We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

6,900
Open access books available

185,000
International authors and editors

200M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities



WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com



5

Recurrent Neural Network with Human Simulator Based Virtual Reality

Yousif I. Al Mashhadany
Al Anbar University, Engineering College, Electrical Dept., Iraq

1. Introduction

During almost three decades, the study on theory and applications of artificial neural network has increased considerably, due partly to a number of significant breakthroughs in research on network types and operational characteristics, but also because of some distinct advances in the power of computer hardware which is readily available for net implementation. In the last few years, recurrent neural networks (RNNs), which are neural network with feedback (closed-loop) connects, have been an important focus of research and development. Examples include bidirectional associative memory (BAM), Hopfield, cellular neural network (CNN), Boltzmann machine, and recurrent back propagation nets, etc.. RNN techniques have been applied to a wide variety of problems due to their dynamics and parallel distributed property, such as identifying and controlling the real-time system, neural computing, image processing and so on.

RNNs are widely acknowledged as an effective tool that can be employed by a wide range of applications that store and process temporal sequences. The ability of RNNs to capture complex, nonlinear system dynamics has served as a driving motivation for their study. RNNs have the potential to be effectively used in modeling, system identification, and adaptive control applications, to name a few, where other techniques may fall short. Most of the proposed RNN learning algorithms rely on the calculation of error gradients with respect to the network weights. What distinguishes recurrent neural networks from static, or feedforward networks, is the fact that the gradients are time dependent or dynamic. This implies that the current error gradient does not only depend on the current input, output, and targets, but rather on its possibly infinite past. How to effectively train RNNs remains a challenging and active research topic.

The learning problem consists of adjusting the parameters (weights) of the network, such that the trajectories have certain specified properties. Perhaps the most common online learning algorithm proposed for RNNs is the real-rime recurrent learning (RTRL), which calculates gradients at time (k) in terms of those at time instant (k-1). Once the gradients are evaluated, weight updates can be calculated in a straightf__Gorward manner. The RTRL algorithm is very attractive in that it is applicable to real-time systems. However, the two main drawbacks of RTRL are the large computational complexity of $O(N^4)$ and, even more critical, the storage requirements of $O(N^3)$, where N denotes the number of neurons in the network.

RNNS are mathematical abstractions of biological nervous systems that can perform complex mappings from input sequences to output sequences. In principle one can wire them up just like microprocessors, hence RNNs can compute anything a traditional computer can compute. In particular, they can approximate any dynamical system with arbitrary precision. However, unlike traditional, programmed computers, RNNs *learn* their behavior from a training set of correct example sequences. As training sequences are fed to the network, the error between the actual and desired network output is minimized using gradient descent, whereby the connection weights are gradually adjusted in the direction that reduces this error most rapidly. Potential applications include adaptive robotics, speech recognition, attentive vision, music composition, and innumerably many others where retaining information from arbitrarily far in the past can be critical to making optimal decisions. Recently, *Echo State Networks* ESNs and a very similar approach, *Liquid State Machines*, have attracted significant attention. Composed primarily of a large pool of hidden neurons (typically hundreds or thousands) with fixed random weights, ESNs are trained by computing a set of weights from the pool to the output units using fast, linear regression. The idea is that with so many random hidden units, the pool is capable of very rich dynamics that just need to be correctly "tapped" by setting the output weights appropriately. ESNs have the best known error rates on the Mackey-Glass time series prediction task.(Abraham, 2005)

Two main methods exist for providing a neural network with dynamic behavior: the insertion of a buffer somewhere in the network to provide an explicit memory of the past inputs, or the implementation of feedbacks. As for the first method, it builds on the structure of feed forward networks where all input signals flow in one direction, from input to output. Then, because a feed forward network does not have a dynamic memory, *tapped-delay-lines* (temporal buffers) of the inputs are used. The buffer can be applied at the network inputs only, keeping the network internally static as in the buffered multilayer perceptron (MLP) or at the input of each neuron as in the MLP with Finite Impulse Response (FIR) filter synapses (FIRMLP). The main disadvantage of the buffer approach is the limited past-history horizon, which needs to be used in order to keep the size of the network computationally manageable, thereby preventing modeling of arbitrary long time dependencies between inputs and outputs it is also difficult to set the length of the buffer given a certain application.

The second method, the most general example of implementation of feedbacks in a neural network is the fully recurrent neural network constituted by a single layer of neurons fully interconnected with each other or by several such layers. Because of the required large structural complexity of this network, in recent years growing efforts have been propounded in developing methods for implementing temporal dynamic feedback connections into the widely used multi-layered feed forward neural networks. Recurrent connections can be added by using two main types of recurrence or feedback: *external* or *internal*. *External recurrence* is obtained for example by feeding back the outputs to the input of the network as in NARX networks; *internal recurrence* is obtained by feeding back the outputs of neurons of a given layer in inputs to neurons of the same layer, giving rise to the so called *Locally Recurrent Neural Networks* (*LRNNs*) (Francesco et al., 2006)

The major advantages of LRNNs with respect to the buffered, tapped-delayed feedforward networks and to the fully recurrent networks are:

1. The hierarchic multilayer topology which they are based on is well known and efficient.
2. The use of dynamic neurons allows limiting the number of neurons required for modeling a given dynamic system, contrary to the tapped-delayed networks.
3. The training procedures for properly adjusting the network weights are significantly simpler and faster than those for the fully recurrent networks.

This chapter consists of the following items:

- Types of RNN.
- Some Special Recurrent Networks
- Training algorithms for recurrent neural networks
- Inverse kinematic For Humanied manipulator with 27-DOFs
- Solution of IKP by using RNN
- Simulation of the humaniod manipulator based upon Virtual Reality
- Conclusion

2. Types of RNN

The categories of RNN is very difficult where many parameters can be division according to consideration, therefore the division will explain by:

2.1 Types of RNN according to the performance of NN

There are many classes of RNN architectures. All architectures can be best described using the state-space model from systems theory. This state-space model is explain in many references (Cheron et al., 2007) (Cruse, 2006)(Dijk, 1999), the following architectures or classes of architectures are presented:

- Fully Recurrent Neural networks (FRNN).
- Subsets of FRNN: Recurrent Neural Networks (RNN).
- Partially Recurrent Networks (PRN).
- Simple Recurrent Networks (SRN).

These architectures emerge by applying constraints to the general state-space model. The architectures have been investigated and tested in applications by many researchers. In the following subsections, these specific constraints will be listed and the resulting architectures will be discussed. Each class is presented together with a quick look at some properties and examples of their application.

The architectures treated can be ordered hierarchically since some architecture is special cases of more general architectures. This hierarchy is visualized in Fig.1. The most general architectures are at the left, specific architectures are at the right. The accolades show what architectures are parts of a more general architecture description.

2.1.1 Fully recurrent neural networks (FRNN)

The Fully Recurrent Neural Network (FRNN) is first described here in terms of individual neurons and their connections, as was done in [Williams e.a., 1989]. Then the FRNN is considered as a special case of the general state-space model and a convenient matrix notation of the network is given.

The name Fully Recurrent Neural Network for this network type is proposed by [Kasper e.a., 1999]. Another name for this type of network is the .Real-Time Recurrent Network.. This name will not be used further, because the name strongly implies that training is accomplished using the Real-Time Recurrent Learning (RTRL) algorithm proposed for this network in [Williams e.a., 1989] which is not necessarily the case because other algorithms can be used. In general a FRNN has N neurons, M external inputs and L external outputs. In Fig.2 an example of a FRNN is given which has N=4 neurons, M=2 external inputs u 1(*n*), u2(*n*) and L=2 external outputs y1(*n*), y2(*n*).(Dijk, 1999).

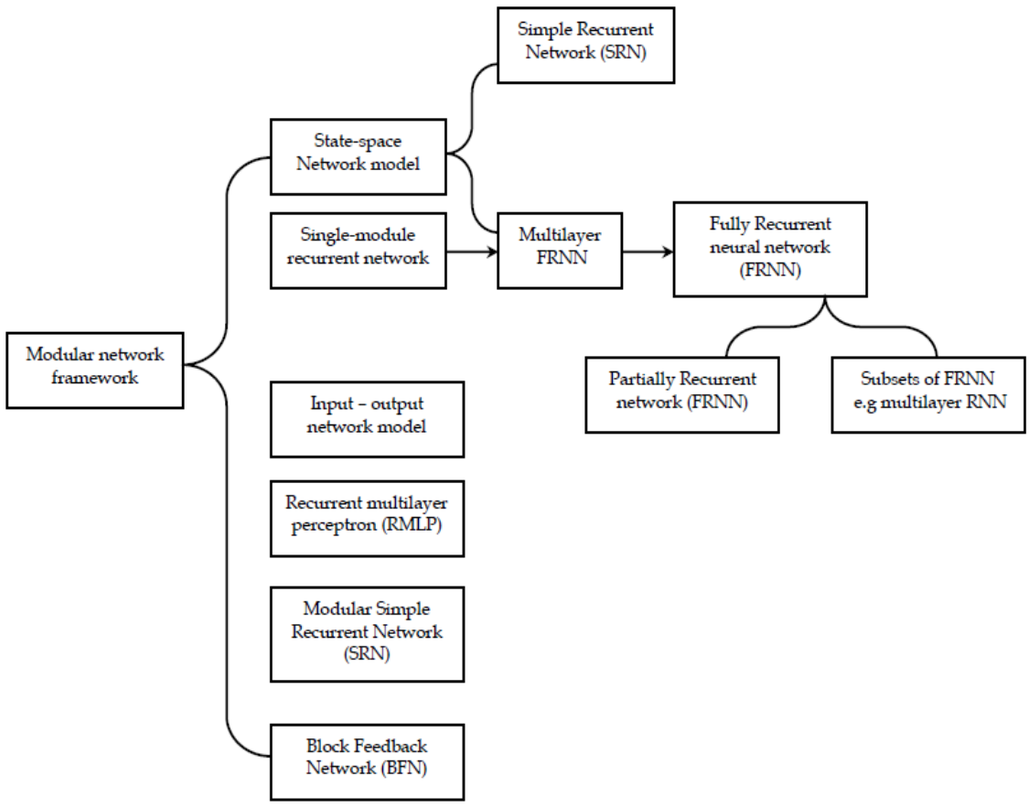


Fig. 1. Recurrent neural network architectures hierarchy (numbers indicate sections)

The network is called Fully Recurrent because the output of all neurons is recurrently connected (through N delay elements and N2 weighted feedback connections) to all neurons in the network. The external network inputs are connected to the neurons by N*M feed forward connections *without* delay element. A bias (also called threshold) can be introduced for every neuron by applying a constant external input u1(*n*) = 1 to the network.

For static neural networks, the number of layers in the neural network can be clearly defined as the number of neurons an input signal passes through before reaching the output. For the FRNN however the same definition is ambiguous, because signals applied at time *n* are fed back and reach the output at times *n*, *n*+1, and so on. The term layer therefore appears to be never used in literature in FRNN descriptions. By redefining the concept of layer to: the *minimum* number of neurons an input signal passes through before reaching the output, a workable definition is obtained for the FRNN.

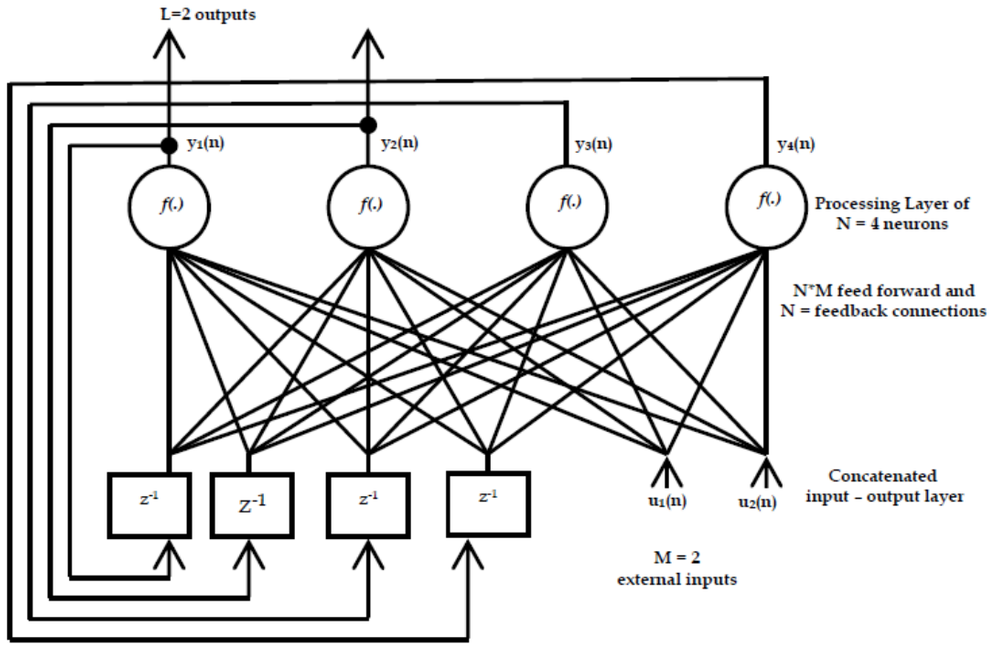


Fig. 2. Example of a fully recurrent neural network (of type 1) (Dijk, 1999).

2.1.2 Subsets of FRNN: Recurrent neural networks (RNN)

Additional restrictions can be imposed on the FRNN architecture described in the previous subsection to create other (restricted) Recurrent Neural Network (RNN) architectures. This subsection will describe some of these restricted architectures. Because the FRNN can be written as a state-space model, all .subsets. of FRNN are in many cases most conveniently written as state-space models.

The following categories of restrictions can be used (individually or in a combination):

1. forcing certain weights to zero (called removing or *pruning* the weight)
2. forcing weights to non-zero value (called *fixing* the weight or making the weight non-learnable)
3. forcing weights to be equal to other weights (called *sharing* of weights) or approximately equal to other weights (called *soft sharing* of weights)

These restrictions will be looked at in this subsection. Note that the three restrictions listed are fairly general and can be applied to other neural networks architecture than the FRNN, for example to the standard feedforward network.

All three restrictions have a property in common: the number of free parameters of the network is reduced when compared to a non-modified FRNN. Reasons for doing so will be given now. More reasons for applying restrictions will be given in the category descriptions. The training of a neural network is in fact a procedure that tries to estimate the parameters (weights) of the network such that an error measure is minimized. Reducing the number of parameters to be estimated may simplify training. Another good reason for reducing the number of free parameters is to reduce training algorithm overhead, which often grows quickly for an increasing number of weights NW.(Cruse, 2006).

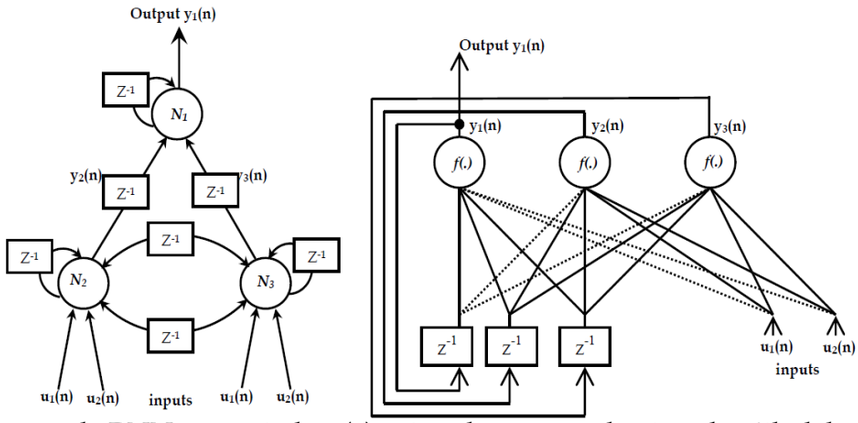


Fig. 3. One example RNN presented as (a). a two-layer neural network with delays and recurrent connections; and (b) as a FRNN with removed connections.

2.1.3 Partially recurrent networks (PRN)

The output vector (*n*) of the FRNN consists of the first L elements of the state vector [(*n*), as was shown in Fig. 2. So the output signals are a .subset. of state signals. In a general state space description this is certainly not the case, the output is determined by a separate calculation (the output equation) which is some function of the external input and the state. To obtain a network that effectively has separate state and output units (analogous to a state space system that has separate process and output equations), the feedback connections from all L output neurons yi(*n*) with *i*=1.L are removed. An example of the *partially recurrent neural network* (PRN) [Robinson e.a., 1991], also named the *simplified recurrent neural network* [Janssen, 1998], that results is shown in Fig. 4. The name partially recurrent neural network. will be used in this report to avoid confusion in the terms simple/simplified recurrent networks in the next subsection.

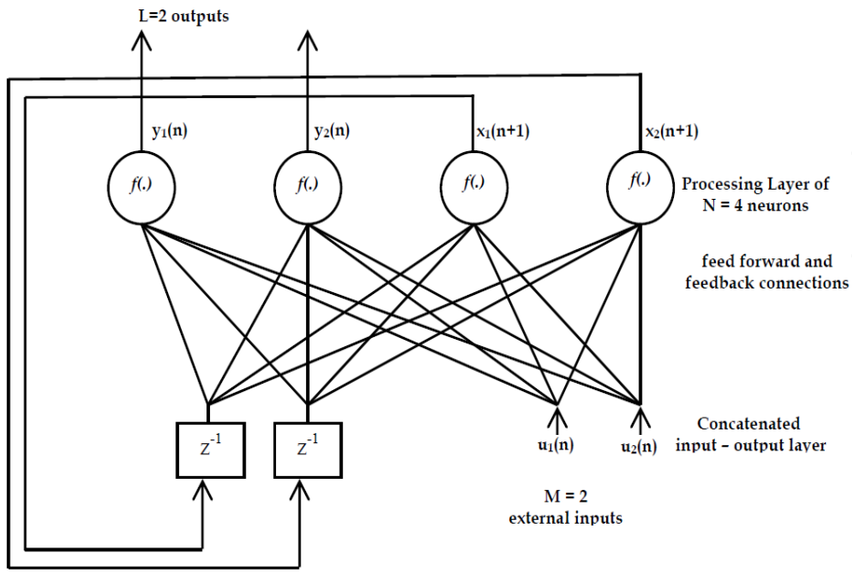


Fig. 4. Example of a partially recurrent neural network

2.2 Types of RNN according to NN structure

Some neural network architectures can be best described as modular architectures. The definition of a modular architecture as used in this report is: an architecture that consists of several static neural networks, that are interconnected in a specific way. There is, in most cases, not a clear boundary between a modular network and a single neural network because the total modular architecture can be looked at as a single neural network, and some existing single networks can also be described as modular networks. It is rather a convenient way of describing complex neural networks.(Paine W. Rainer & Tani Jun, 2004).

In this section the category of modular *recurrent* neural network architectures is looked at, modular architectures that all have one or more internal feedback connections. The modular recurrent neural network architectures were not introduced in previous sections, because they do not fit very well in the state-space system description or the NARX description.

Formally they can be described as a state-space system (like any dynamic system) but this could result in a very complicated and unnecessarily large state-space system description. In this section three classes of modular recurrent neural network architectures are presented:

- Recurrent Multi-layer Perceptron (RMLP).
- Block Feedback Networks (BFN) framework.
- General modular neural network framework.

2.2.1 Recurrent multi-layer perceptron (RMLP)

The first model (RMLP) is a rather specific one and it is included as an example of a modular architecture. Undoubtedly, many more such architectures are proposed in literature and they cannot all be listed here. Another example is the Pipelined Recurrent Neural Network found in [Haykin, 1998] and applied to speech prediction in [Baltersee e.a., 1998]. The second model is far more general and was meant to provide a structured way to describe a large class of recurrent neural networks and their training algorithms. The third model attempts to do the same and it turns out that this model is the most general one: it incorporates the first two as special cases, so in this section the attention will be mainly focussed on the third model, the general modular network framework. An extension of the regular MLP has been proposed by Puskorias e.a. (see [Haykin, 1998]) which adds self-feedback connections for each layer of the standard MLP. The resulting Recurrent Multilayer Perceptron (RMLP) structure with N layers is shown in Fig 5.

Each layer is a standard MLP layer. The layer outputs are fed forward to the inputs of the next layer and the delayed layer outputs are fed back into the layer itself. So the layer output of time n-1 for a certain layer acts as the state variable at time n for this layer. The global state of the network consists of all layer states [i(n) together. Effectively, this type of network can have both a very large total state vector and a relatively small number of parameters because the neurons in the network are not fully interconnected. There are no recurrent interconnections across layers. All recurrent connections are local (1-layer-to-itself).(Sit, 2005).

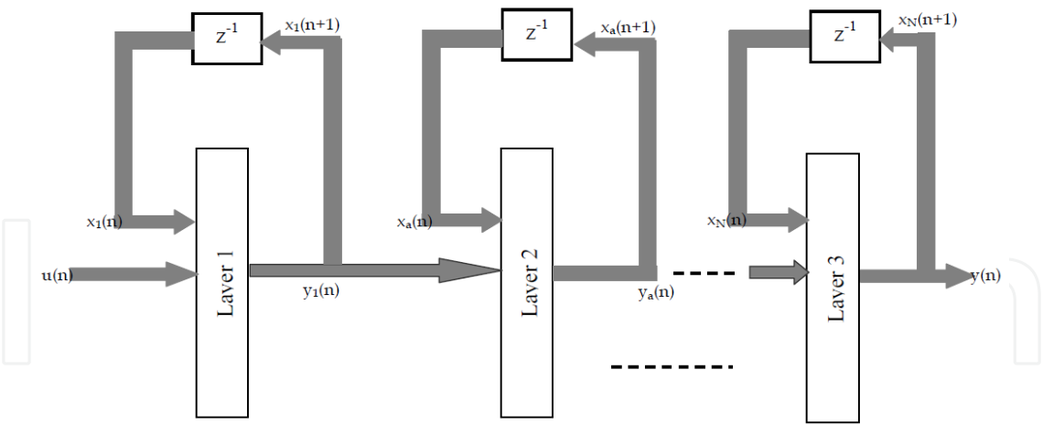


Fig. 5. Recurrent multi-layer perceptron (RMLP) architecture with N layers

2.2.2 Block feedback networks (BFN) framework

A framework for describing recurrent neural networks that has been introduced by [Santini e.a., 1995b] provides a systematic way for modular design of networks of high complexity. This class of networks is called Block Feedback Neural Networks (BFN), referring to the *blocks* that can be connected to each other using a number of elementary connections. The term *feedback* is used because one of the elementary connections is a feedback connection, thus enabling the construction of recurrent neural networks. The network that results from the construction can in turn be considered a .block. and it can be used again as a basic building block for further construction of progressively more complex networks.

So a recursive, modular way of designing networks is provided. The training algorithm for any BFN is based on backpropagation training for MLP and the backpropagation through time (BPTT) algorithm for recurrent networks. It is recursively constructed along with the network structure. So the BFN framework introduces a class of (infinitely many) recurrent networks, which can be trained using a correspondingly constructed backpropagation algorithm. The basic unit is a single neural network layer, an example of which is shown in Fig 6.a. The corresponding matrix notation is shown in Fig 6.b. $ is a 6-by-3 matrix and)(.) is a 6-by-6 diagonal mapping containing the neuron activation functions (of the form of equation 1). One such layer is defined as a *single layer block* N.(Zhenzhen Liu & Itamar Elhanany, 2008)

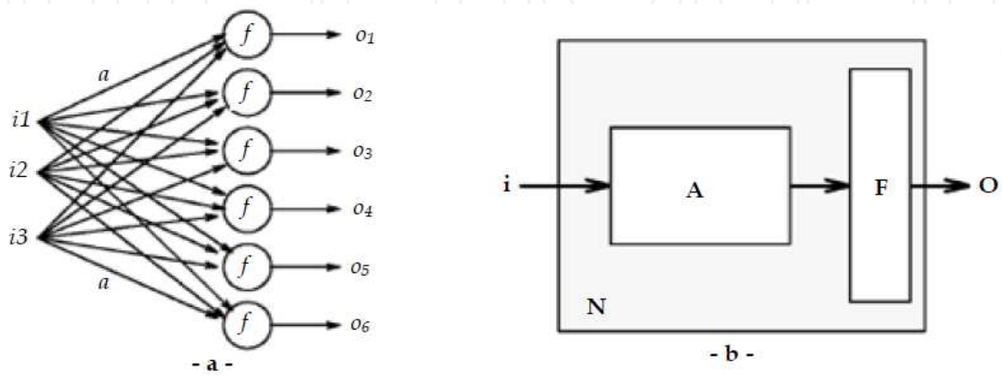


Fig. 6. a) example of a single network layer ; b) the layer in BFN block notation as a block N

This block computes the function:

$$o(n) = F(A.i(n)) \tag{1}$$

Single layer blocks can be connected together using the four elementary connections shown in Fig. 7. They are called the cascade, the sum, the split and the feedback connection. Each of these connections consists of one or two embedded BFN blocks (these are called N1 and N2 in the figure) and one connection layer (which has the structure of a single-layer block). This connection layer consists of the weight matrices $ and %, and the vector function)(.). Each of the four elementary connections itself is defined as a block and can therefore be used as the embedded block of yet another elementary connection.

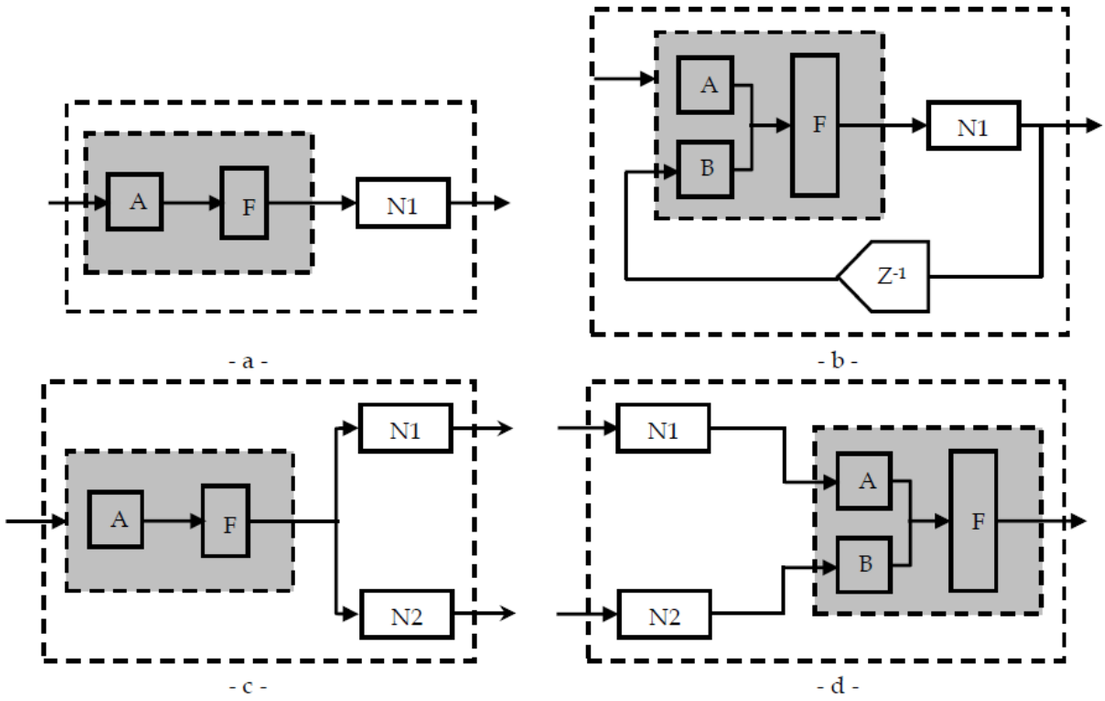


Fig. 7. The four elementary BFN connections: the cascade (a), feedback (b), split (c) and sum (d) connection.

2.2.3 General modular neural network framework

In [Bengio, 1996] a general modular network framework is introduced. This framework is similar to the BFN framework: it can be used to describe many different *modular networks* which are built of neural network modules linked together. Each module is a static feedforward neural network and the links between the modules can incorporate delay elements. The purpose of this framework is:

- to describe many types of recurrent neural networks as modular networks
- to derive a training algorithm for a network that is described as a modular network

The use of the framework for developing such a training algorithm, that can do *joint training* of the individual network modules.(Dijk O. Esko, 1999).

3. Some special recurrent networks

There are many special recurrent networks according to purpose of application or structure of networks. From these types can be explain the following:

3.1 Elman nets and Jordan nets

As was mentioned earlier, recurrent networks represent the most general format of a network. However, there is yet no general theoretical framework that describes the properties of recurrent nets. Therefore, several networks will be discussed below which have a specifically defined structure. In the first example, most layers have only feedforward connections, and only one contains specified recurrent connections. This example is given by Elman (1990) (Fig. 8.**a**). The system has an input layer, a hidden layer, and an output layer all of which are connected in a feedforward manner. The hidden layer, however, is not only connected to the output layer but also, in a simple 1 : 1 connection, to a further layer called the context layer. To form recurrent connections, the output of this context layer is also inputted to the hidden layer. Except for these 1 : 1 connections from hidden to context layer, the weights of which are fixed to 1, all other layers may be fully connected and all weights may be modifiable. The recurrent connections of the context layer provide the system with a short-term memory; the hidden units do not only observe the actual input but, via the context layer, also obtain information on their own state at the last time step. Since, at a given time step, hidden units have already been influenced by inputs at the earlier time steps, this recurrency comprises a memory which depends on earlier states (though their influence decays with time).(Cruse Holk, 2006).

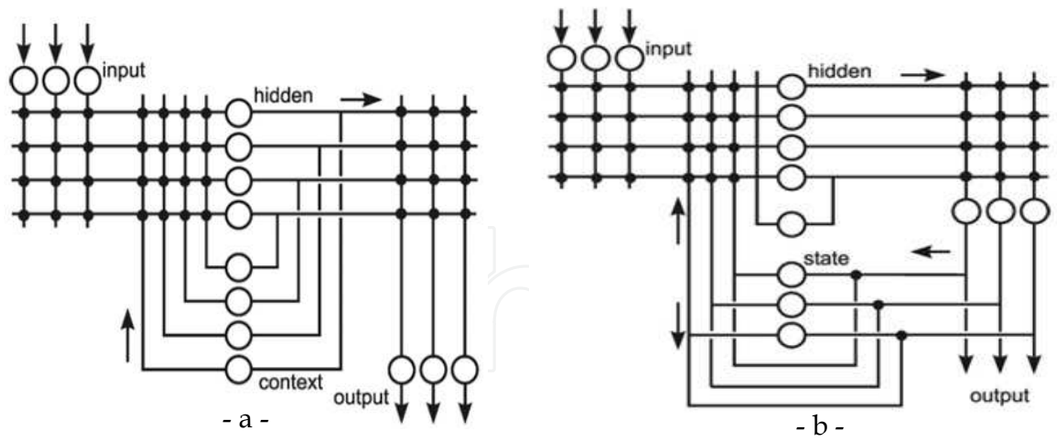


Fig. 8. Two networks with partial recurrent connections. (a) Elman net. (b) Jordan net. Only the weights in the feed forward channels can be modified.

Recurrent networks are particularly interesting in relation to motor control. For this purpose, Jordan (1986) proposed a very similar net (Fig. 8.**b**). A difference is that the recurrent connections start from the output, rather than the hidden layer. Furthermore, the layer corresponding to the context, here called state layer, comprises a recurrent net itself with 1:1 connections and fixed weights.

Another difference is that the network was used by Jordan so that a constant input vector is given to the net, and the output of the net performs a temporal sequence of vectors. The variation in time is produced by the two types of recurrent connections, namely those from the output layer to the state layer and those within the state layer. For each input vector, another temporal sequence can be produced. However, as the Elman net and the Jordan net are quite similar, each can be used for both purposes. Both types of networks have the advantage that only weights in the forward connections are modifiable and therefore no special training methods for recurrent nets have to be introduced.

3.2 Echo state networks

When we are interested to construct a recurrent neural network that shows a complex, but given dynamic behavior, back propagation through time can be applied as described above. A much simpler solution, however, is to use echo state networks (Jaeger and Haas 2004). An echo state network consists of two parts, a recurrent network with fixed weights, called dynamic reservoir, and output units that are connected to the neuroses of the dynamic reservoir. Only one output unit is depicted in Fig. 9 for simplicity.

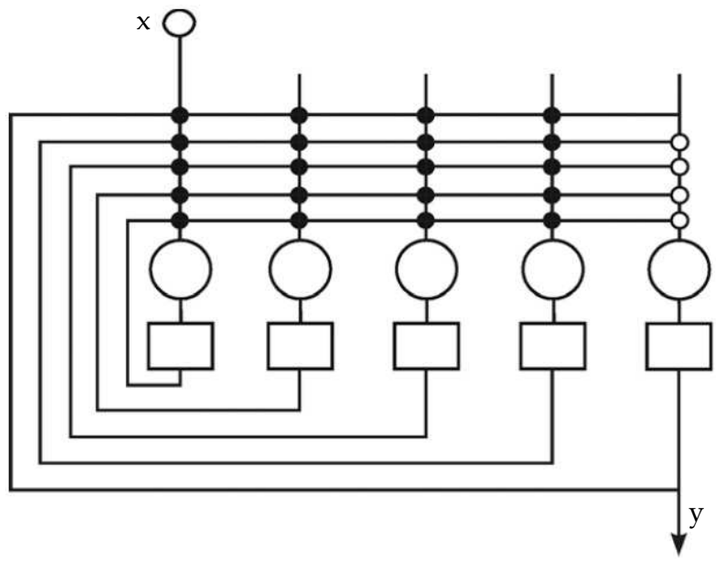


Fig. 9. Echo state network.

The dynamic reservoir consists of recurrently, usually sparsely connected units with logistic activation functions. The randomly selected strengths of the connections have to be small enough to avoid growing oscillations (this is guaranteed by using a weight matrix with the largest eigenvector smaller than 1). To test this property, the dynamic reservoir, after being excited by an impulse-like input to some of the units, may perform complex dynamics which however should decrease to zero with time. These dynamics are exploited by the output units. These output units are again randomly and recurrently connected to the units of the dynamic reservoir. Only those weights that determine the connections from the dynamic reservoir to the output units are learnt. All other weights are specified at the beginning and then held fixed (Hafizah et al., 2008).

3.3 Linear differential equations and recurrent neural networks

With respect to their dynamical properties, recurrent neural networks may be described as showing fixed point attractors. How is it possible to design a neural network with specific dynamics? Dynamical systems are often described by differential equations. In such cases the construction of a recurrent network is easily possible: Any system described by a linear differential equation of order n can be transformed into a recurrent neural network containing n units (Nauck et al., 2003). To this end, the differential equation has first to be transferred into a system of n coupled differential equations of the order one.

3.4 MMC nets

A specific type of recurrent neural networks that show fixed-point attractors and that are particularly suited to describe systems with redundant degrees of freedom are the so called MMC nets. The easiest way to construct such a net is to start with a simpler version. Given a linear equation with n variables

Each of these n equations represents the computation performed by one neuroid. So the complete network represents Multiple Solutions of the Basis Equation, and is therefore termed MSBE net. Different to Hopfield nets, the weights are in general asymmetric (apart from the special case that all parameters a are identical, i.e. a1 = a2 = a3), but follow the rule wij = 1/wji . The diagonal weights, by which each unit excites itself, could be zero, as in the example, or any positive value di, if all weights of this equation are further normalized by multiplication with 1/(di+1). Positive diagonal weights influence the dynamics to adopt low-pass filter-like properties, because the earlier state of this unit is transferred to the actual state with the factor d/(d+1). As di can arbitrarily be chosen (di ≤ 0), the weights may then not follow the rule wij = 1/wji anymore. Starting this net with any vector **a**, the net will stabilize at a vector fulfilling the basic equation. This means that the attractor points form a smooth, in this example two-dimensional, space. This is another difference to Hopfield nets which show discrete attractor points. Furthermore, there are no nonlinear characteristics necessary.

This network can be expanded to form an MMC net. MMC nets result from the combination of several MSBE nets (i.e. several basis equations) with shared units. Such MMC nets can be used to describe landmark navigation in insects and as a model describing place cells found in rodents (Cruse 2002). However, the principle can also be used to represent the kinematics of a body with complex geometry (Steinkühler and Cruse 1998). As a simple example, we will use a three-joint arm that moves in two dimensional space, therefore having one extra degree of freedom (Fig. 10.**a**)(Le Yang &Yanbo Xue, (2009).

The properties of this type of network can best be described by starting with a simple linear version (Fig. 10.**b**). As we have a two-dimensional case, the complete net consists of two identical networks. The output values correspond to the Cartesian coordinates of the six vectors shown in Fig.10.**a**, the x coordinates of the vectors given by the net shown with solid lines, the y coordinates by dashed lines. To obtain the weights, vector equations drawn from the geometrical arrangement shown in Fig. 10.**a** are used as basis equations. This means in this case there are several basis equations possible. For example, each three vectors forming a triangle can be used to provide a basis equation (e.g. **L**1 + **L**2 - **D**1 = 0). As a given variable (e.g. **L**1) occurs in different basis equations, there are several equations to determine this variable.

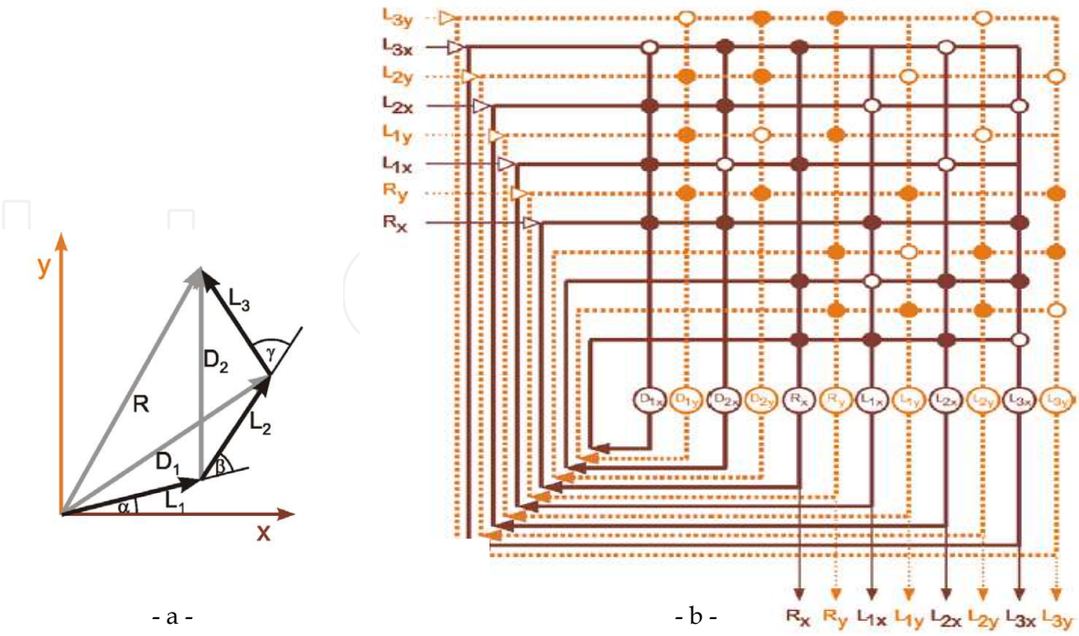


Fig. 10. An arm consisting of three segments described by the vectors L1 L2, and L3 which are connected by three planar joints.

3.5 Forward models and inverse models

Using recurrent neural networks like Jordan nets or MMC nets for the control of behavior, i.e. use their output for direct control of the actuators. Several models are proposed to control the movement of such a redundant (or non-redundant) arm. One model corresponds to a schema shown in Fig 11.**a**, where DK and IK may represent feedforward (e. g., three-layer) networks, computing the direct (DK) and inverse kinematics (IK) solutions, respectively. (In the redundant case, IK has to represent a particular solution)(Liu Meiqin, 2006).

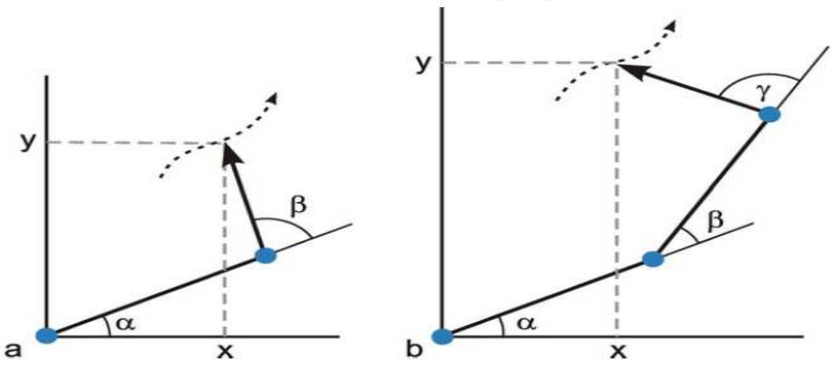


Fig. 11. A two-joint arm (a) and a three-joint arm (b) moving in a two dimensional (x-y) plane. When the tip of the arm has to follow a given trajectory (dotted arrow), in the redundant case (b) an infinite number of joint angle combinations can solve the problem for any given position of the tip.

Depending on the sensory system applied, the DK system might belong to the process, for example when the position of the tip of the arm is registered visually, or it might be part of the neuronal system, for example if joint angles are monitored. In principle, the process can be controlled with this feedback network. However, since in biological systems information transfer is slow, this solution is not appropriate for fast movements because long delays can lead to unstable behavior. A further complication of the control task occurs if there is a process with non negligible dynamic properties. This means that computation is more complex because not only the kinematics but also the forward and inverse dynamics have to be determined.

A simple solution of this problem would be to expand the system by introducing an internal (neuronal) model of the process (Fig. 12.b, FM). By this, the behavior of the process can be predicted in advance, e. g., inertia effects could be compensated for before they occur. This prediction system is called forward model (FM). Usually, although depicted separately in Figure 12.a, in this case the DK is considered part of the forward model. In other words, the FM predicts the next state of the arm (e.g. position, velocity) if the actual state and the motor command is known. Note that the network consists of serially connected IK and FM (including DK) forming a loop within the system and can therefore be represented by a recurrent neural network. The additional external feedback loop is omitted in Fig.12.b, but is still necessary if external disturbances occur. It should be mentioned that the output of the FM, i.e. the predicted sensory feedback, could also be compared with the real sensory feedback. This could be used to distinguish sensors effects produced by own actions from those produced by external activities.(Karaoglan D. Aslan, 2011).

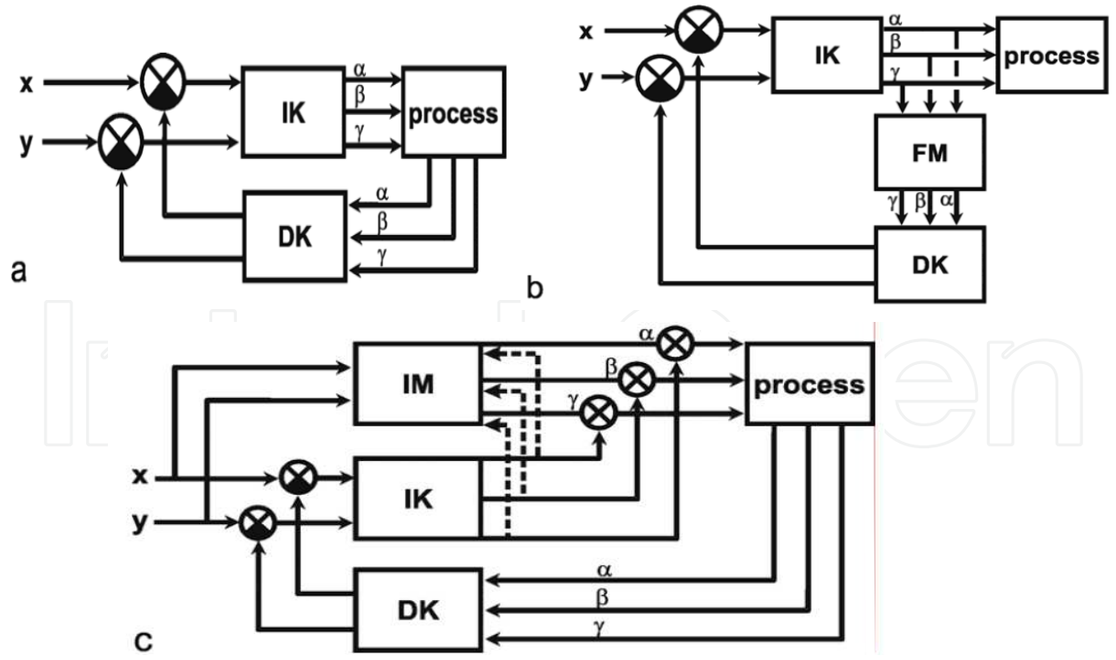


Fig. 12. Three ways of controlling the movement of a redundant (or non redundant) arm as shown in Fig. 11. DK, IK: networks solving the direct or inverse kinematics, respectively. FM: forward model of the process. IM: inverse model of the process. The dashed lines in (c) denote the error signals used to train the inverse model

An even faster way to solve this problem would be to introduce the inverse model (IM), which corresponds to the inverse kinematics, but may also incorporate the dynamic properties of the process (Fig. 11.c, upper part). Given the desired new state of the arm, the inverse model does provide the necessary motor command (**x** -> α) to reach this state. Using such an inverse model, the process could be controlled without the need of an internal feedback loop.

4. Training algorithms for recurrent neural networks

A training algorithm is a procedure that adapts the free parameters of a neural network in response to the behavior of the network embedded in its environment. The goal of the adaptations is to improve the neural network performance for the given task. Most training algorithms for neural networks adapt the network parameters in such a way that a certain error measure (also called cost function) is minimized. Alternatively, the *negative* error measure or a *performance measure* can be maximized.

Error measures can be postulated because they seem intuitively right for the task, because they lead to good performance of the neural network for the task, or because they lead to a convenient training algorithm. An example is the Sum Squared Error measure, which always has been a default choice of error measure in neural network research. Error measures can also be obtained by accepting some general.

Different error measures lead to different algorithms to minimize these error measures. Training algorithms can be classified into categories depending on certain distinguishing properties of those algorithms. Four main categories of algorithms that can be distinguished are:(Dijk, 1999)

1. *gradient-based algorithms.* The gradient of the equation of the error measure with respect to all network weights is calculated and the result is used to perform *gradient descent*. This means that the error measure is minimized in steps, by adapting the weight parameters proportional to the negative gradient vector.
2. *second-order gradient-based algorithms.* In second-order methods, not only the first derivatives of the error measure are used, but also the second-order derivatives of the error measure.
3. *stochastic algorithms*. Stochastic weight updates are made but the stochastic process is directed in such a way, that on average the error measure becomes smaller over time. A gradient of the error measure is not needed so an expression for the gradient doesn't have to exist.
4. *hybrid algorithms*. Gradient-based algorithms are sometimes combined with stochastic elements, which may capture the advantages of both approaches.

The following algorithms for the Fully Recurrent Neural Network and the subsets of the FRNN are investigated in this item:

4.1 Back propagation through time (BPTT) algorithm for FRNN

The Back propagation Through Time (BPTT) algorithm is an algorithm that performs an exact computation of the gradient of the error measure for use in the weight adaptation. In

this section the BPTT algorithm will be derived for a (type 1) FRNN using a Sum Squared Error measure.(Omlin, 1996)

Methods of derivation of the algorithm

There are two different methods to develop the BPTT algorithm. Both are shown in this report:

- derivation by unfolding the network in time, which also gives intuitive insight in how the algorithm works.
- a formal derivation of the algorithm using the *ordered derivative* notation.

The BPTT algorithm can be summarized as follows:

1. set initial time n = n0
2. calculate the N neuron output values for time n using the network.
3. recursively calculate ei(m) then di(m) with backwards in time starting with m= n back to m = n0.
4. calculate for all i,j the weight updates .
5. update the weights wij
6. increase time n to n+1 and go back to step 2

4.2 Real-time recurrent learning (RTRL) algorithm for FRNN

A real-time training algorithm for recurrent networks known as Real-time Recurrent Learning (RTRL) was derived by several authors [Williams e.a., 1995]. This algorithm can be summarized by the explanation as:

In deriving a gradient-based update rule for recurrent networks, we now make network connectivity very very unconstrained. We simply suppose that we have a set of input units, $I = \{x_k(t), 0<k<m\}$, and a set of other units, $U = \{y_k(t), 0<k<n\}$, which can be hidden or output units. To index an arbitrary unit in the network we can use

$$z_k(t) = \begin{cases} x_k(t) & if \quad k \in I \\ y_k(t) & if \quad k \in U \end{cases} \tag{2}$$

Let **W** be the weight matrix with n rows and n+m columns, where $w_{i,j}$ is the weight to unit i (which is in U) from unit j (which is in I or U). Units compute their activations in the now familiar way, by first computing the weighted sum of their inputs:

$$net_k(t) = \sum_{I \in U \cup I} w_{kI} z_I(t) \tag{3}$$

where the only new element in the formula is the introduction of the temporal index t. Units then compute some non-linear function of their net input

$$y_k(t+1) = f_k(net_k(t)) \tag{4}$$

Usually, both hidden and output units will have non-linear activation functions. Note that external input at time t does not influence the output of any unit until time t+1. The network is thus a discrete dynamical system. Some of the units in U are output units, for

which a target is defined. A target may not be defined for every single input however. For example, if we are presenting a string to the network to be classified as either grammatical or ungrammatical, we may provide a target only for the last symbol in the string. In defining an error over the outputs, therefore, we need to make the error time dependent too, so that it can be undefined (or 0) for an output unit for which no target exists at present. Let $T(t)$ be the set of indices k in U for which there exists a target value $d_k(t)$ at time t. We are forced to use the notation d_k instead of t here, as t now refers to time. Let the error at the output units be and define our error function for a single time step as

$$e_k(t) = \begin{cases} d_k(t) - y_k(t) & if \quad k \in T(t) \\ 0 & otherwise \end{cases} \tag{5}$$

The error function we wish to minimize is the sum of this error over all past steps of the network

$$E_{total}(t_o, t_1) = \sum_{\tau=to+1}^{t1} E(\tau) \tag{6}$$

Now, because the total error is the sum of all previous errors and the error at this time step, so also, the gradient of the total error is the sum of the gradient for this time step and the gradient for previous steps

$$\nabla_w E_{total}(t_o, t+1) = \nabla_w E_{total}(t_o, t) + \nabla_w E(t+1) \tag{7}$$

As a time series is presented to the network, we can accumulate the values of the gradient, or equivalently, of the weight changes. We thus keep track of the value

$$\Delta w_{ij}(t) = -\mu \frac{\partial E(t)}{\partial w_{ij}} \tag{8}$$

After the network has been presented with the whole series, we alter each weight w_{ij} by

$$\sum_{t=t_0+1}^{t1} \Delta w_{ij}(t) \tag{9}$$

We therefore need an algorithm that computes

$$-\frac{\partial E(t)}{\partial w_{ij}} = -\sum_{k \in U} \frac{\partial E(t)}{\partial y_k(t)} \frac{\partial y_k(t)}{\partial w_{ij}} = \sum_{k \in U} e_k(t) \frac{\partial y_k(t)}{\partial w_{ij}} \tag{10}$$

at each time step t. Since we know $e_k(t)$ at all times (the difference between our targets and outputs), we only need to find a way to compute the second factor

$$\frac{\partial y_k(t)}{\partial w_{ij}} \tag{11}$$

The key to understanding RTRL is to appreciate what this factor expresses. It is essentially a measure of the sensitivity of the value of the output of unit k at time t to a small change in the value of w_{ij}, taking into account the effect of such a change in the weight over the entire network trajectory from t_0 to t. Note that w_{ij} does not have to be connected to unit k. Thus this algorithm is non-local, in that we need to consider the effect of a change at one place in the network on the values computed at an entirely different place. Make sure you understand this before you dive into the derivation given next.(Baruch, 1999)

5. Inverse kinematic for humanied manipulator with 27-DOFs

Humanoid manipulators are the type of manipulator that practically suitable to coexist with human in builtfor-human environment because of its anthropomorphism, human friendly design and locomotion ability. Humanoid manipulator is different compare to other types of manipulators because the physical structure is designed to mimic as much as human's physical structure. Humanoid's shape shares many basic physical characteristics with actual humans, and for this reason, they are expected to coexist and collaborate with humans in environments where humans work and live. They may also be substituted for humans in hazardous environments or at disaster sites. These demands make it imperative for humanoid manipulators to attain many sophisticated motions such as walking, climbing stairs, avoiding obstacles, crawling, etc.

The *model* was designed in virtual reality to mimic as much as human characteristic, especially for contribution of its joints. The manipulator is consists of total of 27-DOFs: six for each leg, three for each arm, one for the waist, and two for the head. The high numbers of DOF's provide the ability to realize complex motions. Furthermore, the configuration of joints that closely resemble those of humans provides the advantages for the humanoid manipulator to attain human-like motion. Each joint feature a relatively wide range of rotation angles, shown in Table 1, particularly for the hip yaw of both legs, which permits the legs to rotate through wide angles when correcting the manipulator's orientation and avoiding obstacles. The specification of each joint rotation range is considered factors such as correlation with human's joint rotation range, manipulability of humanoid's manipulator, and safety during performing motions.(Nortman, 2001)

In this chapter, we propose and implement a simplified approach to solving inverse kinematics problems by classifying the robot's joints into several groups of joint coordinate frames at the robot's manipulator [11]. To describe translation and rotational relationship between adjacent joint links, we employ a matrix method proposed by Denavit-Hartenberg [12], which systematically establishes a coordinate system for each link of an articulated chain in the robot body.

Kinematical Solutions for 6-dof Arm

The humanoid manipulator design has 6-DOF on each arm: 3-DOF (yaw, roll and pitch) at the shoulder joint, one DOF (roll) at the elbow joint and 2-DOF ((pitch and yaw) at the wrist joint. Fig. 13 shows the arm structure and configuration of joints and links. The coordinate orientation follows the right-hand law, and a reference coordinate is fixed at the intersection point of two joints at the shoulder. Fig. 13 displays a model of the arm describing the configurations and orientation of each joint coordinates. To avoid confusion, only the x and z-axes appear in the figure. The arm's structure is divided into seven sets of joint coordinate's frames as listed below:

Σ*0:* Reference coordinate.
Σ*1:* shoulder yaw coordinate.
Σ*2:* shoulder roll coordinate.
Σ*3:* shoulder pitch coordinate.
Σ*4:* elbow pitch coordinate.
Σ*5:* wrist pitch coordinate.
Σ*6:* wrist roll coordinate.
Σ*h:* End-effector coordinate (at the end of middle fingerer).

Axis	Humanoid manipulator (deg)	Human (deg)
Neck (roll and pitch)	-90 ~ 90	-90 ~ 90
Shoulder (pitch) right & left	-180 ~ 120	-180 ~ 120
Shoulder (roll) right/left	-135 ~ 30/-30 ~ 135	-135 ~ 30/-30 ~ 135
Shoulder (yaw) right/left	-90 ~ 90/-90 ~ 90	-90 ~ 90/-90 ~ 90
Elbow (roll) right/left	0 ~ 135/0 ~ -135	0 ~ 135/0 ~ -135
Wrist (pitch) right/left	-30 ~ 60	-30 ~ 60
Wrist (yaw) right/left	-90 ~ 60	-90 ~ 60
Hip (pitch) right & left	-130 ~ 45	-130 ~ 45
Hip (roll) right/left	-90 ~ 22/-22 ~ 90 -	60 ~ 45/-45 ~ 60
Hip (yaw) right/left	-90 ~ 22/-22 ~ 90 -	60 ~ 45/-45 ~ 60
Knee (pitch) right &left	-20 ~150	0 ~150
Ankle (pitch) right & left	-90 ~ 60	-30 ~ 90
Ankle (roll) right/left	-20 ~ 90/-90 ~ 20	-20 ~ 30/-30 ~ 20
Waist (yaw)	-90 ~ 90	-45 ~ 45

Table 1. Comparison Joint rotation range between humanoid manipulator and Human

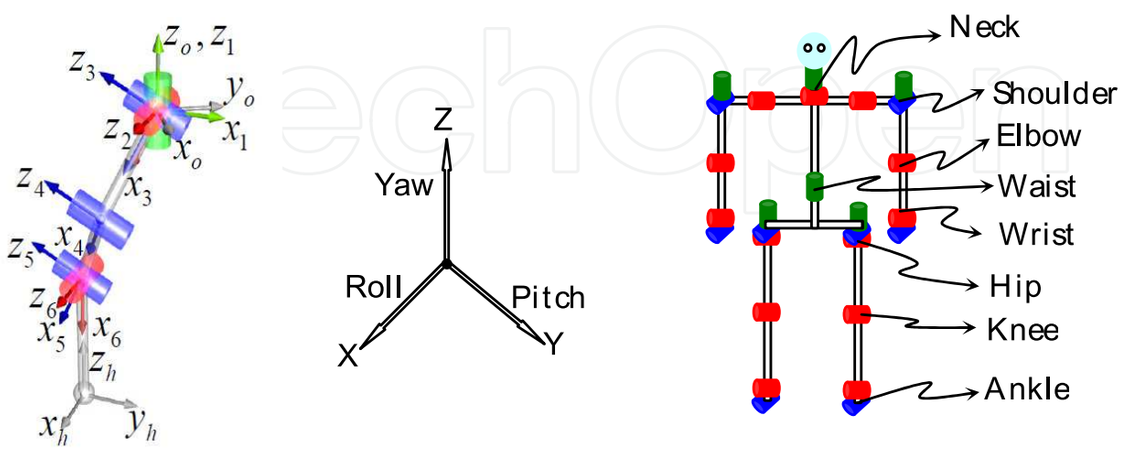


Fig. 13. a. Configurations of joint coordinates at the Manipulator arm with 6-DOF.
b. Structure of humanoid manipulator.

Consequently, corresponding link parameters of the arm can be defined as shown in Table 2. From the Denavit-Hartenberg convention mentioned above, definitions of the homogeneous transform matrix of the link parameters can be described as follows:

$$h_T^0 = Rot(z,\theta)Trans(0,0,d)Trans(l,0,0)Rot(x,\alpha) \tag{12}$$

Here, variable factor θ_i is the joint angle between the x_{i-1} and the x_i axes measured about the z_i axis; d_i is the distance from the x_{i-1} axis to the x_i axis measured along the z_i axis; α_i is the angle between the z_i axis to the z_{i-1} axis measured about the x_{i-1} axis, and l_i is the distance from the z_i axis to the z_{i-1} axis measured along the x_{i-1} axis. Here, link length for the upper and lower arm is described as l_1 and l_2, respectively. The following is used to obtain the forward kinematics solution for the robot arm. (Yussof, 2007)

Link	d_i	a_i	α_i	θ_i
1	d_1	0	$\pi/2$	θ^*
2	0	a_2	0	θ^*
3	0	a_3	0	θ^*
4	0	$-\pi/2$	0	θ^*
5	0	$\pi/2$	0	θ^*
6	d_6	0	0	θ^*

Table 2. DH parameters for the arm of humaniod manipulator.(* ≡ joint variable).

$$A_1 = \begin{bmatrix} c_1 & 0 & s_1 & 0 \\ s_1 & 0 & -c_1 & 0 \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}; A_2 = \begin{bmatrix} c_2 & 0 & s_2 & a_2 \\ s_2 & 0 & -c_2 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} c_3 & 0 & -s_3 & a_3 \\ s_3 & 0 & c_3 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}; A_4 = \begin{bmatrix} c_4 & 0 & -s_4 & 0 \\ s_4 & 0 & c_4 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{13}$$

$$A_5 = \begin{bmatrix} c_5 & 0 & s_5 & 0 \\ s_5 & 0 & -c_5 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}; A_6 = \begin{bmatrix} c_6 & 0 & s_6 & 0 \\ s_6 & 0 & c_6 & 0 \\ 0 & 1 & 0 & d_6 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The inverse kinematic is achieved by closed solution of above eqn's, and the general solution of angles of rotation can be summarized as follows:

$$
\left.
\begin{aligned}
&\theta_1 = a\tan 2(x_c, y_c) \\
&\theta_3 = a\tan(D, \pm\sqrt{1-D^2}) \\
&where: D = \frac{x_c^2 + y_c^2 - d^2 + (z_c - d_1)^2 - a_2^2 - a_3^2}{2a_2a_3} \\
&\theta_2 = a\tan 2(\sqrt{x_c^2 + y_c^2 - d^2}, z_c - d_1) \\
&\theta_4 = a\tan 2(c_1c_{23}r_{13} + s_1c_{23}r_{23} + s_{23}r_{33}, \\
&\qquad -c_1s_{23}r_{13} + s_1s_{23}r_{23} + c_{23}r_{33}) \\
&\theta_5 = a\tan 2(s_1r_{13} - c_1r_{23}, \pm\sqrt{1-(s_1r_{13} - c_1r_{23})^2}) \\
&\theta_6 = a\tan 2(-s_1r_{11} + c_1r_{21}, s_1r_{12} + c_1r_{22}) \\
&where: \\
&c_i = \cos(\theta_i), and \quad s_i = \sin(\theta_i)
\end{aligned}
\right\} \tag{14}
$$

Kinematical Solutions for 6-dof Leg

Each of the legs has 6-DOFs: 3-DOFs (yaw, roll and pitch) at the hip joint, 1-DOF (pitch) at the knee joint and two DOF (pitch and roll) at the ankle joint. A reference coordinate is taken at the intersection point of the 3-DOF hip joint. In solving calculations of inverse kinematics for the leg, just as for arm, the joint coordinates are divided into eight separate coordinate frames as listed bellow.

Σ*0:* Reference coordinate.
Σ*1:* Hip yaw coordinate.
Σ*2:* Hip roll coordinate.
Σ*3:* Hip pitch coordinate.
Σ*4:* Knee pitch coordinate.
Σ*5:* Ankle pitch coordinate.
Σ*6:* Ankle roll coordinate.
Σ*h:* Foot bottom-center coordinate.

Furthermore, the leg's links are classified into three groups to short-cut the calculations, where each group of links is calculated separately as follows:

i. From link 0 to link 1 (Reference coordinate to coordinate joint number 1).
ii. From link 1 to link 4 (Coordinate joint number 2 to coordinate joint number 4).
iii. From link 4 to link 6 (Coordinate joint number 5 to coordinate at the bottom of the foot). Basically, i) is to control leg rotation at the *z*-axis,
iv. is to define the leg position, while iii) is to decide the leg's end-point orientation.

The solution in details is explain in the reference.(Youssof, 2007).

6. Solution of IKP by using RNN

This section introduces the basics of ANN architecture and its learning rule. Inspired by the idea of basing the feed forward and back propagation network structure. Fig.14 shows this structure , the Learning rule which is used in this paper is fast momentum back-propagation

with delta rule structure of network with dimension (I-M-N). The inputs are the position of end-effector in (x,y,z) ,the network is single layer with dimension (33) neurons (this dimension limited by trial and error). The output dimensions are the angles of rotation and translation displacement in joints.

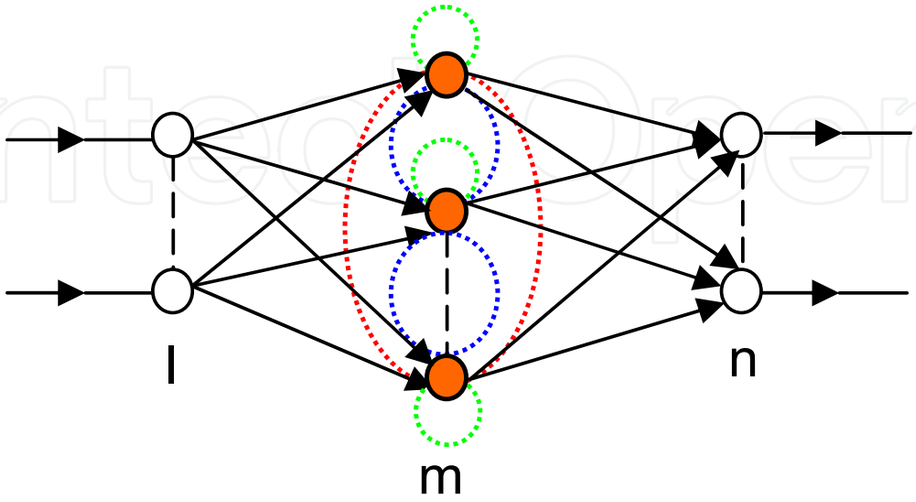


Fig. 14. FRBP Network with I inputs, one hidden layer of (M) unit and (N) outputs.(The dotted curve lines denote the finite recurrent connection)

The majority of adaptation learning algorithms are based on the fast momentum back propagation the mathematical characterization of a multilayer feed forward network is that of a composite application of functions each of these functions represents a particular layer and may be specific to individual units in the layer, e.g. all the units in the layer are required to have same activation function. The overall mapping is thus characterized by a composite function relating feed forward network inputs to output. That is $O=f_{composite}(x)$. Using (p) mapping layers in a (p+1) layer feed forward net yield:

$O=f^{Lp}(f^{Lp-1}(f^{L1}(x).))$. Thus the interconnection weights from unit (k) in L_1 to unit (I) in L_2 are w^{L1-L2} . If hidden units have a sigmoidal activation function, denoted f^{sig} .

$$O_i^{L2} = \sum_{k-1}^{Hi} w_{ik}^{L1 \to L2} \left\{ f_k^{sig} \left[\sum_{j-1}^{I} w_{kj}^{Lo \to L1} i_j \right] \right\} \tag{15}$$

Above equation illustrates neural network with supervision and composition of non-linear function.The learning is a process by which the free parameters of a neural network are adapted through a continuing process of simulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameters changes take place. A prescribed set of well defined rules for the solution of a learning problem is called learning algorithm. The Learning algorithms differ from each other in the way in which the adjustment kj, Δw to the synaptic weight w_{kj} is formulated. The basic approach in learning is to start with an untrained network.

The network outcomes are compared with target values that provide some error. Suppose that $\mathbf{t}_k(n)$ denote some desired outcome (response) for the K^{th} neuron at time n and let the

actual response of the neuron is $O_k(n)$. Suppose the response $y_k(n)$ was produced when x(n) applied to the network. If the actual response $y_k(n)$ is not same as $\mathbf{t}_k(n)$, we may define an error signal as: $\mathbf{e}_k = \mathbf{t}_k(n) - \mathbf{y}_k(n)$. The purpose of error-correction learning is to minimize a cost function based on the error signal $\mathbf{e}_k(n)$. Once a cost function is selected, error-correction learning is strictly an optimization problem.

In case of non-linear neural network, the error surface may have troughs, valleys, canyons, and host of shapes, where as in low dimensional data, contains many minima and so many local minima plague the error landscapes, then it is unlikely that the network will find the global minimum. Another issue is the presence of plateaus regions where the error varies only slightly as a function of weights see and. Thus in presence of many plateaus, training will get slow. To overcome this situation momentum is introduced that forces the iterative process to cross saddle points and small landscapes.

7. Simulation of the humaniod manipulator based upon virtual reality

The humanoid manipulator 27-DOF model is built by using VR environment. It is shown in Fig. 15. The simulation of this model is achieved by solution the IKP with analytical model firstly. The data for analytical solution was using to learning FRNN that is shown in previous section with FRNN structure (15-33-27). The inputs are (I=27) six for each two arms and two lags (three position of end-effecter), one for waist and two for neck. The output of FRNN has dimension (N=27). The outputs are six angles of joint for each the limbs, one for waist and two for neck.

The initial posture form of humanoid manipulator is shown in Fig. 15. After solution of IKP by FRNN and get the joint angles. The values of joint angle were implemented by forward kinematic solution to get the posture of humanoid manipulator. The interaction between Matlab/Simulink and VR model will used the calculation of IKP by FRNN to implement the posture. The overall simulation design is shown in Fig. 16.

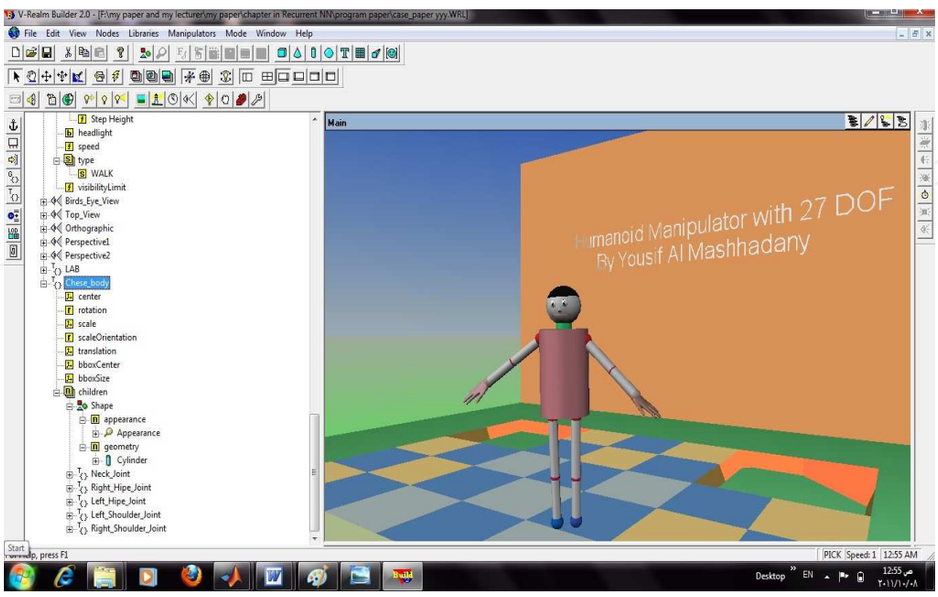


Fig. 15. Humanoid manipulator in VR.

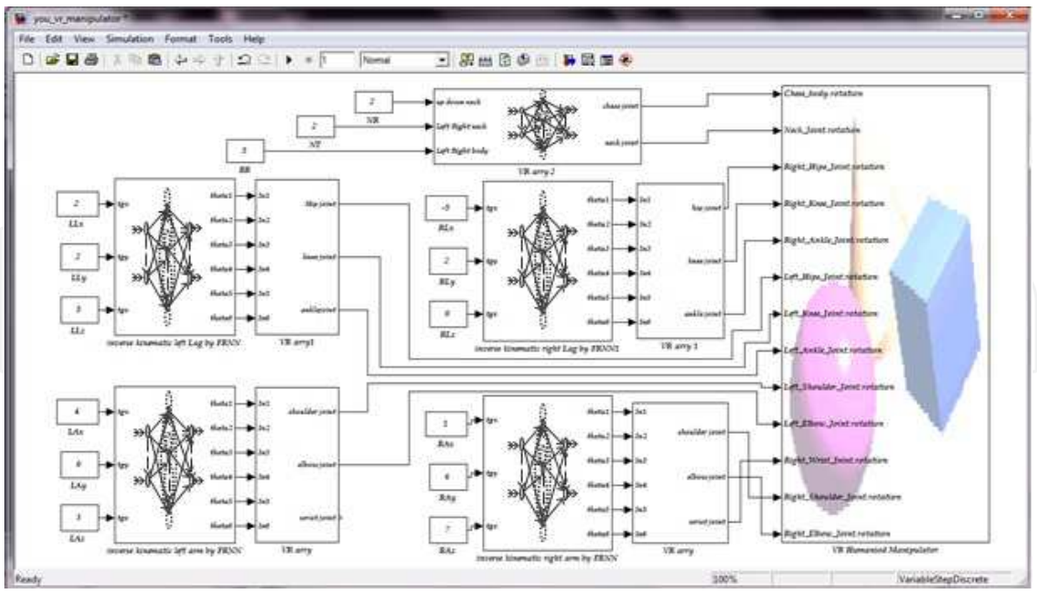


Fig. 16. The overall simulation of Humanoid Manipulator 27-DOF with Virtual Reality

The results of the simulation can be display by two methods. The first by running the design as one iteration by input one set of position and orientation of the limbs where the results is the posture form of humanoid manipulator in VR same as results shown in Fig. 17.a,b,c,d. The second method is achieved by enter many sets of coordinates to design. The results were movie of model in side the VR environment according to the path of inputs coordinate.

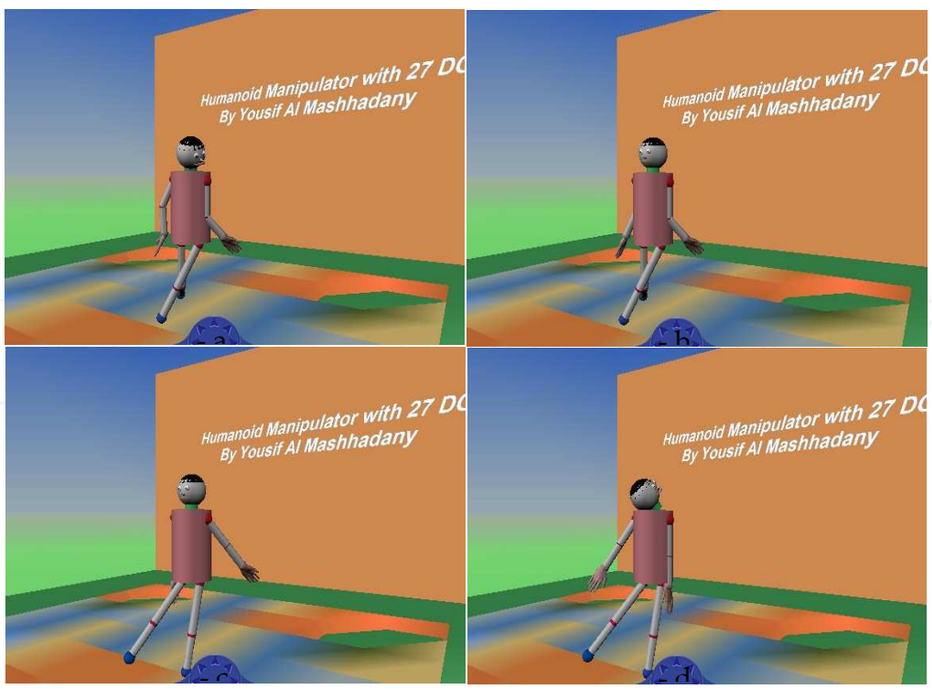


Fig. 17. a, b, c, d. The simulation results for posture of humanoid manipulator 27-DOF. Based upon FRNN with VR model.

8. Conclusion

From the description of the all types of RNN, some of special modified in the main types to get special RNN structure related to special application such as the calculation of system parameters by identification for the states system, recognition of fault with experience system and From the description of main types of training RNN can be seen that the RNN is very flexible structure to apply many mathematical algorithms and implement to solving system problem. The implement of feedback technique with memory in RNN gives the ability to dealing with many problems that needs high calculation with iteration to get the solution.

The solution of IKP is achieved by different method but the powerful method with practice application is the analytical solution. The calculation of posture for human body or manipulator which has the similar kinematic structure with human can be achieved by analytical solution, but this solution is very difficult because of high mathematic and very length calculation. This difficulty is increasing when we need calculated the posture with time to get the movie of manipulator. This problem is solving in this chapter by using FRNN with back propagation training algorithm.

The results of calculation are achieved for each limb separately at beginning such as for arm, lag and neck. After checking the accuracy of results by applying the forward kinematic to get the same coordinate of end- effecter for each part. The overall calculation of IKP is achieved to get the posture of manipulator. The data base for IKP is used to identify the FRNN for all point of envelop for movement where the internal memory and feedback in FRNN assistance this structure to overcame the problems of high calculation in the IKP solution.

The link between Matlab / Simulink and VR environment and the suitable of Matlab to execute any algorithm with high calculation are assistance to implement the posture of manipulator with high accuracy for two cases of running (movies and once posture). *The future* works for this design are:

- The calculation of IKP by RNN can be used as data base for implement the manipulator as practice system with human robot 27 DOF.
- The VR model for manipulator can be used as human – robot interactive system based on pc computer or microcomputer chip.
- Implement the algorithm of IKP solution based FRNN as microprocessor to able used with human robot manufacturing for many applications.

9. References

Abraham Ajith, (2005), 129: Artificial Neural Networks, *Handbook of Measuring System Design*, edited by Peter H. Sydenham and Richard Thorn. 2005 John Wiley & Sons, Ltd. ISBN: 0-470-02143-8. *Oklahoma State University, Stillwater, OK, USA*, From Web: http://www.softcomputing.net/ann_chapter.pdf

Baruch Ieroham, Gortcheva Elena, Thomas Federico & and Ruben Garrido, (1999), A neuro-fuzzy model for nonlinear plants identification, Proceedings of the IASTED International Conference Modeling and Simulation (MS '99), May 5-8, 1999, Philadelphia, Pennsylvania – USA, 291-021 – 326

Cheron G., etl, 2007, Toward an Integrative Dynamic Recurrent Neural Network for Sensorimotor Coordination Dynamics, Recurrent Neural Networks for Temporal Data Processing, 2007

Cruse Holk, 2006, Neural Networks as Cybernetic Systems – 2nd and revised edition, *Brains, Minds & Media,* ISSN 1861-1680, www.brains-minds-media.org

Dijk O. Esko, (1999), Analysis of Recurrent Neural Networks with Application to Speaker Independent Phoneme Recognition, *M.Sc. Thesis, University of Twente, department of Electrical Engineering*, June 1999, No. S&S 023N99

Francesco Cadini, Enrico Zio & Nicola Pedroni, (2008), Recurrent neural network for dynamic rellability analysis, *R&RATA # 2 (Vol.1)* 2008, June, pp 30 - 42

Hafizah Husain, Marzuki Khalid & Rubiyah Yusof, (2008), Direct Model Reference Adaptive Controller Based-On Neural-Fuzzy Techniques for Nonlinear Dynamical Systems, *American Journal of Applied Sciences* , Issue 7, Volume 5 ,2008, pp 769 – 776, ISSN 1546-9239

Huaien Gao, Rudolf Sollacher, (2008), Conditional Prediction of Time Series Using Spiral Recurrent Neural Network, (2008), *proceedings European Symposium on Artificial neural networks,* 23-25 April 2008, ISBN 2-930307-08-0

Le Yang &Yanbo Xue, (2009), Development of A New Recurrent Neural Network Toolbox (RNN-Tool) *,A Course Project Report on Training Recurrent Multilayer Perceptron and Echo State Network,* McMaster university , 01 June 2006

Liu Meiqin, (2006), Delayed Standard Neural Network Models for the Stability Analysis of Recurrent Neural Networks, *International Journal of Computational Intelligence Research.* Vol.2, No. 1 (2006), pp. 10-16, ISSN 0973-1873, Research India Publications http://www.ijcir.info

Karaoglan D. Aslan, (2011), An integrated neural network structure for recognizing autocrrelated and trending processes , *Mathematical and Computational Applications*, Vol. 16, No. 2, 2011, pp. 514-523

Nortman Scott, Peach Jack, Nechyba Michael, Brown S. Louis & Arroyo A., (2001), Construction and Kinematic Analysis of an Anthropomorphic Mobile Robot, Machine Intelligence Laboratory, University of Florida, Gainesville, FL, FCRAR, May 10-11, 2001, FAMU

Omlin W. Christian & Giles C. Lee, (1996), Constructing Deterministic Finite-State Automata in Recurrent Neural Networks, Journal of the ACM, Vol. 43, No. 6, November 1996, pp. 937–972

Paine W. Rainer & Tani Jun, (2004), Adaptive Motor Primitive and Sequence Formation in a Hierarchical Recurrent Neural Network, Paine & Tani 2004: Accepted in SAB2004

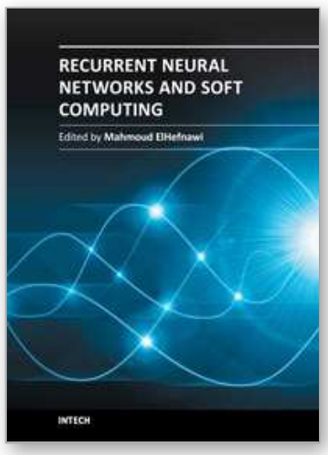
Reza Jafari &Rached Dhaouadi, (Adaptive PID Control of a Nonlinear Servomechanism Using Recurrent Neural Networks, Advances in Reinforcement Learning, PP 275-296

Richards J, Holler P, Bockstahler B, Dale B, Mueller M, Burston J, Selfe J & Levine D, A comparison of human and canine kinematics during level walking, stair ascent, and stair descent, (2010*), Wien. Tierrztl. Mschr. - Vet. Med. Austria 97*, (2010), pp 92 - 100

Sit W.Chen, (2005),Application of artificial neural network-genetic algorithm in inferential estimation and control of a distillation column, Master of Engineering (Chemical) Faculty of Chemical and Natural Resources Engineering Universiti Teknologi Malaysia July 2005

Yussof Hanafiah, Yamano Mitsuhiro, Nasu Yasuo & Ohka Masahiro, (2007), Performance of a Research Prototype Humanoid Robot Bonten-Maru II to Attain Human-Like Motions, *WSEAS Transactions on Systems and Control,* Issue 9, Volume 2, September 2007, PP 458-467, ISSN: 1991-8763

Zhenzhen Liu & Itamar Elhanany, (2008), A Fast and Scalable Recurrent Neural Network Based on Stochastic Meta Descent, *IEEE Transactions on neural networks,* Vol. 19, No. 9, SEPTEMBER 2008, PP 1651 – 1657



Recurrent Neural Networks and Soft Computing
Edited by Dr. Mahmoud ElHefnawi

ISBN 978-953-51-0409-4
Hard cover, 290 pages
Publisher InTech
Published online 30, March, 2012
Published in print edition March, 2012

How to reference
In order to correctly reference this scholarly work, feel free to copy and paste the following:

Yousif I. Al Mashhadany (2012). Recurrent Neural Network with Human Simulator Based Virtual Reality, Recurrent Neural Networks and Soft Computing, Dr. Mahmoud ElHefnawi (Ed.), ISBN: 978-953-51-0409-4, InTech, Available from: http://www.intechopen.com/books/recurrent-neural-networks-and-soft-computing/recurrent-neural-network-with-human-simulator-based-virtual-reality-

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen

© 2012 The Author(s). Licensee IntechOpen. This is an open access article distributed under the terms of the Creative Commons Attribution 3.0 License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

IntechOpen

IntechOpen