This article can be cited as V. M. Aparanji, U. V. Wali and R. Aparna, Multi-Layer Auto Resonance Network for Robotic Motion Control, International Journal of Artificial Intelligence, vol. 18, no. 1, pp. 19-44, 2020.

Multi-Layer Auto Resonance Network for Robotic Motion Control

Vadanical Mathada Aparanji[1], Udayakumar Veerappa Wali[2], Ramalingappa Aparna[3]

[1]Dept of E&CE, Siddaganga Institute of Technology, Tumakuru, Karnataka, India
Email: vmasit@sit.ac.in
[2]Dept of E&CE, SG Balekundri Institute of Technology, Belagavi, Karnataka, India
Email: udaywali@gmail.com
[3]Dept of ISE, Siddaganga Institute of Technology, Tumakuru, Karnataka, India
Email: raparna@sit.ac.in

ABSTRACT

Recent developments in machine learning and artificial intelligence have evoked interest in many research areas considered as NP Hard. Humanoid motion is one such area. Controlling robots with large number of mechanical joints poses challenges due to non-linearity of displacement, redundant configurations, dynamic user environments, etc. Analytical and approximate solutions to Inverse kinematic problems of typical industrial robots with up to 6 Degrees of Freedom (DoF) have been presented in literature. Humans use more than a hundred joints for locomotion, each with one to three degrees of freedom increasing the complexity beyond comprehension. Therefore algorithmic and even heuristic approaches have not been successful in humanoid structures. Recently, machine learning and artificial intelligence are being used for robotic applications. This paper reports a new type of Artificial Neural Network (ANN) called Auto-Resonance Network (ARN) derived from synergistic control of biological joints. The network can be tuned to any real valued input without any degradation in learning ability. Due to the approximating nature of ARN, neuronal density of the network is low and grows at a linear or low order polynomial rate with input classification. Input coverage of the neuron can be tuned dynamically to match properties of input data. ARN can be used as a part of hierarchical structures to support deep learning applications. As a case study, the network has been used to generate optimal path for locomotion avoiding obstacles. The network can optimize the length of traverse and resolution during training much like a biological system. The network is able to learn without supervision taking cues from the environment.

Keywords: Artificial intelligence, artificial neural network, auto resonance network, deep neural network, Hebbian learning, hierarchical neural networks, machine learning, multi-layer neural networks, robotic motion.

Computing Classification System: Computer systems organization~Neural networks, Computing methodologies~Motion path planning

1. INTRODUCTION

Creating a machine that works like a human being has been a dream of mankind since the beginning of civilization. A humanoid is expected to learn from the environment and be able to take autonomous decisions. Therefore, adaptation of Artificial Neural Networks (ANN) for controlling a humanoid is a natural choice. All higher vertebrates use a musculoskeletal assembly containing a large number of bone joints. Each bone is attached to several muscle groups through serially connected tendons in protagonist and antagonist configurations. Muscles can only pull or relax, but not push. Each such group can be approximated to have one Degree of Freedom (DoF). Providing a single muscle for each direction of motion represents a minimal musculoskeletal configuration of a joint.

Considering that there are more than 100 joints related to gross locomotion of the human body (3 pelvic, 24 spinal, approx. 4x20per limb), with most of them capable of limited motion in two primary planes, we could approximate a total of 200 DoF for robotic system. On the other hand, advanced humanoids use far fewer DoF, e.g., 34 in Honda's Asimo. Analytical Inverse Kinematic (IK) solutions for joints with 6 to 7-DoF have been described in literature. Incremental Newton-Euler and other such formulation and techniques have been used (Otaduy and Lin, 2006). Kinematic equations are sensitive to various factors like effect of gravity, direction of motion, angles subtended at the joints, etc. Therefore, it is difficult to develop analytical or incremental solutions for robots with high DoF. Apart from the complexity of inverse kinematic solution, there are more fundamental problems in robotic motion, e.g., the classical Movers' problem dealing with movement of a convex object through a set of connected passages is NP Hard (Cruz et al., 2016). Further, it has been reported that iterative solutions may not always generate all possible solutions or reach globally near-optimal solutions (Veslin et al., 2014). This highlights the complexity of control for a design that is anywhere close to a human structure.

A heuristic modeling algorithm expressed in terms of homogenous combinations of the classical system dynamics and the Bayesian degree of belief has been presented in literature (Pozna et al., 2010). An arm powered by FESTO fluidic muscles with only two degrees of freedom has been described in recent literature (Trojanová and Hošovský, 2019). In recent years, Multi-layer ANNs have been used to solve complex problems like image identification and natural language processing which were not solvable by conventional computer algorithms. The Fukushima nuclear disaster in 2011 and the subsequent DARPA announcement of humanoid robotic challenge in 2012, focus of robotic research shifted to use of artificial neural networks. Since then, there have been phenomenal advances in this technology. However, most of the work remains highly classified.

This work is an attempt to solve the robotic path planning problem using a multi-layer ANN. A new type of hierarchical network based on a Hebbian Learning Model (HLM), Auto Resonance Networks and Path-nets is presented in this paper. One notable work in this direction was reported by Bing et al. using spiking neural networks (Bing et al., 2018). Analytical search algorithms have also been presented for robotic path planning (Purcaru et al., 2013). A good review of neuro-robotic research has been recently published by Li et al. (Li, et al., 2019). Nature-Inspired Optimization Algorithms for fuzzy controlled servo Systems that may be used for robotic control has been reported recently (Precup and David, 2019).

The problem of locomotion can be split into two sub problems: firstly, to estimate the joint angles such that the end effector reaches the desired location; and secondly, to incrementally move the end effecter from existing position to a new position circum-navigating the obstacles while trying to optimize the cost of path traversed. Both have high complexity and therefore a soft computing approach seems to be the only way to reach a reasonable solution. A neural network specifically designed to handle joint control may be required to achieve a generalized solution to this problem. Reinforcement learning capabilities of deep neural networks hold the key to develop complex robotic control systems without the need for closed form or iterative solutions (Kober et al., 2013). These requirements have triggered development of new neural architectures to address specific problems in building humanoid robots (Martin and Gregg, 2016) (Aparanji et al., 2016) (Wei et al., 2016a) (Wei and Sun, 2016b)(Wei et al., 2017).Given the complexity of biological

systems, it is natural to assume that existing neural models may not always work in newer application domains. Therefore, use of ANN for specific functionality may require development of specific neural structures. For example, Convolution Neural Networks (CNN) are used in image recognition (Yangqing et al., 2014) while Long Short-Term Memory (LSTM) networks are used for time series prediction (Schmidhuber and Hochreiter, 1997). More recently, spiking neural networks are being explored for robotics and several other areas in Artificial General Intelligence (AGI). In the following sections, we describe a novel hierarchical neural network structure based on synergistic control of musculo-skeletal systems, which we call Auto Resonance Network (ARN). ARN can classify real valued multi-dimensional input and have an adjustable acceptance threshold (ρ) for each node in the network. Each layer of ARN has a specific goal and searches are always local to a layer, reducing the overall computational load. Hierarchical ARN represents a feed forward network of ARN layers and other cellular automata. Every node in ARN resonates with in a small controllable volume in input space, called the coverage of the node. Resonance allows approximation to the locus of resonance, giving the network an ability to respond to input different from training set. Long chains of ARN nodes can therefore generate complex classification and recognition patterns. This tunable approximation is critical to working of ARN architecture. To a certain extent, ARN is similar to Radial Basis Function (RBF) network but with entirely different control algorithm. The size of the network grows with input and therefore overcomes the binding problem and plasticity stability dilemma. These networks can be used as generic data classifiers by adding node labeling method or neuronal layers. We illustrate their application to robotic motion. However, they can be used as generic data classifiers and find applications in various areas of artificial intelligence. A modified version of the algorithm has been reported for image classification (Mayannavar and Wali, 2019).

In this paper, ARN has been applied to simulate the joint structures generally found in biological systems. ARN can find multiple solutions when they exist using a feature called as folds which is also described in this paper. Each fold represents a set of joint configurations. Several folds can offer same solution but with different joint configurations. Some among such solutions can be used to overcome obstacles in the robotic path. There is a cost associated with the process of switching between folds as it requires the joint angles to change by large angles, effectively moving the joints from one configuration to the other equivalent configuration. The problem of finding folds among possibilities is complex and needs further work.

This paper presents combined hierarchical ARN with other cellular automata that can be used to solve path generation problems of robotic joints motion. The goal is to guide the end effector of the joints system from its current location to a target location through a series of steps on a near optimal path avoiding obstacles in between. The hierarchical network has been designed with an ARN as input classifier. Intermediate layers provide approximate paths to trained and untrained areas in the work space. Higher levels of the network are used to compare available paths and to select an optimal of path.

The paper is divided into five sections. Section 2 describes the basic ideas and mathematics of ARN. Section 3 describes an application of ARN for robotic motion control. Results and Conclusions are in section 4 and 5 respectively.

2. AUTO RESONANCE NETWORK

Basic ARN will have a single layer of multiple neurons (also called nodes) that is able to identify patterns in input set. Each node resonates to a set of closely related patterns in input set. No two nodes will resonate identically over their entire input set but may have partially overlapping regions. All layers are isolated at the interface from other layers. Each layer works independently. Output of one layer interfaces with the input of next layer in the hierarchy. Therefore, understanding one layer is sufficient. Training a layer of the network would be as follows: Input will be applied as a set, one set at a time. The layer would consist of a single layer of nodes all of which are connected to all inputs. Each node is tuned to recognize a set of closely related patterns of input vector. Internal memory of the node may have an exact or approximate or transformed version of input it matches. When a new input is applied one of the three things happens: (i) one of the nodes is at resonance or (ii) some nodes are near resonance or (iii) none of the nodes are in resonance. In the first case resonating node is the winner. In second case, the node with the highest output is selected as winner if the output is above a selection threshold. Otherwise, there is no recognition. When there is no recognition, a new node may be created such that it is tuned to match the present input. Success of this network depends on finding a suitable function that offers good tunability and a variable cover for every node.

When a new node is appended to ARN, it is pre-tuned to resonate with current input. We will illustrate the concept with a single input network. Resonance of a node can be described using a simple equation

$$y_p = x(1-x) \tag{1}$$

where, x is the input represented by a real number normalized to the range of {0…1}. Equation (1) will yield a maximum value of $1/4$ when the input $x = 1/2$, i.e., the node will resonate if the input is $1/2$. Therefore, we can use

$$y = 4x(1-x) \tag{2}$$

to normalize the result to 1. In order to set the resonance at any value of $x_r \in (0...1)$, we can scale the input such that

$$wx_r = 1/2 \quad \text{or} \quad w = 1/(2x_r) \tag{3}$$

and calculate the output of the node as

$$y = 4wx(1-wx) \tag{4}$$

It is also possible to use other transformations on input, as discussed later in section 2.1. The *resonant weight* w is computed when the node is inserted in the network and remains largely unaltered as an in-memory reflection of the input x_r present at the time of creating the node. This feature will allow the network to remember episodic events. It is important to remember that the resonating nature of the node will allow other inputs in the close vicinity of resonance also to generate an output that is above a selected threshold. The chart in Figure 1 shows the behavior of nodes tuned at various points of resonance identified by x_r. In Figure 2 (a), scaling of single input by the resonant weight and

computation of resonator output is shown. A node with N input nodes is shown in Figure 2 (b). The input to a node consists of a vector

$$X = \{x_1, x_2, ...x_N\}, x_i \in (0...1), \quad i = 1...N \tag{5}$$

For each of the inputs, output of each resonator module given by (4) are summed and normalized as at the output of the node. A layer of ARN will have several such nodes as shown in Figure 2 (c). Assuming that there are K nodes, each one is tuned to a different input vector $X_k \mid t = t_k$ where t_k is the time at which k-th node was created. We can extend (4) to describe output of a k-th node as

$$y_k = \frac{4}{N}\sum_i (1 - w_{ki}x_i)(w_{ki}x_i), \quad i = 1...N, k = 1...K \tag{6}$$

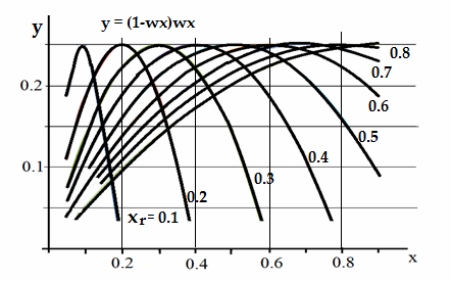


Figure 1. Resonance curves for various x_r

where, w_{ki} is the scaling factor for i-th input of k-th node. The resonant weight w_{ki} represents the in-the-node reflection of input x_{ki} stored as memory in the node. Note that x_i is the i-th element of the input vector while x_{ki} represents the input x_i for k -th node. This k -th node will produce maximum value of 1 when $x_i = 1/(2w_{ki}), i = 1..N$. In Figure 2 (a), the resonator function is indicated as $d(1-d)$, where $d = wx$.

To summarize, each resonator corresponds to one input of one node. Each node has N inputs and same number of resonators. The output of a node is maximal when all the resonators produce maximum output. Therefore, the resonant weight of a node is expressed in vector form as $W = 1/(2X_r)$.

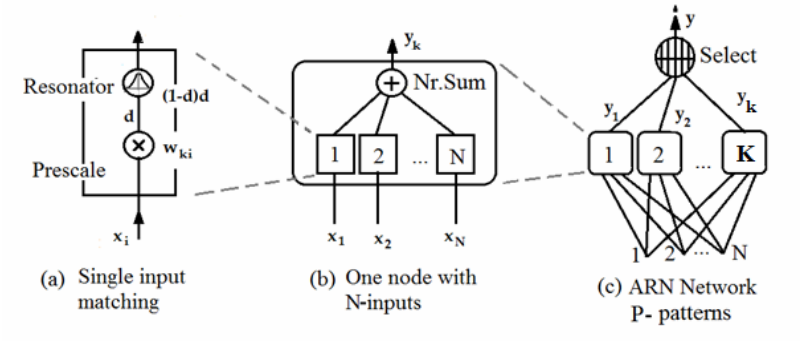


Figure 2. Auto Resonance Network structure

When the node is added to the network, initial resonant weights are computed. The weights will undergo further tuning as learning occurs later during training. However, such tuning

adjustments are small and the node remains in resonance in the close vicinity of initial locus. Therefore, the node is automatically tuned to the input it is presented with. The learning procedure ensures that a specific combination of inputs is maximally matched by only one node in the layer. Therefore the network is called as Auto Resonance Network (ARN). It is interesting to note that resonance also implies that the node responds to inputs that approximately match the locus of node resonance. Therefore, such approximation actually contributes to the learning by network.

2.1 Envelope functions

It can be seen from Figure 1 that output of a resonating node decreases as a continuous function of input on either side of input values. If the output of a node is above a threshold value the node is considered to be a winner. The range of input values at which the node can be a winner may be adjusted by reducing or increasing the threshold. If the threshold is reduced, the range increases and if the threshold is raised, the input range reduces. The set of all inputs when the output of a node is above a threshold is called *coverage* of the node.

The coverage of k-th node can be expressed as

$$C_k = \{X \mid (y_k > \rho) and (y_k > y_i, \forall i \neq k)\} \tag{7}$$

where ρ is the threshold value and k .is a set of all input values.

We may further note from Figure 1 that coverage of area for each node is not same for a given threshold. For example the peak for $X_r = 0.1$ is significantly sharper than the one at $X_r = 0.2$. We could set separate thresholds to individual nodes such that all nodes have similar coverage. However, a better way to correct this situation is to use a non-linear scaling of input. We call these functions as *envelop functions*.

Envelop functions can provide several advantages. For example, they can transform unbound input $x \in \Re$ into bound region like {0…1}. If the envelop function modifies the input $x_s = g(x)$ then, the resonant weights also should be scaled with identical function.

$$w_{ki} = 1/(2g(x_{ki})) \tag{8}$$

$$y_k = \frac{4}{N}\sum_i w_{ki} g(x_i)(1 - w_{ki} g(x_i)) \tag{9}$$

Envelop functions stretch or compress a specific part of the input range in order to exemplify an area of interest. Effect of some of these functions is shown in Figure 3.

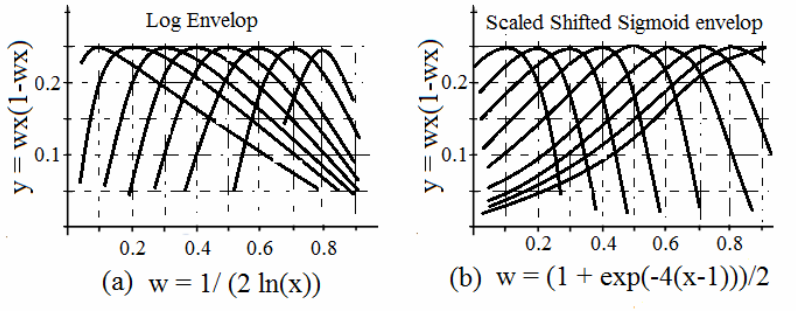


Figure 3. Effect of envelop functions on classification

A simple scaled log function shown in Figure 3 (a) implements

$$g(x) = \sigma \ln(x) \tag{10}$$

where σ is scaling factor. By adjusting σ we can effectively compensate for the non-linear coverage of (4). Figure 3 (a) shows the effect of (10) on coverage with $\sigma = 1$. A modified sigmoid function shown in Figure 3 (b) exhibits a controllable linear coverage. It uses an envelope function

$$g(x) = 1/(1 + e^{-\sigma(x-1)}) \tag{11}$$

A value of σ = 4 is used for illustration. It is clear that envelop functions can reduce the non-uniform coverage across the input range.

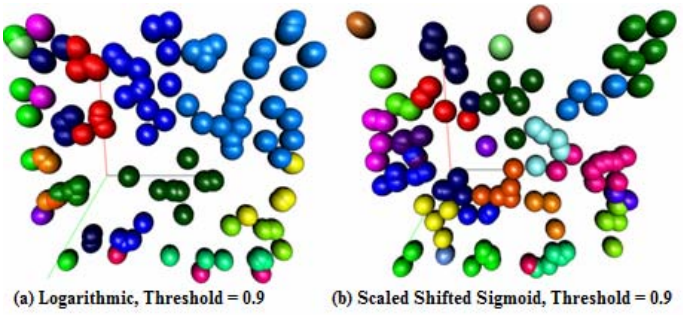


Figure 4. Effect of envelop functions on classification

Figure 4 shows the clustering of nodes using two envelop functions with input vector size of 2. Each node in this ARN network is identified by a color. Group of bubbles of same color represents a single output cluster. The image shows the effect of using envelop function on the node coverage. Fig 4(a) shows larger cluster for higher values of input while the sizes are more uniform in Fig. 4(b). Cluster sizes depend on the envelop function and the threshold.

2.2 Extending the input range

Accepting input in the normalized range of {0…1} need not be a limitation, but it would be convenient if there are other functions that provide a larger input range, yet maintain the resonance property. Interestingly, there are many other monotonic functions to implement such resonance and build an ARN. A generic approach would be to define an additive inverse of the function over a range and multiply the two to get a resonance function. One such simple function is the difference function given by $(M_{ki} - x_i)$ such that

$$y_k = 1 + \sum_i (M_{ki} - x_i)(x_i - M_{ki}) \tag{12}$$

where M is the value of the tuned input (M for mean resonance value). Figure 5 shows the distribution of (12) for various M values. This function has a maximum value of 1 when $M_{ki} = x_i$. Note that the node stores the input without applying any transformation. From Figure5 we observe that resonance can be set at arbitrary points on $x \in \Re$ and not limited to {0…1} range. The complexity of (12) is of $O(N)$ and hence fast to compute. Coverage of the nodes is uniform and given by range points described by the following equation.

$$x_p = M \pm \sqrt{1 - \rho} \tag{13}$$

where ρ is the coverage threshold. The above equation gives two range points between which the node will have maximal output. Effect of threshold on classification of data for this network is shown in Figure 6. A new node is added when maximum output value of all nodes in the recognition layer is below the threshold. The image clearly shows that increasing the threshold increases the number of output nodes.

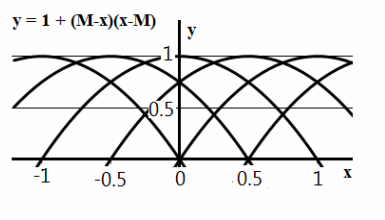


Figure 5. Resonance graphs for nodes using function in (12)

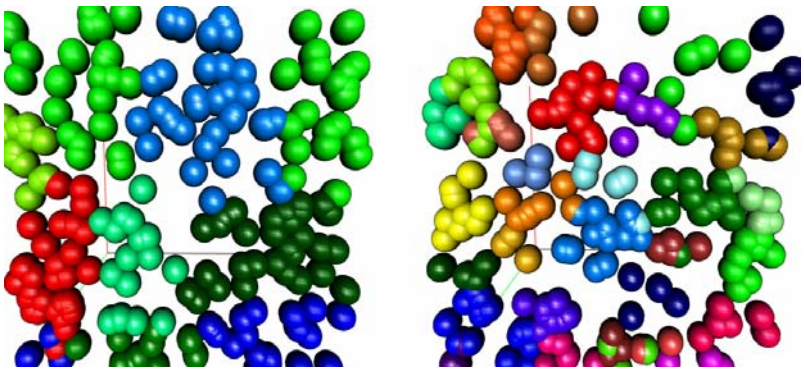


Figure 6. Effect of threshold on number of clusters
(a) Threshold = 0.7, (b) Threshold = 0.9

Another good candidate is the Scaled and Shifted Sigmoid Function (3SF) given below: This function

$$y_s = 1/(1+e^{-\sigma(x-M)}) \tag{14}$$

will map a real number in the range of $-\infty:\infty$ to a monotonically increasing value in the range 0:1. Interestingly, the function has a value of 0.5 at $x = M$. Therefore, we could replace $w_{ki}x_i$ in (4) with this function. It may be noted that $(1-y_s)$ can be easily computed as

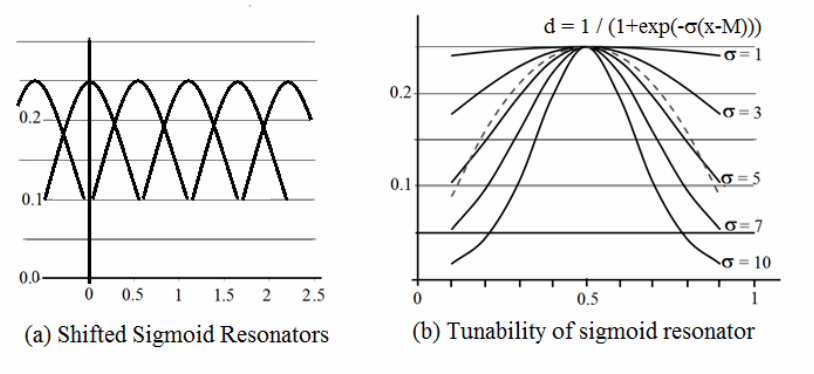
$$y_{s-} = 1/(1+e^{\sigma(x-M)}) \tag{15}$$

Therefore, for N=1, we can rewrite (2) as

$$y = 4\left(\frac{1}{(1+e^{-\sigma(x-M)})}\right)\left(\frac{1}{(1+e^{\sigma(x-M)})}\right) \tag{16}$$

The sigmoid nature of this curve fits well with a physical neuron activation which shows saturation as the input increases, rather than growing monotonically. Equation (16) allows a node to be set to resonate at any $x \in \Re$. Note that M can be used to select the point of resonance while σ can be used to control the tuning and hence coverage of the node as shown in Figures 7 (a) and (b), respectively. Equation (16) provides a generalized function

for implementing ARN nodes, albeit with increased complexity. The tradeoff between complexity versus flexibility can tilt towards (4) or (16) depending on end use.



(a) Shifted Sigmoid Resonators (b) Tunability of sigmoid resonator

Figure 7. Tunability of ARN using (16)

2.3 Tuning the nodes and coverage

Coverage of a node acts like an approximator by providing near maximal output when the input is close to the resonating value. Coverage of the node can vary dynamically based on input. Node may slowly shift to a different tuning point depending on the statistical properties of incoming data but normally stays close to the original tuned location.

Controlling the quality of resonance as shown in Figure 7 (b) can be used to sharpen the performance of ARN nodes. For example, if a node receives exactly the same input repeatedly, we can increase its sharpness by increasing σ. On the other hand, if the input value varies around the resonance value but within a threshold, we can reduce the value of σ to increase its coverage. A simple relation that can be used to tune the resonance is given by,

$$\sigma_{(n+1)} = \sigma_n + \eta\kappa f\ /(1-\nu) \tag{17}$$

where η is learning rate, κ is a proportionality constant related to statistical frequency f, i.e., number of times the node matched the input and ν is related to signal variance. Therefore, this equation provides basis for reinforcement learning on ARN nodes.

Stable nodes can undergo further tuning to increase or decrease the area covered by the nodes. This can be achieved by varying the selection threshold or the σ value associated with the node. This requires that the nodes compute statistical moments as they are accessed. A mapping of σ value to the coverage is given in Figure 9 (b). For a node described by (16), and knowing that $y_{\max} = 1$ and assuming $x_r = 0$ we can write the value of x for threshold of $y = \rho$ as,

$$\rho = \frac{4}{N}\left(\frac{1}{(1+e^{-\sigma x})\{1+e^{\sigma x}\}}\right) \tag{18}$$

which gives an expression for coverage of an ARN node as a function of threshold and tuning factor. Equation (18) can be rewritten as

$$x_\rho = \frac{1}{\sigma}\cosh^{-1}\left(\frac{2}{N\rho} - 1\right) \tag{19}$$

which gives the coverage of an ARN node for various values of threshold and scale factor around the peak value. This relation is plotted in Figure 8.

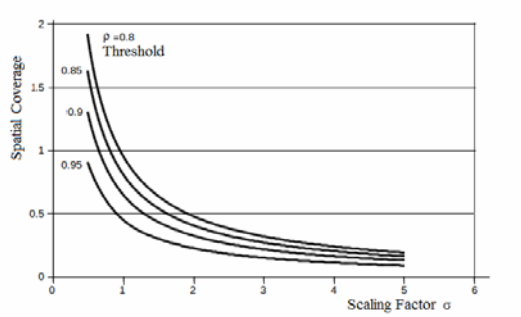


Figure 8. Spatial Coverage of a node as a function of σ

It is clear from Figure 8 that as threshold increases, the spatial coverage reduces and hence the node activation becomes more sensitive to input. This controllability of the coverage of ARN node can be used to alter the selectivity of the node. For example, in Figure 9 (a), two nodes are located near each other but have separate coverage. However, one of the nodes may expand its coverage such that a part of second node is covered. Second node will generate maximal output if the input matches exactly but may not get selected though its output is above threshold (Figure 9 (b)). In other words, the node has gone below cognitive level. It exists but resides in a 'hidden,' or 'subconscious' layer. Note that such nodes can move out of dormancy when the precise input is applied.

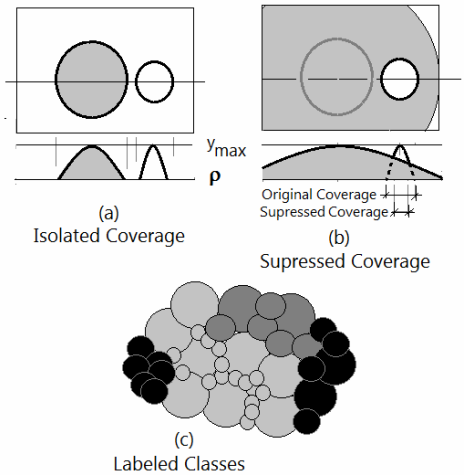


Figure 9. Effects of Spatial coverage of ARN nodes.
(a) Isolated Coverage, (b) Suppressed Coverage and (c) Labeled classes

Figure 9 (c) shows clustering of nodes in the neighborhood as well as in separable neighborhoods. This is facilitated by providing labeling capability to ARN nodes. This labeling capability can be used to associate multiple clusters of data into a single disjoint set. This association can be marked with a single label. For example, when two joint configurations move the end effector the same location, we can label them identically. We will discuss this further as 'folds' in Section 2.5.

2.4 Types of ARN nodes

Typically, ARN nodes are created when an input does not trigger resonance in any of the existing nodes. Such nodes may be reliably labeled if the expected class of output is known, as in case of supervised learning. These nodes have a well defined point of

resonance, an adjustable coverage and a well defined output label. We will call these as *Type-1* nodes.

Additional nodes can be created by cloning and perturbing existing Type-1 nodes, without a need for external input. Such nodes may also be created by interpolating properties derived from Type-1 nodes. Because of non-linearity of real systems, such perturbations and interpolations may only be done in the vicinity of existing nodes. The output and associated input data can be estimated as a perturbation of Type-1 nodes or interpolated using a suitable approximation. These values need to be adjusted during further training or run time for the point of resonance, output values and labels. We will call them as *Type-2* nodes. In general, perturbation of Type-1 nodes can be performed by considering one input or output variable at a time. The type of data Type-1 node associates has a bearing on development of perturbation methods. Type-2 nodes are essentially mutated copies of Type-1 nodes. Their input and expected output values are approximated using some approximation functions. When an isolated Type-1 node has to append Type-2 nodes to the ARN layer, a new node is created with randomly perturbed input in the vicinity of the Type-1 node. The expected output is computed by using the displacement of the input from current node and an approximation function. One such set of approximation functions is shown in Figure 10. Actual diffusion function to be used depends on the end application and may vary from region to region in input/output space. Figure 10 shows four flexed lines indicating how the output varies in the vicinity of the Type-1 node under consideration. These can be seen much like the shape functions used in finite element modeling of structures. These curves give an indication of how much the output value should change in close vicinity of the cell. Here the function used is given by,

$$\gamma = \pm k(1 - e^{\pm \lambda r}) \tag{20}$$

where r is a measure of distance between input of two nodes, λ is a scaling factor such that $|\lambda r|$ < 0.5 and γ is the output scale factor. Expected output of the mutated node is computed as $y_n = y(1+\gamma)$. This is only an initial value which will be updated during training of the network. Therefore it is not critical to know the exact value. Note that (20) generates four curves flexing in different directions. The parameter k controls the magnitude of γ and λ controls the flexing.

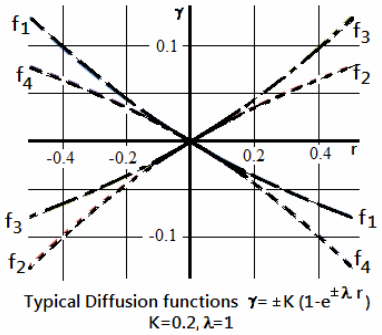


Figure 10. Typical output diffusion function

It is possible to use other shape functions as suitable for the application. When two or more nodes are available, the new nodes may be assigned linearly interpolated values.

Type-2 nodes are necessary to address random or unknown situations beyond the noise margins of ARN nodes, which the network may encounter later. These nodes are appended to the same layer as Type-1 nodes. Tuning of Type-2 nodes requires special attention but is a rewarding procedure. Perturbation methods work well when the output of the network is a

continuous function. However, if the output is a discrete set of values, input perturbation may work like a clustering problem but output perturbation methods may not be meaningful.

2.5 Folds of solutions

Often it is possible that several distinct combinations of inputs yield the same output. For example, several joint angles may move the end effector of a robotic arm to the same position. Each such set of solutions is called a fold. Folds occur as contiguous set of nodes in input space and are separable as clusters or by specific properties of input. Each fold can be associated with a cost, giving a priority over usage. Moving between folds may also involve a certain cost as the system may need time or effort to shift between folds. Therefore, choice of a specific fold depends on input and previous outputs or states of the system.

Consider a planar mechanism with two segments, a fixed joint and a flexible joint, as shown in Figure 11 (a). The solid lines indicate one set of joint angles and positions of the linkages The small-dashed lines indicate an alternate position of joints and linkages to reach the same end location. It is clear from the figure that two sets of angles will set the end point of the mechanism to the same location. This is an important feature of many real life situations: redundant solutions exist to many problems and they need to be explored based on the situation. For example, if part of the output space where the joint can flex is obstructed, we have to use the possible alternative to move the end point of the mechanism. There will be a set of such continuous locations that can be reached using such similar sets of angles (Aparanji et al., 2017a), (Aparanji et al., 2017b). Therefore, the concept of folds gives an ability to lower the cost of finding a solution by reducing the search space. Additionally, the search can yield low power changes to joint angles to reach the intended final location.

Note that identifying folds is a complex process and needs to be studied in detail for a given application. In Figure 11 (b) the bubbles indicate coverage of a ARN node. Each plane of such nodes has angles that are continuously varying.

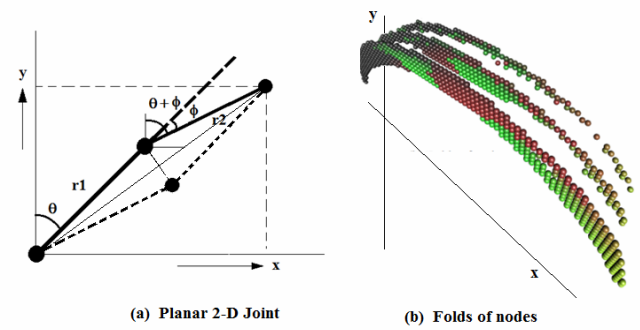


Figure 11. (a) Two segments Planar Joint and end point displacement in x, y space.(b) Folds of solutions

Third layer is caused by the indecisiveness of the fold identification process to assign the joint angles to specific fold. It is easy to observe following points:

i. Neighborhoods in input space produce outputs that are neighbors in output space.
ii. It is possible that several combinations of inputs exist that yield the same output.

Folds occur in continuous set of nodes and are hence separable by specific properties of input. Each fold can be associated with a cost. Much as moving from one location to the other involves cost, moving between folds also involves a certain higher cost. Therefore, choice of a specific fold depends on input and how it was handled during earlier occurrences.

Folds of solutions are a natural phenomenon in many multivariate systems. Ability to identify such folds helps in reaching an optimal solution faster and at lower cost of transition.

2.6 Learning in ARN nodes

Structure of an ARN node is different than that of perceptron or other neural networks of that type. Learning in ARN occurs when a set of input and output vectors are applied to a layer of nodes. These may be part of a training set as in case of supervised learning, or picked up from the environment (unsupervised or reinforcement learning). The resonator can tune the node to a specific input set and produce a constrained output, limited to a predefined level (usually 1). Maximum output is reached at the point of resonance and hence there is no scope for a race condition that causes an unbounded output. The node and hence the network are inherently stable in terms of output. Oscillatory behavior does not occur in ARN because it is a feed forward network, and the rules of selection / creation will assign a specific node as winner corresponding to an input set. Ambiguity resolution is precisely defined by selection / creation rules.

The weights of the nodes are initialized at the time of creation, based on the applied input and hence represent a memorized impression of the input. The weights may vary but only to a small extent, in the close vicinity of the original location. It is not likely that the point of resonance is shifted far away from the original resonant point. Incremental shifting of the resonant point is useful in interpolation of the response as well as extrapolation of input range. If we consider a single node (with multiple inputs) at the beginning, appending additional nodes is equivalent to expanding the operating range.

It may be noted that the ARN node is in resonance with respect to the input. The output is bound to an upper limit (usually 1). Weights are updated entirely at the node, based on local information, without consideration of the overall operating range. Statistical mean and variance of input are good enough. Therefore, there is no need to search for an optimal set of weights along negative gradient of the learning function. Learning is achieved by tuning the node to adjust the coverage of the node rather than shifting the node weights in response to error in output. In case of cluster identification, label of a node may be shifted to neighboring cluster or a new node be added with new label. In that case, node suppression indicated in Figure 9 (b) takes effect. We do not need to reach an optimal value of coverage as it is a time varying property of the node. As more input is applied, the coverage can be adjusted based on the average spread of data, typically indicated by the standard deviation or similar statistical property. Selection of the threshold and scaling factor to match the observed coverage is given in next sub-section.

Coverage of the node is an indicator of the noise tolerance of the ARN node and its ability to distinguish between noise and novelty of presented data. Output of ARN node is essentially an unconstrained $\Re^n \rightarrow \{0...1\}^k$ map function, where n is the number of inputs to the layer and k is the number of nodes (or cluster labels) in a layer of the network. Labeling of the nodes allows identification of convex or concave sets of data.

2.7 Radial Basis Function (RBF) networks and ARN

Equation (14) and its characteristics shown in Figure 7 are similar to a Radial Basis Function (RBF) (Wei He et al., 2016a). Gaussian form of the RBF which is generally used form of RBF is expressed as

$$y = \sum_{i=1}^{k} w_i e^{-(x_i - M_i)^2 / r_i^2} \tag{21}$$

The equation has a minima when input $x_i = M_i, \forall i$. The parameter r indicates a radius of input values within which the output is significant, similar to threshold in ARN. Size of the input vector x is k. Normalized form of (19) is also used frequently in literature. If we define $s(x) = \exp(-(x - M)^2 / r^2)$, then the normal form of (21) is given by

$$y = \frac{\sum_{i=1}^{k} w_i s_i(x)}{\sum_{i=1}^{k} s_i(x)} \tag{22}$$

The function $s(x)$ describes a measure of Euclidian distance of input x from center value M. It is possible to use other measures like Manhattan distance. Both ARN and RBH use a similar strategy to train the network, i.e., present the input and expected output and let the network remember the applied data set. The centers of RBH nodes viz., M, are obtained by algorithms like k-means clustering. In case of ARN, the center of resonance is established at the time of creation, with a single input. The outputs of RBH nodes as in (21) are presented to a perceptron layer to compute the network output. ARN nodes simply remember the data associated with each node, eliminating the stability issue. Radial basis functions are useful as generic approximates and find applications in statistics and machine learning.

2.8 Spiking neural networks and ARN

Spiking neural networks implement information transfer using a pulse train, similar to a biological neuron. Inputs at the dendrites decay over a period of time. The neuron integrates all the decaying inputs and fires when the summation exceeds a threshold. There are some distinct advantages of spiking neural networks:

(i) A temporal relation is established between the pulses that can code for information enhancing the search space, and

(ii) Several combinations of inputs can generate similar pulse trains, establishing a relation between several combinations, similar to folds in ARN. Pathnets described in association with ARN also have similar capabilities, as described later in this paper.

(iii) Application of certain pulse sequences can quench current pulse trains, suppressing expression of recognition. This feature will be useful in switching between possible solutions quickly. Pathnets in ARN are continuous functions rather than pulses, allowing a smoother transition.

3. NEURAL NETWORK FOR JOINTS MOTION CONTROL

A deep learning network has multiple layers of cognitive and computational nodes. Neuronal layers are used for cognitive function as they are better suited for that task. On the other hand conventional digital computers are better used for mathematical and logical parts. Combinations of the two types of structures have been shown produce usable and intelligent systems. We demonstrate that ARN with other cellular structures can be effectively used for robotic joints motion control.

Joints motion control is generally solved using iterative inverse kinematics methods wherein joint angles and the force are calculated. Closed form analytical solutions are possible for joints with small number of DoF. Iterative solutions using Newton-Euler representation are commonly used for 6-7 DoF (Houston and Kelly, 1982). Neural and fuzzy systems have been used for joint systems with higher DoF. It appears that generalized solutions for joints systems with large DoF may be possible with deep neural networks.

3.1 The joint system

A joint system consists of movable mass-spring-damper components and a drive system. Movement of the joint system is effected by several hinged components each with a fixed number of DoF. We have used two types of joints. A planar joint with two segments is shown in Figure 11 (a). It has 1-DoF per segment. We have used it as a first example to illustrate the concepts.

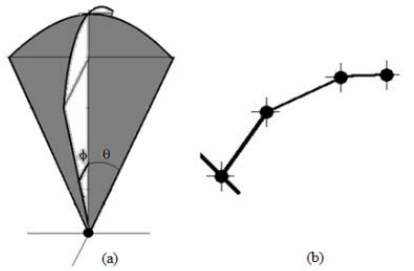


Figure 12. (a) Single segment of Joint configuration in x, y, z space
(b) a multi- segment joint using details shown in (a)

The spherical joint shown in Figure 12 (a) has 2-DoF per segment. A number of these joints and segments can be added to a complex joint system as in Figure 12 (b). These two types of joints are used for all simulation work during this work.

Figure 13 shows different views (top, front and side) of distribution of nodes using two segments of spherical joint shown in Figure 12. The number of nodes created depends on the joint configuration and the spatial distribution of Type-2 nodes. Interestingly, number of nodes reduces as the spacing between Type-2 nodes decreases. This is because nodes suppress attempts to insert new nodes. On the other hand, when the spacing is large, the coverage is not enough and new nodes do get inserted in the network.

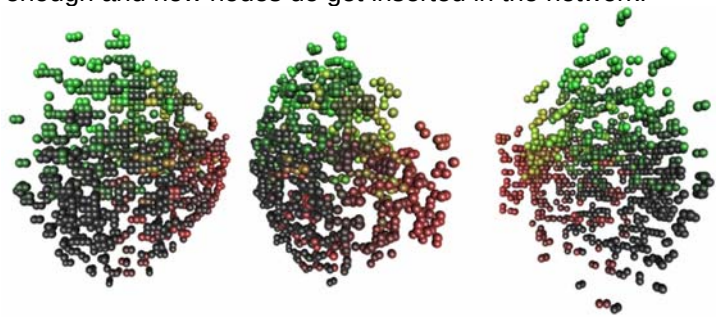


Figure 13. Formation of nodes in a spherical two segment joint

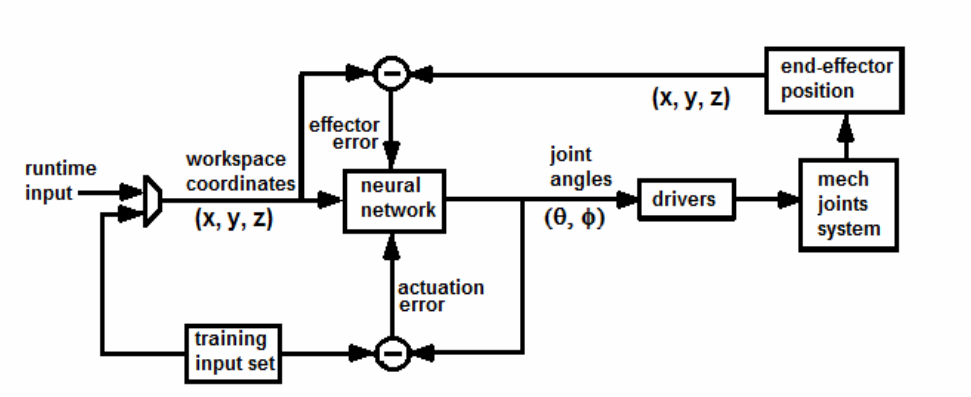


Figure 14. Overview of the Joints control system using ARN

3.2 Neural network for motion control

A multi-layer neural network for motion control with ARN as input classifier has been implemented. The goal is to guide the end effecter of the joint system from its current location to a target location through a series of nodes located on a near optimal path, avoiding obstacles in between. Higher levels provide paths for motion and control structure for optimization of path. Some details of the implementation are discussed here.

The implemented system consists of a layer of ARN nodes that receive location of the end effector as input. The nodes are labeled with joint angles to be used to reach the required end location. It is possible to use a hierarchy of ARN layers that refine the accuracy successively as the effector moves towards the expected target. During run time, the end effector can move to a untrained location using the approximation provided by ARN.

Figure 14 shows an overview of multi-layered joints control system using a neural network. The multi-segmented joints system has a well defined 3-D work area defined by the mechanical design. The system uses unsupervised learning. During training, the joints are randomly excited and the end-point is located, much like a child trying to move its hand. During training, the network learns to relate *end point location* with a set of node which are labeled by *joint angles.* A node therefore identifiable by joint locations and uses effector location as input i.e., $\{(x,y,z),(\theta_1,\theta_2,...)\}$. The first part of the set used as a input for resonance while the second part is used as a node label. This data will be used to reach the target position. During initial simulation, the end point position is calculated from the joint angles, using forward kinematic equations. Every such end positions is used as label for resonating node. In a real system, this calculation is not necessary because when the joints will actually move corresponding to the applied random excitation. During run time, the location of the end point is known and hence input to the ARN. The label from the resonating node will indicate the joint angles to be used to reach the required location.

A database oriented implementation may be possible using an approach presented by Zall et al. (2019). Multi-relational data can be on relational databases where they consist of multiple relations that are linked together by entity-relationship links. The class label can be predicted by correlating the information of related data. Labels can then be propagated to create the paths.

In Figure 14, two distinct types of errors are shown. During training, actuation error is used to insert and label ARN nodes. These are all Type-1 nodes. On the other hand, during run time, effector may move to a somewhat different location than expected location, which is then corrected by shifting the location of active node (tuning Type-2 nodes). Such repositioning will make the network to produce a new set of actuation signals. The advantage of this feature is that the network need not retune during field operations,

ensuring that stored nodes and edges are not disturbed from their tuned locations. The ratio of Type-1 to Type-2 nodes can vary but of the order of 1:100.

3.3 Hierarchical ARN

ARN can perform real-world input classification. It requires additional support structure to implement application specific functionality. A hierarchical network of nodes can provide such support. One such possible structure for motion control is given in Figure 15.

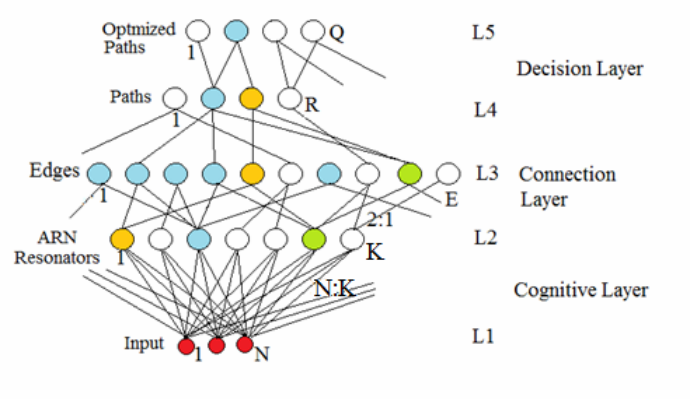


Figure 15. Hierarchical ARN for joints motion control

ARN nodes are in L2. A short connection layer (L3) is added to the network to superimpose a topological order among the nodes. Nodes in L3 can indicate a simple 1-hop neighbor or hyper-edge indicating a closely connected set of nodes. These connections can be interpreted as paths between node locations. The nodes in the Decision layer (L4) are capable of storing a series of node sets. In this case, they store a series of edges to be traversed between the start and end location indicated by nodes in L3. Each node in the path layer (L4) has a cost, which is stored as associative data with path nodes. When there are multiple paths with same source and destination node, nodes are added to a higher layer (L5) that evaluates and selects optimal paths. Both algorithmic and neural implementations of the hierarchical network are possible (Masumeh et al., 2017).

3.4 Perturbation methods

Adding Type-2 nodes requires some attention. Due to non-linearity of displacement, unreachable areas, and directionality of the joints system, distribution of nodes and the associated data cannot be known exactly with a limited set of training data (Spall, 1992). Therefore, we need some guaranteed methods to add Type-2 nodes to L2. There are two approaches to create these new nodes. The first one is to perturb the resonance input of the cloned node and approximate the joint angles. The second one is to perturb the joint angles and use forward kinematic equations to simulate location of end point. Alternately, we can interpolate the values instead of using forward equations. In real life situation, interpolation would be more natural than actual motion, which is equivalent to using forward kinematic equations during simulation. In both the methods, linear and exponential types of scatter functions can be used for perturbation of the network. Granularity and coverage depend on the scatter function used. The effect of scatter function on generation of nodes is illustrated in following figures.

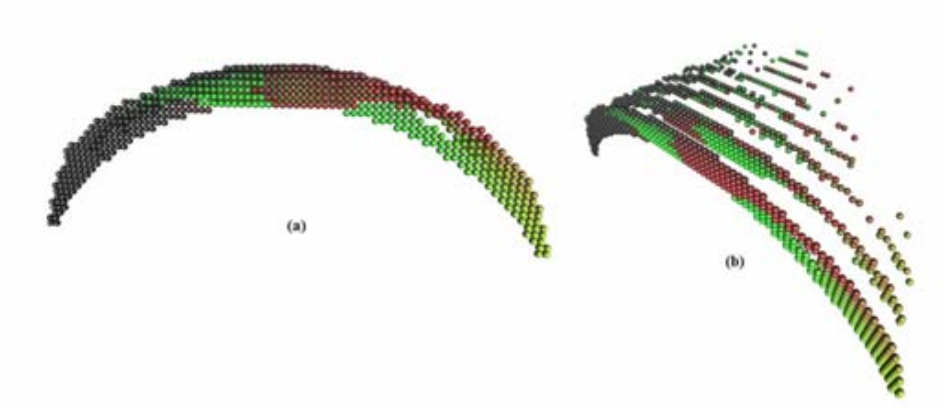


Figure 16. (a) Formation of cells for a single segment planar joint with linear gradient function. (b) Different 'Folds' of solutions

Figure 16 (a) shows the formation of cell for a planar joint with two segments, using a linear scatter function, which generates a fixed grid. Different folds for this joint system are shown in Figure 16 (b). Physical system has fewer folds but the network has identified many more due to uncertainties in associating a node with a specific fold. The results are rotated by an angle in Figure 16 (b) to make the folds visible. An exponential scatter function like $\theta_{new} = \theta + e^{-\lambda d\theta}$ has been used for a two segment planar joint in Figure 17, for comparison.

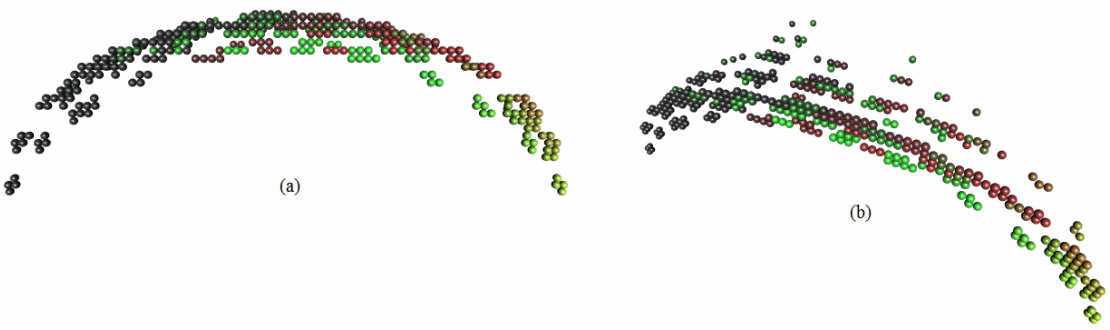


Figure 17. (a) Formation of cells for a planar joint with exponential gradient function. (b) Different 'Folds' of solutions for the joint system

3.5 The connection Layer

Edges are created between different Type-1 and Type-2 cells. An edge represents the connection between two cells present in a fold. Possible instances of creating the edges are

1. Soon after a new cell is created in Layer L2.
2. If the Euclidean distance between any two Type-2 cells of the same group is less than the threshold. Here group represents a collection of Type-1 and its Type-2 cells.
3. If the Euclidean distance between any two Type-2 cells of different groups is less than the threshold.

Figure 18 shows the creation of edges between groups of nodes. A group represents a collection of a Type-1 node and its neighboring Type-2 nodes. Large dots (red color) in the center line of Figure 18 show Type-1 nodes. Small dots (blue color) represent Type-2 nodes. Edges (green line) have been created between the nodes present in a same group as well as different groups.

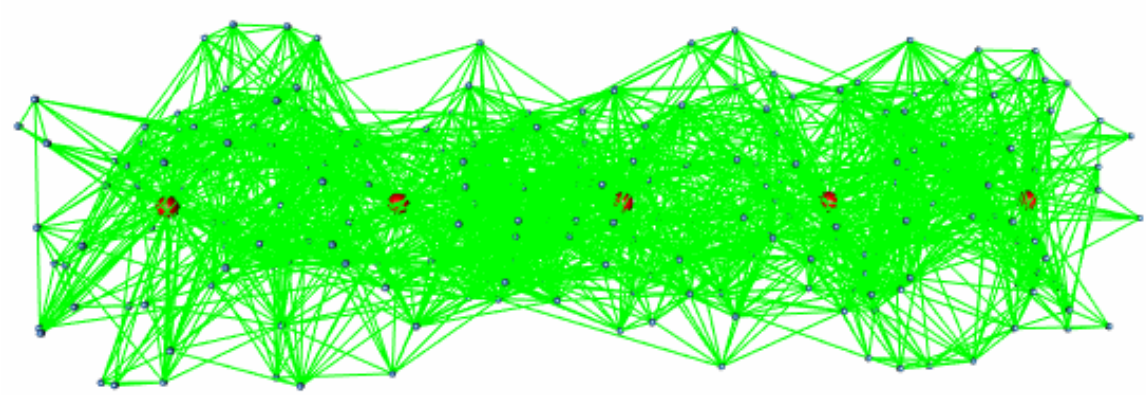


Figure 18. Formation of edges for a three segment spherical joint with 5 Type-1 Nodes and associated Type-2 nodes

3.6 Path Layers

Transition of winning node in a sequence of events represents a path along the nodes. This can occur during training or during run time. Specific paths can be stored during training. However, during run time, when the desired location of the end point is input to the neural network, one of the possible neighboring nodes is triggered based on a cost function. That node now emits the joint control parameters. This process is repeated till the end point is reached. If an edge is not available, the motion tries to search alternate ways. Some times adding a new edge in L3 will hasten the transition making in more energy efficient. As the process of finding the path is dynamically chosen, it is possible to circumvent any obstructions as and when they arise.

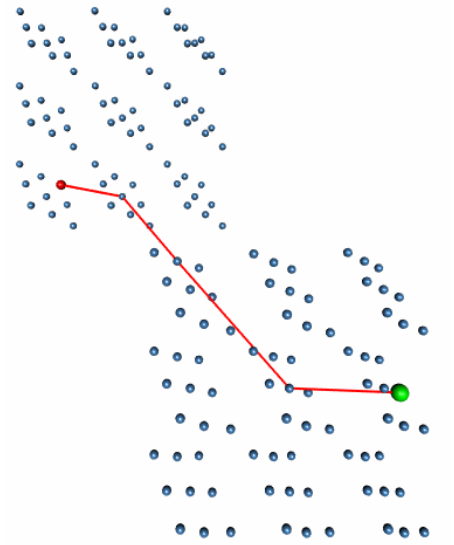


Figure 19. Random path of a two segment spherical joint

Paths constructed using L2 nodes and L3 edges are shown in Figure 19. For clarity, edges other than those on the path are not shown in Figure. 19. Source location is represented by Red dot whereas, destination/end point location in space is indicated by green dot and blue dots indicate the neighbor nodes. Series of red lines indicate the path traversed from source to destination.

Paths can be shortened by selecting chords between closely located nodes. Paths constructed using L2 nodes and L3 edges and selection of chords (blue line represents the chord) are shown in Figure 20. Paths can be optimized by considering a cost of path. The cost is defined by two parameters: Total length of the path and angles subtended between the sections of path. Acute angles are more expensive in terms of power and hence cost. They are also slow to traverse. A simple cost function as a sum of cost of traversed edges is

$$C = k \sum_{i=1...L} \frac{l_i}{w_i} + s \sum_{j=1...n} c(\pi - a_j) \tag{23}$$

where C is the cost of the path, L is the length of the path in terms of sections of path, s is a scaling factor, c is cost of angle a in radians.

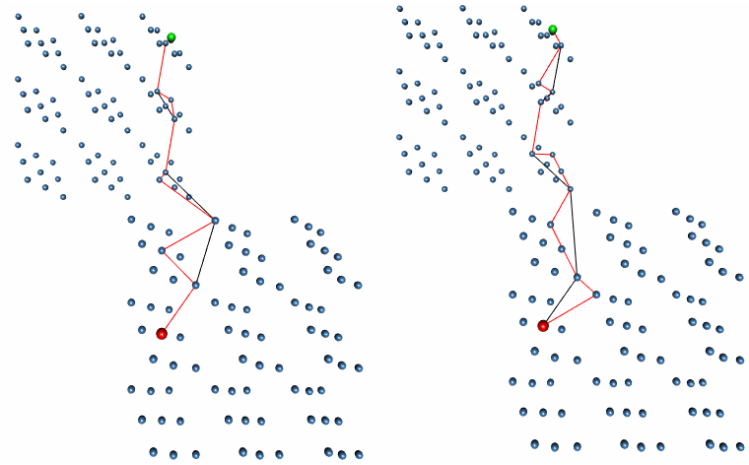


Figure20. Random path of a two segment spherical joint and partial optimization

The nodes in the path-layer store the path cost. This information is used by nodes in the higher layers to select an appropriate path among several possible paths. Figure 21 shows selection of a cost optimized path among several paths between given start and end nodes. Different colored lines indicate the multiple paths. The path with the thick line indicates the best path among several paths.

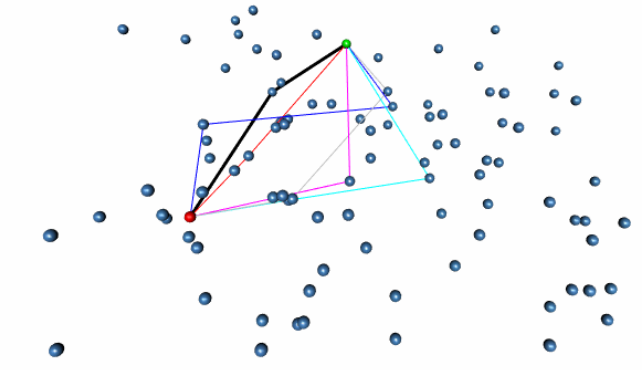


Figure21. Selection of optimized path among several other paths

Paths are searched dynamically. In that sense, hierarchical ARN is under perpetual training (reinforcement learning). Therefore, obstructions can be inserted during run time. Hierarchical ARN can locate paths around the obstacles as shown in Figure 22. Orange dot indicates the source and green dot indicates the destination. Dots in the red and blue color

indicate the obstructed nodes and the free nodes respectively. Cubes represent the obstructed area. We can observe that the best path has the least path length and least cost as compared with the other paths.

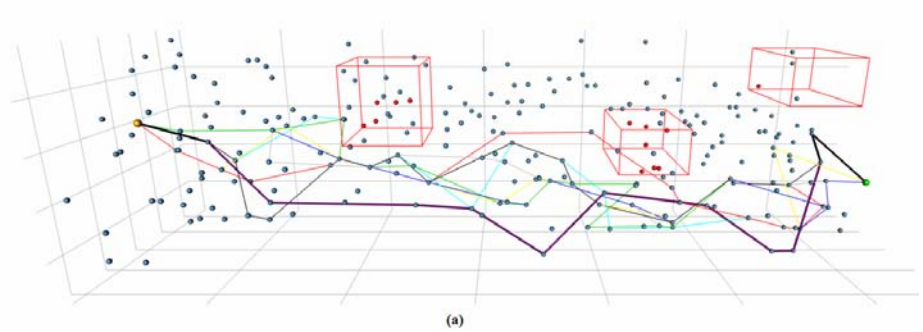


Figure. 22. An optimized path for a three segment spherical joint with obstructions

3.7 Pathnet

Most of the discussion in the previous section looks more like an algorithmic superposition on a basic ARN network. However, a completely neural interpretation of the various layers is possible. For example, creation of edges (connections) between nodes follows a Hebbian learning rule. Nodes that are used in a path increase their weights, providing a low cost option to select among several other possibilities. The formation of chord can be seen as firing of neural pathways based on synaptic weights, introducing a temporal aspect to the path selection problem. This allows a neural network to simultaneously explore multiple paths with a breadth or depth priority based on cost of path(Aparanji et al., 2018).

Pathnet offers a generic approach to solve several problems in artificial intelligence. But the possibilities of its application in natural language processing, time series prediction exist and need to be explored further.

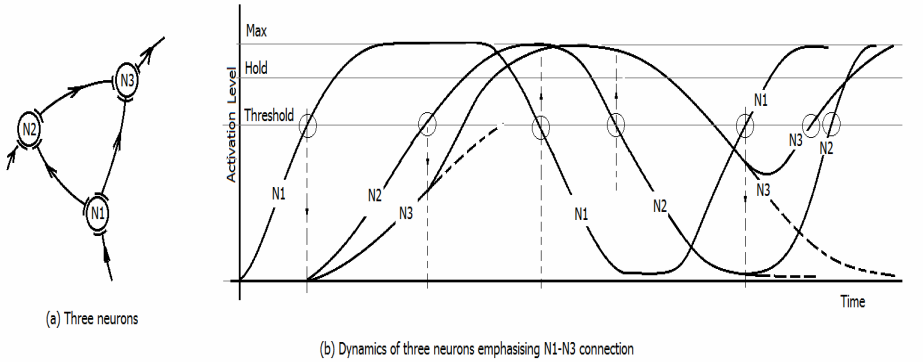


Figure 23. Formation of chords in connection layer

3.8 Combined training algorithm

The combined algorithm to build the network, corresponding to Figure 15 would involve the following steps:

(i) Initially, the network has no nodes. So the first input will create a node as per equation (4) or similar equation, e.g. equation (16). Each node will store its weights corresponding to the input at the time of creation. A default coverage is assigned to the nodes. Node is labelled as computed by forward kinematic equations (for simulation) or as actual end effector position (using real system). As more input is applied, more

nodes get added along with the labels to slowly covering the complete work area. All These are Type-1 nodes.

(ii) When new nodes are inserted in L2, nodes in the vicinity of other Type-1 nodes are added to L3. This process continues as long as new nodes are added to L2. A node in L3 represents a spatial connection between two nodes. These nodes may be used to interpolate distances during insertion of Type-2 nodes.

(iii) Similar to step (ii), nodes are added at a higher level indicating a relation among nodes of a lower hierarchy. For example, a node in L4 can indicate a path traversed using a set of L3 nodes. L3 nodes in turn indicate an spatial neighbourhood among L2 nodes.

(iv) As the node density increases, the nodes will start getting repeated inputs, which help the nodes to tune the coverage. Tuning curves discussed earlier may be used to improve accuracy of recognition.

(v) Simultaneously, the nodes start spawning Type-2 nodes in their neighbourhood. Typically a radial distribution function or a middle point between nodes may be selected. These nodes will fix their tuning initially based on the parent nodes and later tune their parameters corresponding to the inputs over period of time. Spawning allows more connections to be created which will increase the number of possible paths.

(vi) Multiple paths will always exist. As specific paths are explored, nodes get added to higher layers of nodes, which now can implement path optimization based on the segment length. Nodes recognizing shorter paths will have lower cost.

(vii) Higher layers will combine successively larger number of segments and provide better selection among paths. Identification of chords requires edge nodes to be spawning more nodes in response to repeated input. Hebbian learning allows recognizing such paths by repeated use.

4. RESULTS

As the process of finding the path is dynamically chosen, it is possible to circumvent any obstructions as and when they arise. Hierarchical ARN can locate paths around the obstacles as shown in Figure. 24. Obstructions can be introduced simply by blocking the cells inside the obstruction area. Edges that are inside or cross the obstruction are also blocked. In the following figures, the source is represented by orange dot and the destination is represented by green dot. Dots in the red and blue color indicate the obstructed nodes and the free nodes respectively. Cubes represent the obstructed area. We can observe that the best path (thick line) has the least path length and least cost as compared with the other paths.

Figure 24 shows a typical simulation result of the hierarchical network shown in Figure 15 for a 3 segment joint system with 6-DoF, using spherical joints shown in Figure 12. Nodes are shown as blue dots. Red volumes indicate where paths are blocked. Some of the selected paths between orange and green nodes at extreme left and right are shown as lines. A background grid has been included to give an idea of work space. Actual arm has not been shown to avoid clutter. The network is able to avoid obstacles and find traversable paths and select best path among the possible paths. Responses for a 5 segment joint system also show very similar results.

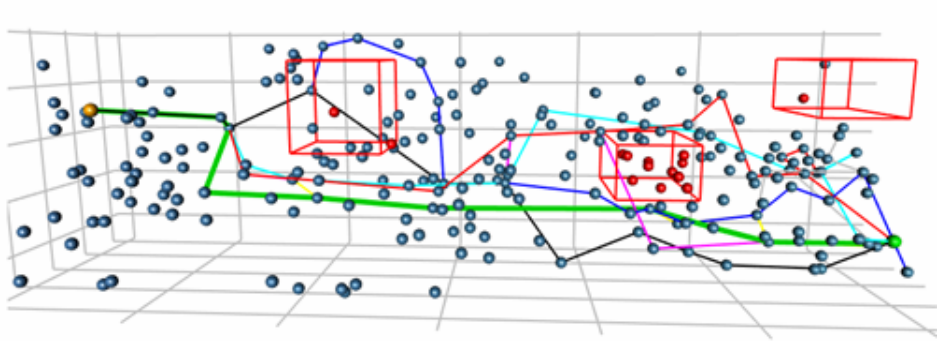
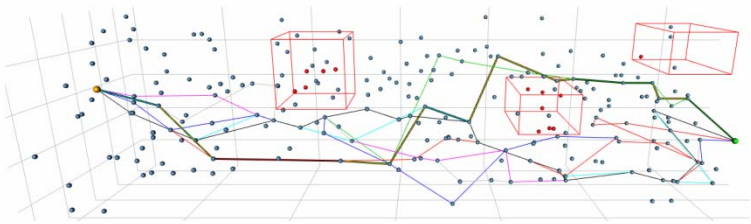
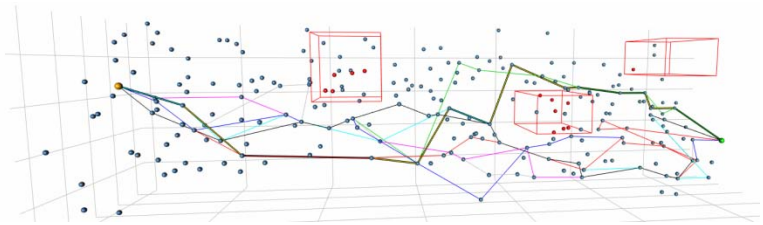


Figure 24. Moving end effecter along optimal paths while avoiding obstacles



(a) A 3-D view of a path

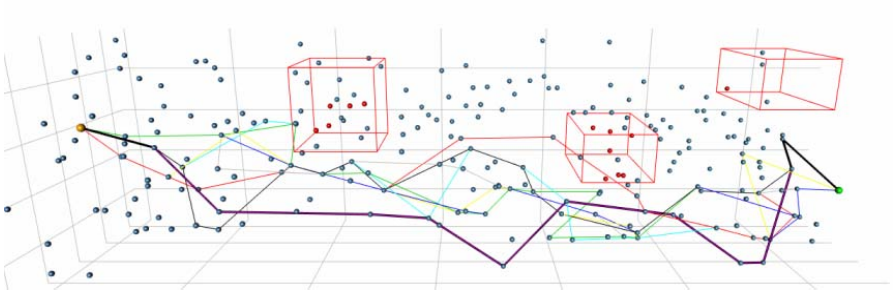


(b) A slightly rotated view of the same path

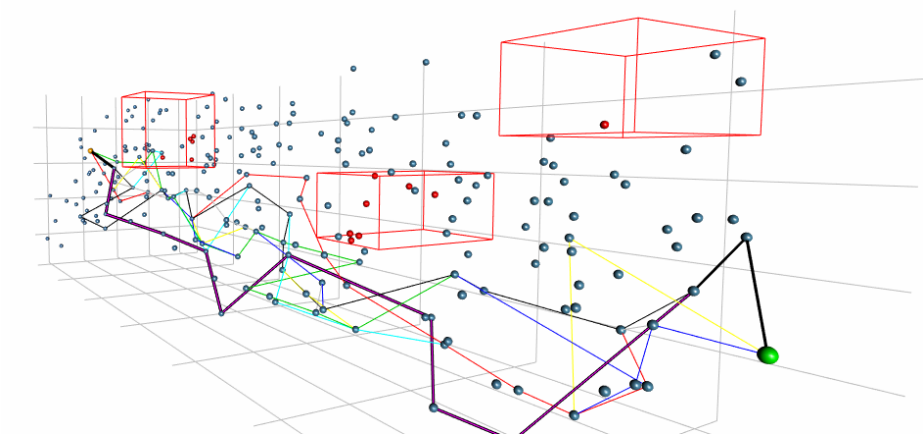
Figure25. Best solution among multiple optimized solutions of a three segment spherical joint with obstructions

The effect of obstructions is shown in Figure 25 (a). The results are rotated by an angle in Figure 25 (b) to show that the joint is not traversing though the obstructed area. Different paths are indicated by different colored lines and the best path is indicated by a thick line.

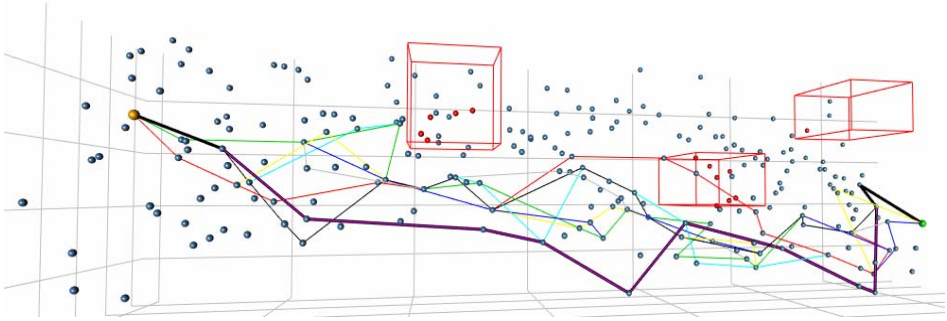
Alternate paths identified during separate runs may be seen by comparing the paths shown in Figures 24, 25 (a) and 26 (a), all of which have same starting and end points. Results are rotated by some angle in Figures 25 (b), 26 (b) and 26 (c) to show that the path is avoiding the obstructions. During training, nodes are presented in random order and hence the creation of nodes is not always identical (see Figure 26). That results in creation of different paths and costs. However, these solutions have similar costs. Results of several runs may be combined in to a single set. Folds may be used to store alternate possible paths.



(a)



(b)



(c)

Figure26. Best solution among multiple optimized solutions of a three segment spherical joint with obstructions

It is observed that the temporal behaviour of neurons give rise to many such effects that can be related to a path related algorithm. Therefore, we consider the path model of brain as a critical model to be studied further. A hierarchical network for image recognition built using ARN has been reported in (Pavithra et al., 2019).

R-Code for all the models discussed and presented here may be downloaded from our research web site cqserver.in/download. Publications and Code Examples are separately listed. Under publications, links to our papers published in journals and conference papers are available. Other publications are in PDF format. Code examples are compressed in zip or tar format. The code examples can be downloaded and next the files decompressed. Most of the code examples have a file indicating how to use the code.

5. CONCLUSION

A new structure for neural network called Auto Resonance Network is presented. A single layer ARN can be used to classify convex data sets. Deep learning hierarchical structure using Hebbian learning has been used with ARN to simulate a robotic system that can learn to move in a workspace with obstacles. Advantages of the proposed network are its simplicity and scalability. Every node in ARN corresponds to a small volume in input space with a controlled coverage space around a resonating center. To a certain extent, the network is similar to RBF network but with different control algorithm. ARN can associate input and output using a Hebbian like learning mechanism. The size of the network grows with input. An illustrative example of a robotic arm is presented. There is no limitation on the number of segments present in the joint or the size of output space other than the performance issues. The network provides a tradeoff between accuracy and number of nodes required. The system learns from the environment continuously, growing with experience over period of time.A neuronal interpretation of the connection layer and path layers in the hierarchical ARN, generally called Pathnet, has been presented. ARN and Pathnet can be used as a general purpose classifier and AI tool. This provides new ways to explore many complex problems in AI. Searching method used here can also be easily adapted for semantic webs and natural language processing. It is possible to develop many application for modern gadgets using ARN.

REFERENCES

Aparanji, V. M., Uday V. W., and Aparna, R., 2016, A novel neural network structure for motion control in joints, IEEEXplore digital library, 227-232.

Aparanji, V. M., Uday V. W., and Aparna, R., 2017a, robotic motion control using machine learning techniques, IEEEXplore digital library,1241-1245.

Aparanji, V. M., Uday, V. W., and Aparna, R., 2017b, Automated path search and optimization of robotic motion using hybrid ART-SOM neural networks, Springer LNNS, 415-423.

Aparanji, V. M., Uday, V. W., and Aparna, R., 2018, Pathnet: A neuronal model forRobotic Motion Planning, Springer CCIS801, 386–394.

Bing, Z.-S., Meschede, C., Rahrbein, F., Huang, K., and Knowll, A. C., 2018, A survey of robotics control based on learning-inspired spiking neural networks. Frontiers in Neurorobotics, 12, 12-35.

Cruz, F., Magg, S., Weber, C., and Wermter, S., 2016, Training agents with interactive reinforcement learning and contextual affordances. IEEE Transactions on Cognitive and Developmental Systems, 8, 4, 271-283.

Houston, R. L., and Kelly, F.A., 1982, The development of equations of motion of single-arm robots. IEEE Transactions on Systems, Man, and Cybernetics, 12, 3, 259-266.

Ján, V., 2012, Adaptation of fuzzy cognitive maps by migration algorithms. Kybernetes, 41, 3-4, 429-443.

Kober, J., B, A. J., and Peter, J., 2013, Reinforcement learning in Robotics: A survey. International Journal of Robotics Research, 32, 11, 1238-1278.

Li, J.-J., Li, Z.-J., Chen, F., Bicchi, A., Sun, Y., and Fukuda, T., 2019, Combined sensing, cognition, learning, and control for developing future neuro-robotics systems: A survey. IEEE Transactions on Cognitive and Developmental Systems, 11, 2, 148-161.

Martin, A, E., and Robert, G. D., 2016, Incorporating human-like walking variability in HZD-based bipedal model. IEEE Transactions on Robotics, 32, 4, 943-949.

Masumeh, S., Rashedi., E., Dashti, S. M., and Hakimi, A., 2017, Ideal gas optimization algorithm. International Journal of Artificial Intelligence, 15, 2, 116-130.

Mayannavar, S., and Uday, W., 2019, A noise tolerant auto resonance network for image recognition, CCIS, Springer, 4th International Conference on Information, Communication and Computing Technology, IIC, New Delhi.

Otaduy, M. A., and Lin, M. C., 2006, A modular haptic rendering algorithm for stable and transparent 6-DoF manipulation. IEEE Transactions on Robotics, 22, 4, 751-762.

Pavithra, U., Sneha, M. R., SagarSrivatsa, S. R., Ravichandra, T., and Aparanji, V. M., 2019, OCR system for automating answer script marks using auto resonance network, IEEE Xplore digital library, 0857-0861.

Pozna C., Precup, R.-E., Tar, J. K., Škrjanc, I., and Preitl, S., 2010, New results in modelling derived from Bayesian filtering. Knowledge-Based Systems, 23, 2, 182-194.

Precup, R.-E., and David, R.-C., 2019, Nature-Inspired Optimization Algorithms for Fuzzy Controlled Servo Systems, Butterworth-Heinemann, Elsevier, Oxford, UK.

Purcaru, C., Precup, R.-E., Iercan, D., Fedorovici, L.-O., David, R.-C., and Dragan, F., 2013, Optimal robot path planning using gravitational search algorithm. International Journal of Artificial Intelligence, 10, S13, 1-20.

Schmidhuber, J., and Hochreiter, S., 1997, Long short-term memory. Neural Computation, 9, 8, 1735-1780.

Spall, J. C., 1992, Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. IEEE Transactions on Automatic Control, 37, 3, 332-341.

Trojanová, M., and Hošovský, A., 2019, Comparison of different neural networks models for identification of manipulator arm driven by fluidic muscles. Acta Polytechnica Hungarica. 15. 7-28.

Veslin, E. Y., Dutra, M. S., Lengerke, O., Carreno, E. A., and Tavera, M. J. M., 2014, A hybrid solution for the inverse kinematic on a seven DOF robotic manipulator. IEEE Latin America Transactions, 12, 2, 212-218.

Wei, H, Chen, Y.-H., and Yin, Z., 2016a, Adaptive neural network control of an uncertain robot with full state constraints. IEEE Transactions on Cybernetics, 46, 3, 620-629.

Wei, H., and S., C.-Y., 2016b, Adaptive neural impedence control of a robotic manipulator with input saturation. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 46, 3, 334-344.

Wei, H, Ouyang, Y.-C.., and Jong, J., 2017, Vibration control of a flexible robotic manipulator in the presence of input dead zones. IEEE Transactions on Industrial Informatics, 13, 1, 48-59.

Yangqing, J., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T., 2014, Caffe: Convolutional Architecture for Fast Feature Embedding, arXiv:1408.5093v1 Cornell University[cs.CV] 20 Jun 2014.

Zall, R., and Kangavari, M. R., 2019, On the construction of multi-relational classifier based on canonical correlation analysis. International Journal of Artificial Intelligence, 17, 2, 23-43.