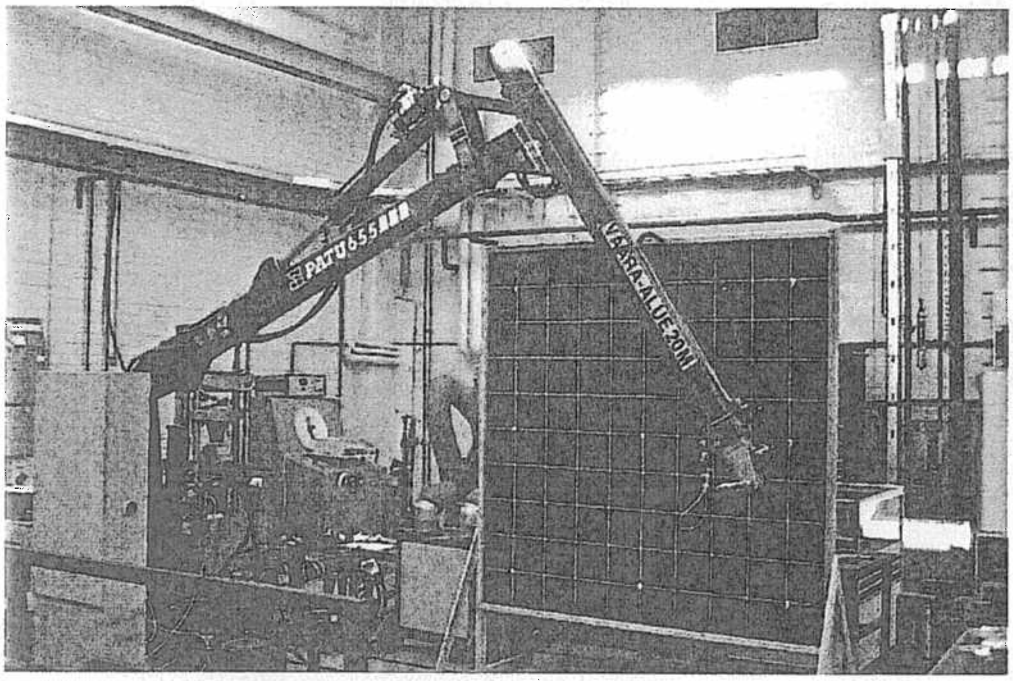


Figure 4.19 The measured boom structure.

The position of the boom end–effector was measured using a plywood grid and laser light. The plywood grid was positioned parallel with the boom so that the center of the grid was in the coordinates 3.0, 0.95. The measurements were done in nine separate points, figure 4.20, using three payloads: 0, 86 and 172 kg. Figure 4.20 also shows the used boom arm combinations.

The measured working area is chosen so that the direction of the force changes in the jib cylinder and the direction of the deflection changes in the jib arm. The selected area also covers the mostly used working area of the log crane quite well.

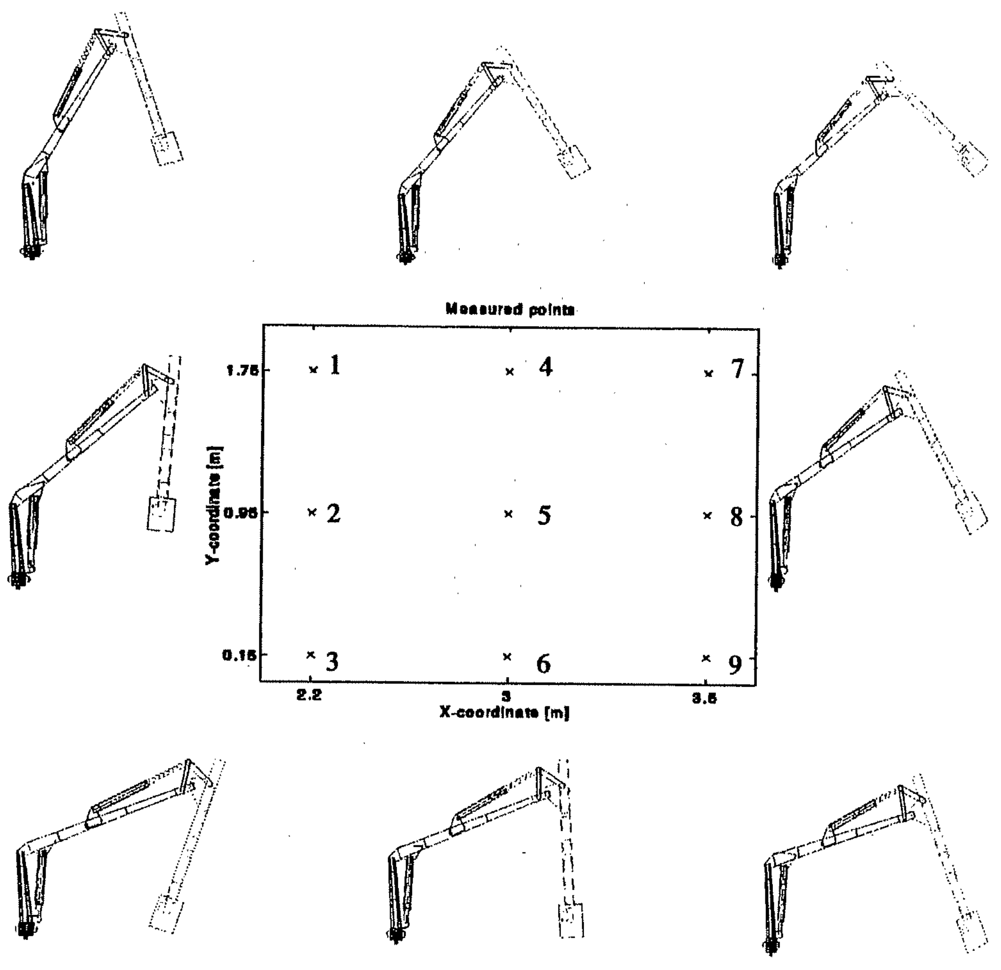


Figure 4.20 Measured test points in working area.

The laser transmitter was attached to the boom end so that the laser beam pointed the position of the boom end–effector to the plywood grid, figure 4.21. The achieved points were marked to the plywood with round stickers and the position of the stickers was measured in order to solve the positioning accuracy.

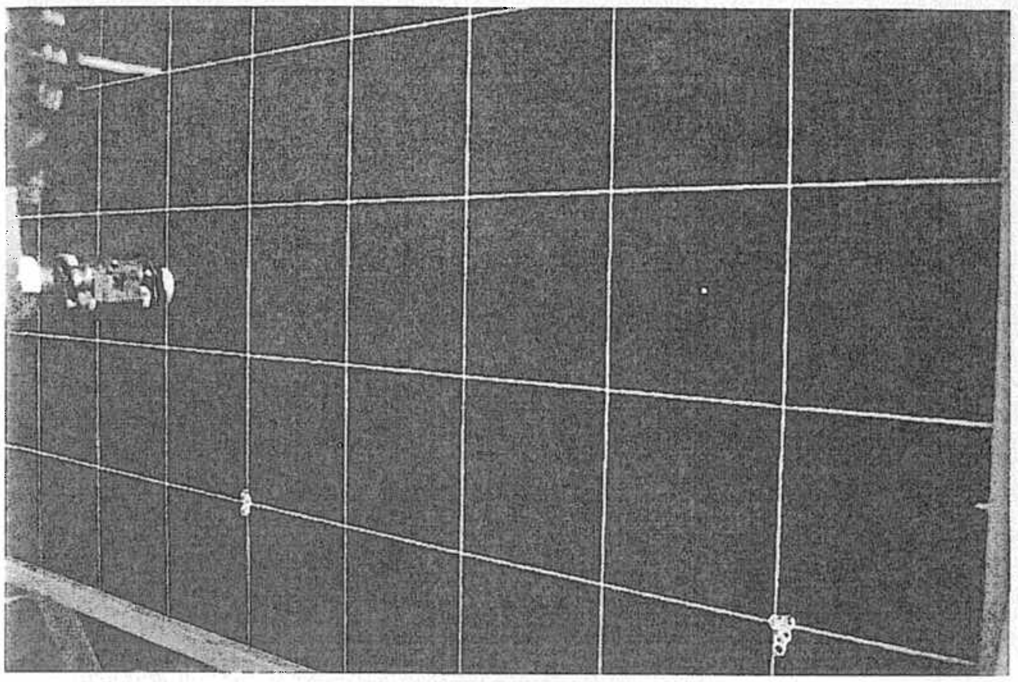


Figure 4.21 The laser transmitter and the plywood grid.

Figure 4.22 presents the measured positioning error in the X– and Y–coordinates. The error was measured from theoretical target points. Figure 4.23 presents the measured positioning error in the X– and Y–coordinates, where the error was measured using the point achieved with 0 kg pay load and compensation as a reference point. This was done in order to avoid errors caused by the inaccuracy in the kinematics model and the joint free play. The positioning accuracy in each point was measured using three mass loads: 0, 86 and 172 kg so that the leftmost bar of each point represents the error measured with 0 kg, the middle bar with 86 kg, and the rightmost bar with 172 kg mass load.

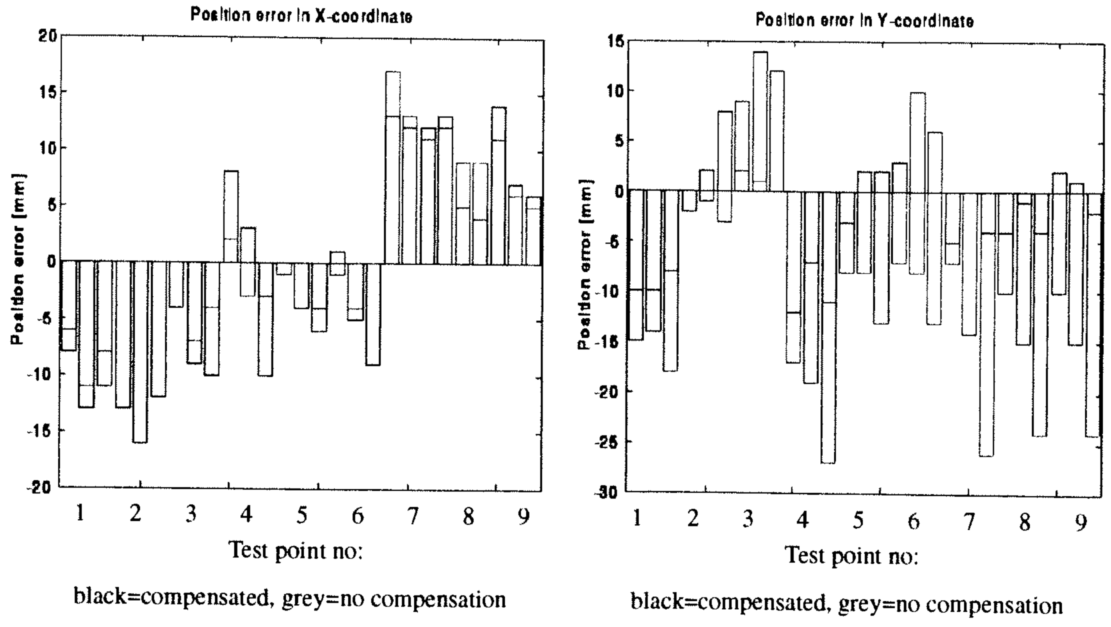


Figure 4.22 Measured positioning error from theoretical target point.

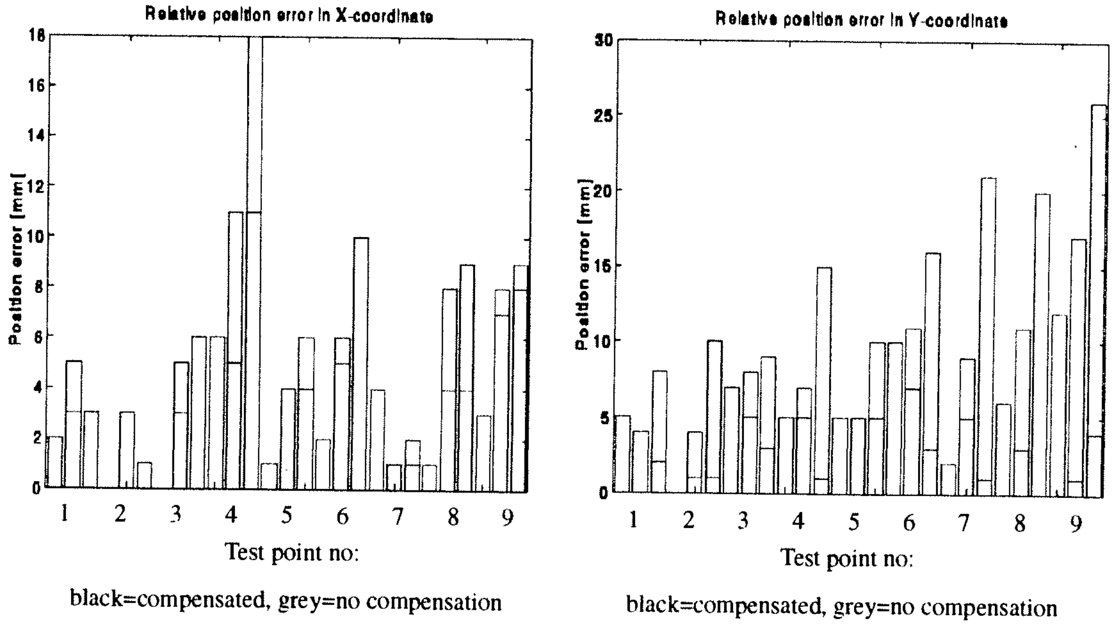


Figure 4.23 Measured positioning error from achieved reference point.

The measured maximum and mean errors are shown in tables 4.2 and 4.3. Table 4.2 presents the positioning errors measured from a theoretical target point, figure 4.22, and table 4.3 presents the errors measured using the point achieved with 0 kg pay load and compensation as a reference point, figure 4.23. The main reason for this presentation is to avoid positioning errors caused by joint free play, joint misalignment and inaccuracy in the kinematics model. The estimated measuring accuracy is 0.5 mm.

Table 4.2. Measured positioning errors

	0 kg no compensation	0 kg compensated	86 kg no compensation	86 kg compensated	172 kg no compensation	172 kg compensated	All no compensation	All compensated
ΔX_{max} [mm]	17	14	16	16	12	12	17	16
ΔY_{max} [mm]	17	12	19	14	27	12	27	14
ΔR_{max} [mm]	18.4	14.4	19.2	16.6	28.9	15.6	28.8	16.6
ΔX_{mean} [mm]	7.4	8.2	8.1	8.2	8.2	7.9	7.9	8.1
ΔY_{mean} [mm]	8.7	5.6	10.6	5.2	16.4	6.3	11.9	5.7
ΔR_{mean} [mm]	12.8	10.9	14.5	10.7	19.3	10.6	15.5	10.7

The results show that the maximum positioning error decreases from 28.8 mm to 16.6 mm when compensation is used. The mean error decreases from 15.5 mm to 10.7 mm. It can also be noticed that increase in payload does not have a significant effect on the positioning accuracy when compensation is used. When the reference point is changed to the point achieved with 0 kg mass load and with compensation on, the maximum error decreases even more, from 27.5 mm to 11.0 mm and the mean value decreases from 10.6 mm to 4.7 mm. The better accuracy is clearly due to better results in Y–direction accuracy.

Figures 4.22 and 4.23 show that positioning accuracy in test points 2 and 3 is better without compensation. As can be seen from figure 4.20 the position of the jib arm in these two points is on the other side of the vertical line compared to the other measured points. This causes the direction of the cylinder force to change from pulling to pushing and this changes the effect of joint free plays.

Table 4.3. Measured positioning errors using the achieved reference point

	0 kg no compensation	0 kg compensated	86 kg no compensation	86 kg compensated	172 kg no compensation	172 kg compensated	All no compensation	All compensated
ΔX_{max} [mm]	6	0	11	8	18	11	18	11
ΔY_{max} [mm]	12	0	17	7	26	10	26	10
ΔR_{max} [mm]	12.4	0	18.8	9.2	27.5	11.0	27.5	11.0
ΔX_{mean} [mm]	2.1	0	4.6	4.9	5.6	5.9	4.1	3.6
ΔY_{mean} [mm]	5.8	0	8.1	3.9	14.0	3.2	9.3	2.4
ΔR_{mean} [mm]	6.5	0	9.7	6.7	15.7	7.5	10.6	4.7

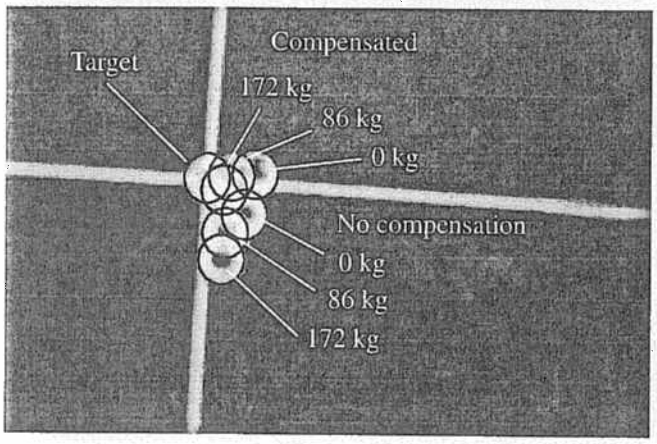


Figure 4.24 Achieved coordinates in point (3.8, 0.15).

Figures 4.24 and 4.25 present the achieved coordinates in points (3.8, 0.15) and (3.8, 0.95). These figures show the effect of compensation clearly. The effect of payload in deflection can be seen as a decrease of the Y–coordinate value when increasing the payload. Compensation keeps the points near each other. Especially the points using 86 kg and 172 kg payloads are very near to each other.

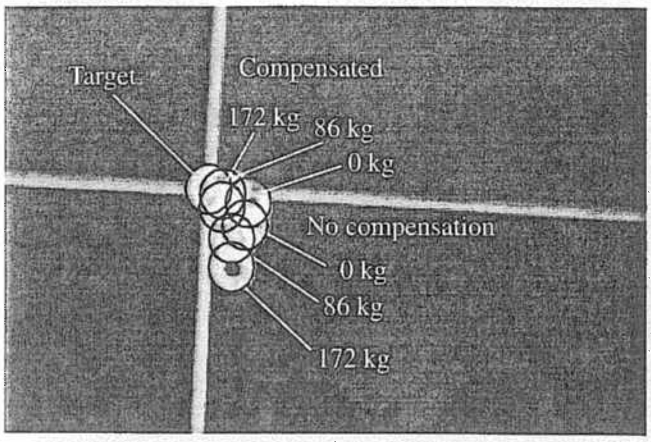


Figure 4.25 Achieved coordinates in point (3.8, 0.95).

5 Conclusions

The use of neural networks in the robot positioning of a large, redundant, flexible manipulator was studied. A three–degree of freedom log crane was used as an example. The study was done for a plane mechanism so the studied structure had redundant degrees of freedom. Training information, required by neural networks, for deflection compensation and for kinematics is almost impossible to measure accurately, but it could be easily computed utilizing a simulation model that includes structural flexibility. The verified simulation model was used for computing 4000 static analyses with different actuator strokes and using different amounts of mass load. A simulation model that describes the kinematics of a log crane was used for computing corresponding information for the rigid structure. Four different neural network architectures were trained and tested.

The first neural network architecture was used for determining the magnitude of deflection as a function of actuator strokes and mass load. The trained network was used together with an direct kinematics model of a rigid manipulator and genetic algorithm which were used for solving the inverse kinematic of a rigid manipulator structure. Firstly the analytical model was used for computing the theoretical actuator strokes for the desired coordinate point, the neural network then computed compensation values for both X– and Y– coordinates. These values were added to the desired point and the analytical model was applied again. The achieved actuator strokes were used as reference values in robot positioning for the desired coordinate point. The second neural network architecture was used for modelling the direct kinematics of a flexible manipulator. The network computed the coordinates of the end–effector of the manipulator for a given actuator stroke and mass load combination. The inverse kinematics solution was achieved using the inverse function of the network. The inverse function of the trained network was computed utilizing the gradient descent method. Because of the redundancy of the studied structure it was necessary to ensure that the network converged to the desired solution. This was done by keeping the value of mass load constant during the iteration and giving the initial values for the actuator strokes nearby the desired ones. The third network architecture consisted of two separate networks that were combined in order to model the direct kinematics of the flexible manipulator structure. One network was trained to model the direct kinematics of the rigid structure and the other network was used for modelling the difference between the network modelling the rigid structure and the simulation model including structural flexibility. The inverse kinematics of a flexible structure was determined by the inverse function of the combined networks computed utilizing the gradient descent method. The convergence to the desired solution was ensured as in the previous method. The fourth neural network architecture was used for modelling the inverse kinematics of a flexible structure. In order to avoid the problems caused by the redundancy of the studied structure, one actuator stroke was used as an input value besides the end–effector coordinates and mass load. The value of the actuator stroke was calculated using a heuristic rule based on circles describing the working area of the manipulator.

The functionality of the selected architectures was tested by a simulation model and measuring the existing log crane. The robot positioning was simulated in 25 separate test points. The positioning accuracy in each test point was simulated with five different payloads. Neither the test points nor the amounts of payload were included to the training information. The functionality of the best method based on simulation results was tested by measuring the existing log crane. Measurements were done in nine separate test points using three different payloads in each point.

The simulated and measured results show that by using neural networks in robot positioning of a redundant, flexible manipulator it is possible to reduce position error due to deflection caused by payload. Neural networks are also capable of solving the inverse kinematics of a flexible robot structure. Different neural network architectures can be used in different types of applications, depending on the available computing power and required accuracy. Adding the compensation loop to an existing robot controller is also possible but the computing time used for calculation one coordinate point is doubled due to required iteration procedure. The results also show that verified simulation models with structural properties can be used in collecting training information for neural networks. Especially data, such as the boom end–effector position in space, that is difficult, slow or impossible to measure from a real structure can be achieved accurately and quickly.

5.1 Future for robot positioning of flexible manipulators

The increase of computational power will enable the use of model based controllers in robot applications. These controllers will be capable of modelling the dynamical behavior of the robot in real–time. The structural flexibility will anyhow cause problems because it increases the number of degrees of freedom in the model and demands more computational power. One possible solution might be the combination of neural network models together with analytical models. Certain deformation behavior of each robot link could be added to a rigid kinematics model.

The advantages in sensor technology might give a possibility to measure the position of the robot end–effector in space and that way enable the use of direct control of the position in Cartesian space. However, the mapping from Cartesian space to joint or actuator space is required but the exact knowledge about the controlled variable makes the control much easier. Machine vision offers also possibilities to determine the position of a desired point in space. The placement of the required cameras and the need for large computational power restrict the use of vision systems nowadays. The goal of the study of structurally flexible robots is to increase the relation of payload to the manipulator's own weight and that way make robots more efficient. The other advantage is that by using intelligent control systems it is possible to use more rough mechanisms in robot applications.

References

[1] Baldi Pierre, Hornik Kurt, Neural Networks and Principal Component Analysis: Learning from Examples Without Local Minima, Neural Networks, Vol. 2, 1989. pp. 53–58.

[2] Baroglio, C., Giordana, A., Kaiser, M., Nuttin, M., Piola, R., Learning Controllers for Industrial Robots, Machine Learning–The Journal, Vol. 23, 1996.

[3] Bestaoui Yasmina, An Unconstrained Optimization approach to the Resolution of the Inverse Kinematic Problem of Redundant and Nonredundant Robot Manipulators, Robotics and Autonomous Systems Vol. 7, 1991, pp. 37–45.

[4] Bricout, J.N., Debus, J.C. & Micheau, P., A Finite element model for the dynamics of flexible manipulators, Mechanism and Machine Theory, Vol. 25, No. 1, 1990, pp. 119–128.

[5] Craig, J.J., Introduction to Robotics: Mechanics and Control, 2nd ed., Addison–Wesley Publishing Company, Inc. 1989.

[6] Craig, R.R., Structural dynamics: An introduction to computer methods, John Wiley & Sons, 1981.

[7] Deck, J.F., Dubowsky, S., On the limitations of predictions of the dynamic response of machines with clearense connections, Journal of Mechanical Design, Vol. 116, 1994, pp.833–841.

[8] De Luca Alessandro, Panzieri Stefano, End–effector Regulation of Robots with Elastic Elements by an Iterative Scheme, International Journal of Adaptive Control and Signal Processing, Vol. 10, 1996, pp. 379–393.

[9] DeMers David and Kreutz–Delgado Kenneth, Global Regularization of Inverse Kinematics for Redundant Manipulators, Advances in Neural Information Processing Systems, Vol. 5, 1993, pp. 255–262.

[10] Demuth Howard, Beale Mark, Neural Network Toolbox User's Guide Version 3.0, The Math Works, Inc. 1998.

[11] Funahashi Ken–Ichi, On the Approximate Realization of Continuous Mappings by Neural Networks, Neural Networks, Vol. 2, 1989, pp. 183–192.

[12] Goldberg, D.E., Genetic Algorithms in Search, Optimization and Machine Learning, Addison–Wesley Publishing Company, Inc., 1989.

[13] Gonzalez, R.C., Woods, R.E., Digital Image Processing, Addison–Wesley Publishing Company, Inc., 1993.

[14] Handroos, H.M., A Method for Postulating Flexible Models for Individual Components for the Fluid Power Circuit Simulation, ASME, FLUCOM'91, San Francisco, August 29–31, 1991.

[15] Haykin Simon, Neural networks: a comprehensive foundation, Macmillan College Publishing Co., New York, 1994.

[16] Hiller Manfred, Modelling, simulation and control design for large and heavy manipulators, Robotics and Autonomous Systems, Vol. 19, 1996, pp. 167–177.

[17] Jin, B., Guez, A., The Trajectory Planning and Tracking of Redundant Manipulators by a Hierarchical Neurocontroller, Proceedings – IEEE International Conference on Robotics and Automation 1995. Vol. 3, pp. 2490–2495.

[18] Kohonen Teuvo, Hynninen Jussi, Kangas Jari, Laaksonen Jorma. SOM_PAK: The Self–Organizing Map Program Package, Technical Report A31, Helsinki University of Technology, Finland, 1996.

[19] Kohonen Teuvo, Hynninen Jussi, Kangas Jari, Laaksonen Jorma, Torkkola Kari. LVQ_PAK: The Learning Vector Quantization Program Package, Technical Report A30, Helsinki University of Technology, Finland, 1996.

[20] Kohonen Teuvo, Oja Erkki, Simula Olli, Visa Ari and Kangas Jari, Engineering Applications of the Self–organizing Map, Proceedings of the IEEE, Vol. 84, No. 10, October 1996.

[21] Koikkalainen Pasi (ed.), Neuraalilaskennan mahdollisuudet, TEKES julkaisu 43/94, Helsinki, Finland, 1994.

[22] Kosko B., Neural networks and fuzzy systems: a dynamical systems approach to machine intelligence, Englewood Cliffs (NJ) Prentice–Hall cop. 1992.

[23] Kuroe, Y., Nakai, Y., Mori, T., A New Neural Network Learning of Inverse Kinematics of Robot Manipulator, The IEEE 1994 International Conference on Neural Networks, pp. 2819–2824.

[24] Kuroe, Y., Nakai, Y., Mori, T., A Neural Network Learning of Nonlinear Mappings with Considering Their Smoothness and Its Application to Inverse Kinematics, Proceedings of IECON '94 : Twentieth Annual International Conference on Industrial Electronics, Control and Instrumentation 1994. Pp. 1381–1386.

[25] Lampinen J., Advances in Neural Network Modeling, Proc. TOOLMET'97, Tool Environments and Development Methods for Intelligent Systems, April 17–18, 1997, Oulu, Finland, pp. 28–36.

[26] Lampinen J. and Selonen A., Using Background Knowledge in Multilayer Perceptron Learning, Proc. of The 10th Scandinavian Conference on Image Analysis, Vol. 2, 1997, pp. 545–549.

[27] Lampinen Jouko, Taipale Ossi, Optimization and Simulation of Quality Properties in Paper Machine with Neural Networks, Proc. IEEE World Congress on Computational Intelligence, Orlando, Florida, June 28 – July 2, 1994, pp. 3812–3815.

[28] Lippmann Richard, An Introduction to Computing with Neural Nets, IEEE ASSP Magazine, April 1987, pp. 4–22.

[29] McKerrow, P.J., Introduction to robotics, Addison–Wesley Publishing Company, Inc. 1991.

[30] Meirovitch, L., Computational methods in structural dynamics, Sijthoff & Noordhoff, 1980.

[31] Mikkola Aki, Studies on fatigue damage in a hydraulically driven boom system using virtual prototype simulations, Ph.D. Thesis, Lappeenranta University of Technology Research Papers No. 61, Lappeenranta, 1997.

[32] Mikkola Aki, Hydraulikäyttöisen puomin dynaamiset rasitukset, MSc. Thesis. Lappeenranta University of Technology 1994.

[33] Mikkola, A., Handroos, H., Modelling and Simulation of a Flexible Hydraulically Driven Log Crane, Proceedings of TheNinth Bath International Fluid Power Workshop – Fluid Power Systems, University of Bath, Great Britain, 9 – 11 September, 1996, pp. 431–443.

[34] Miller, T.W., Sutton, R.S., Werbos, P.J., (editors), Neural Networks for Control, Cambridge (MA) MIT Press corp., 1990.

[35] Moody, J.E., The Effective Number of Parameters: An Analysis of Generalization and Regularization in Nonlinear Learning Systems, Advances in Neural Information Processing Systems 4, Morgan Kaufman Publishers, 1992, pp. 847–854.

[36] Nearchou Andreas, Solving the Inverse Kinematics Problem of Redundant Robots Operating in Complex Environments via a Modified Genetic Algorithm, Mechanism and Machine Theory, Vol. 33, No. 3, 1998, pp. 273–292.

[37] Nikravesh, E.P., Computer–Aided Analysis of Mechanical Systems, Prentice–Hall, Inc., 1988.

[38] Oja, E., Neural Networks and Pattern Recognition, Course 595, CEI – Europe Courses in Advanced Technology, November 13 – 15, 1995, Cambridge.

[39] Riedmiller Martin, Advanced Supervised Learning in Multi–layer Perceptrons – From Backpropagation to Adaptive Learning Algorithms, International Journal of Computer Standards and Interfaces, Special Issue on Neural Networks (5), 1994.

[40] Rouvinen Asko, Hydraulijärjestelmien mallintaminen ADAMS–ohjelmistossa, MSc. Thesis. Lappeenranta University of Technology 1995.

[41] Rouvinen Asko, Handroos Heikki, Robot Positioning of a Flexible Hydraulic Manipulator Utilizing Genetic Algorithm and Neural Networks, Proceedings of Fourth Annual Conference on Mechatronics and Machine Vision in Practice, Toowoomba, Australia, September 23–25, 1997, pp. 182–187.

[42] Rumelhart, D.E., McClelland, J.L. and the PDP Research Group, Parallel Distributed Processing, Vol. 1, MIT Press corp., 1986.

[43] Salama Ramari, Owens Robyn, Evolving Neural Controllers for Robot Manipulators, Proceedings of 4th Int. Conf. on Genetic Algorithms and Neural Networks, East Anglia, April 1997.

[44] Schneider Martin, Hiller Manfred, Modelling, Simulation and Control of a Large Hydraulically Driven Redundant Manipulator with Flexible Links, Proceedings of the 9th World Congress on Machines and Mechanisms, 1995. Pp. 3038–3043.

[45] Selonen A., Lampinen J. and Ikonen L., Using Background Knowledge in Neural Network Learning, Proc. SPIE on Intelligent Robots and Computer Vision XV: Algorithms, Techniques, Active Vision, and Materials Handling, Vol. 2904, 1996, pp. 239–249.

[46] Shabana, A.A., Dynamics of multibody systems, John Wiley & Sons, 1989.

[47] Smagt, P.P. van der, Kröse, B.J.A., A Real–Time Learning Neural Robot Controller, Proceedings of the 1991 International Conference on Artificial Neural Networks, pp. 351–356, North–Holland/Elsevier Science Publishers, June 1991.

[48] Snyder W.E., Industrial robots: computer interface and control, Prentice–Hall, Inc., 1985.

[49] Surdilovic, D., Vukobratovic, M., Deflection Compensation for Large Flexible Manipulators, Mechanism and Machine Theory, Vol. 31, No. 3, 1996, pp. 317–329.

[50] Watanabe, E., Shimizu, H., A Study on Generalization Ability of Neural Network for Manipulator Inverse Kinematics, Proceedings of the 17th International Conference on Industrial Electronics, Control and Instrumentation – IECON '91, pp. 957–962.

[51] Williams, D.W., Turcic, D.A., An Inverse Kinematic Analysis Procedure for Flexible Open–Loop Mechanisms, Mechanism and Machine Theory, Vol. 27, No. 6, 1992, pp. 701–714.

[52] Wu, G., Wang, J., A Recurrent Neural Network for Manipulator Inverse Kinematics Computation, The IEEE 1994 International Conference on Neural Networks, pp. 2715–2720.

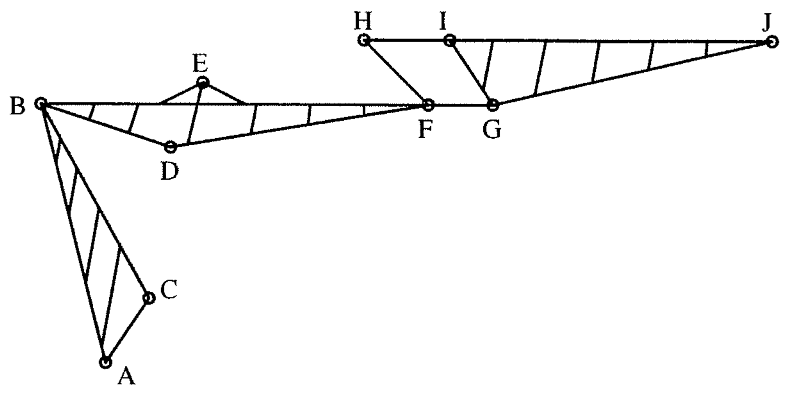
[53] Xi F., Fenton R.G., A Quasi–Static Motion Planner for Flexible Manipulators Using the Algebra of Rotations, Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California – April 1991.

[54] Zalzala, A.M.S., Morris, A.S., Neural networks for robotic control: theory and applications, Ellis Horwood Limited, 1996.

[55] Zurada, J.M., Introduction to artificial neural systems, West Publishing Company, 1992.

Kinematics of the crane

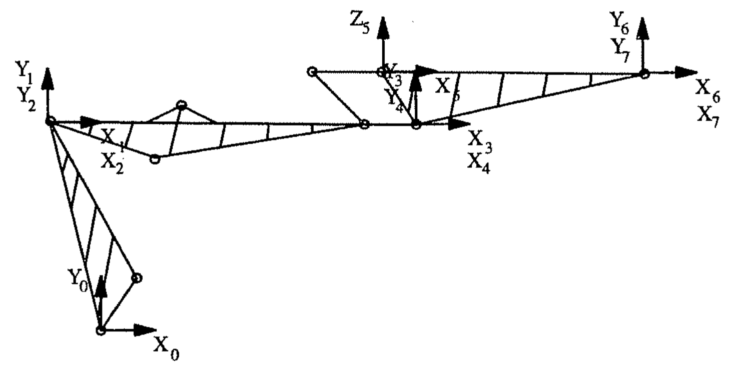
The kinematic dimensions of the crane can be found in the following table. The dimensions are in Cartesian coordinates. Used subscriptions are shown in the figure below.



Vector	length (m)	Vector	length (m)
AB_x	0.09	BD_y	1.04
AB_y	1.115	BD_x	0.3025
BG_x	2.88	EF_y	0.203
BG_y	0.024	EF_x	1.4165
GI_x	0.095	FG_y	0.015
GI_y	0.24	FG_x	0.19
IJ_x	2.141	FH	0.45
IJ_y	0.008	HI	0.48
AC_x	0.26	CD_0	0.8
AC_y	0.105	EH_0	1.037

Kinematic model of the crane

The local coordinate system used in the solution of the rigid kinematics of the crane is shown below. The corresponding transformation matrices are also presented.



$$T_0^1 = \begin{bmatrix} 1 & 0 & 0 & -0.09 \\ 0 & 0 & -1 & 1.115 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_1^2 = \begin{bmatrix} cos(\theta_1) & -sin(\theta_1) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -sin(\theta_1) & -cos(\theta_1) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_2^3 = \begin{bmatrix} 1 & 0 & 0 & 2.88 \\ 0 & 0 & -1 & -0.024 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_3^4 = \begin{bmatrix} cos(\theta_2) & -sin(\theta_2) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -sin(\theta_2) & -cos(\theta_2) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_4^5 = \begin{bmatrix} 1 & 0 & 0 & -0.095 \\ 0 & 0 & -1 & 0.24 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_5^6 = \begin{bmatrix} 1 & 0 & 0 & 2.141 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -0.008 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_6^7 = \begin{bmatrix} 1 & 0 & 0 & s_3 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The MATLAB code of rigid kinematic model of the boom is presented below.

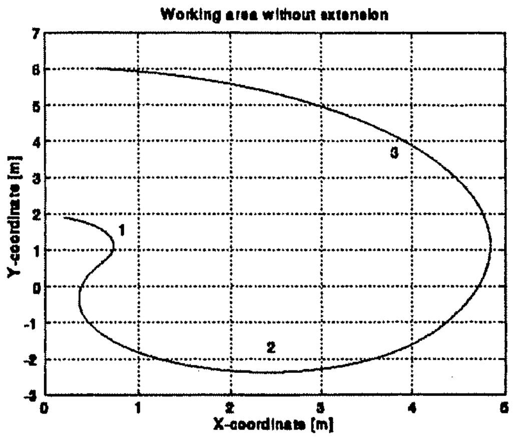
```
function [x,y]=mitpatu(s1,s2,s3)

%      mitpatu(s1,s2,s3)
%
% Mitpatu draws the position of Patu 655 log crane and returns the x- and y-coordinates
% of the boom end, input values are the actuator strokes
%
%      where s1  = Lift cylinder stroke (0.. 0.535)
%            s2  = Jib cylinder stroke (0.. 0.75)
%            s3  = Extension cylinder stroke (0.. 1.6)
%
%      Written by Asko Rouvinen 1997
%
```

```
T01=[1 0 0 -0.09;0 0 -1 1.115;0 1 0 0;0 0 0 1];
T23=[1 0 0 2.88;0 0 -1 -.024;0 1 0 0;0 0 0 1];
T45=[1 0 0 -0.095;0 0 -1 .24;0 1 0 0;0 0 0 1];
T56=[1 0 0 2.141;0 1 0 0;0 0 1 -0.008;0 0 0 1];
T67=[1 0 0 s3;0 0 1 0;0 -1 0 0;0 0 0 1];

a11=.260;
b11=1.040;
a12=.105;
b12=.3025;

a21=0.203;
b21=1.4165;

a23=0.015;
b23=0.19;

N1=0.45;
N3=0.48;
N4=0.2581;

N11=sqrt(a11^2+b11^2);
N12=sqrt(a12^2+b12^2);
epsi1=atan(a11/b11);
epsi2=atan(a12/b12);

N5=sqrt(a21^2+b21^2);
N2=sqrt(a23^2+b23^2);
epsi3=atan(a21/b21);
epsi8=atan(a23/b23);
epsi4=atan(.095/.24);

nisku=s1+.8;
tisku=s2+1.037;

nkulma=acos((nisku^2-N11^2-N12^2)/(-2*N11*N12))+epsi1+epsi2-pi/2;

T12=[cos(nkulma) -sin(nkulma) 0 0;0 0 1 0;-sin(nkulma) -cos(nkulma) 0 0;0 0 0 1];

epsi6 = acos((tisku^2-N5^2-N1^2)/(-2*N5*N1));
beta = pi-epsi3-epsi6-epsi8;
xxx=sqrt(N1^2+N2^2-2*N1*N2*cos(beta));
epsi7 = acos((N1^2-xxx^2-N2^2)/(-2*xxx*N2));
```

```
epsi5 = acos((N3^2–xxx^2–N4^2)/(–2*xxx*N4));

tkulma = (–pi/2–epsi8+epsi7+epsi5+epsi4);

T34=[cos(–tkulma) –sin(–tkulma) 0 0;0 0 1 0;–sin(–tkulma) –cos(–tkulma) 0 0;0 0
0 1];

p1=T01*T12*T23;
p2=T01*T12*T23*T34*T45;
p3=T01*T12*T23*T34*T45*T56;
p4=T01*T12*T23*T34*T45*T56*T67;

x=p4(1,4);
y=p4(2,4);

tulx=[0 T01(1,4) p1(1,4) p2(1,4) p3(1,4) p4(1,4)];
tuly=[0 T01(2,4) p1(2,4) p2(2,4) p3(2,4) p4(2,4)];

plot(tulx,tuly,'g');axis([–.1 7 –2 7]);grid
hold on
plot(p1(1,4),p1(2,4),'ro',p2(1,4),p2(2,4),'go',p4(1,4),p4(2,4),'yo')

grid on
hold off
```

Heuristic solution for the extension cylinder stroke

The method used for reducing the degrees of freedom of the manipulator was based on the idea that the extension cylinder should always be as little extended as possible. The working area of the manipulator, where the extension cylinder stroke is zero, can be described using three circles as shown in the figure below.



The working area which can not be achieved without extension can be recognized using the equation of the circle

$$(x - x_0)^2 + (y - y_0)^2 = (r^2 + s3^2)$$

where x_0 and y_0 are the coordinates of the centerpoint of the circle, x and y are the desired coordinates, r is the base circle and $s3$ is the extension, and heuristic limits that can be found out by studying the maximum and minimum coordinates of each circle. For circles no: 1 and 3 the centerpoint is the coordinate point of the joint of the lift arm. For circle no: 2 the centerpoint is the coordinate point of the joint of the jib arm in the position where the lift actuator has the minimum stroke.

For example the geometric and heuristic rules for the area outside circle no: 1 in the figure above can be expressed as:

if (sqrt((x+0.09)**2+(y–1.115)**2) is less than 0.829

and

x is less than 0.829–0.09

The value 0.829 is the distance from the centerpoint of the circle to the end–effector of the boom when the jib cylinder is fully extended and the stroke of the extension cylinder is zero.

When the part of the working area is recognized, it is possible to solve the required magnitude of the extension

$$s3 = \sqrt{(x - x_0)^2 + (y - y_0)^2} - r$$

In the case of the area outside circle no:1, the stroke of the extension cylinder is:

```
s3=-(sqrt((x+0.09)**2+(y-1.115)**2)-0.829)
```