Applied Soft Computing 49 (2016) 352–364

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Neural networks to determine task oriented dexterity indices for an underwater vehicle-manipulator system

Panagiotis Sotiropoulos*, Nikos Aspragathos

Mechanical Engineering and Aeronautics Department, University of Patras, Patras 26500, Greece

ARTICLE INFO

Article history:
Received 4 February 2015
Received in revised form 2 April 2016
Accepted 18 August 2016
Available online 21 August 2016

Keywords:
Dexterous task execution
Feed-forward back-propagation neural networks
Radial basis function neural networks
Fast approximation of high complexity function
High performance of underwater vehicle-manipulator systems

ABSTRACT

A method for the fast approximation of dexterity indices for given underwater vehicle-manipulator systems (UVMS) configurations is presented. Common underwater tasks are associated with two well-known dexterity indices and two types of neural networks are designed and trained to approximate each one of them. The method avoids the lengthy calculation of the Jacobian, its determinant and the computationally expensive procedure of singular value decomposition required to compute the dexterity indices. It provides directly and in a considerably reduced computational time the selected dexterity index value for the given configuration of the system. The full kinematic model of the UVMS is considered and the NN training dataset is formulated by the conventional calculation of the selected dexterity indices. A comparison between the computational cost of the analytical calculation of the indices and their approximation by the two NN is presented for the validation of the proposed approach. This paper contributes mainly on broadening the applications of NN to a problem of high complexity and of high importance for UVMS high performance intervention.

1. Introduction

Unmanned Underwater Vehicles (UUV) are being used nowadays in a wide range of underwater operations. Common UUV tasks include manipulation of valves and switches on underwater facilities such as control panels on hydrocarbon underwater sites, inspection and maintenance of subsea structures, object recovery and survey of the sea bottom. A vast majority of tasks require a vehicle equipped with a manipulator, composing a redundant Underwater Vehicle Manipulator System (UVMS). Remotely Operated Vehicles (ROVs) and Intervention Autonomous Underwater Vehicles (IAUV) are typical UVMS. ROV are robot submersibles connected to a mother ship by a coaxial cable, transferring power and data, and operated by experienced pilots, whereas IAUV are deployed by their mother ship and operate untethered.

The term autonomous operation is synonymous with the IAUV operations, though current trends indicate that more and more ROV mission components tend to become autonomous. Online motion planning is a key element of every underwater mission and an efficient motion plan that could be adapted according to sensor readings is essential for an IAUV mission. Though, such a motion planning algorithm could also benefit an ROV mission leaving the pilot with a supervisory role.

A common underwater mission scenario for a UVMS on an underwater site is illustrated in Fig. 1, where the vehicle approaches a control panel equipped with control valves, commonly used in underwater hydrocarbon facilities.

Usually in a single mission several interventions have to be made, referred to as tasks and subtasks. The vehicle could either dock near the task area to perform the intervention or navigate while executing it. In the latter case, due to the unconstrained movement of the main body of the UVMS, apart from the degrees of freedom (dof) of the manipulator, there are six more dof added to the system that could allow the UVMS use additional configurations to achieve a given end-effector pose and subsequently a trajectory for a given task. An optimal motion planning algorithm should therefore make use of the redundancy of the UVMS in order to perform the mission in the most efficient manner.

Depending on the task in hand there exist several indices that could quantify the system's ability to either move the end-effector with high speed, or apply force or even move with accuracy around a certain point in the workspace. There are numerous studies in the literature regarding dexterity indices for stationary robot manipulators. Yoshikawa [1] proposed dexterity indices based on the kinematic and the dynamic manipulability ellipsoid in order to provide a quantitative measure of a robot's ability to achieve high speed of the end-effector or apply great force on the environment.

* Corresponding author.
E-mail addresses: psotirop@upatras.gr, psotirop@mech.upatras.gr (P. Sotiropoulos), asprag@mech.upatras.gr (N. Aspragathos).

http://dx.doi.org/10.1016/j.asoc.2016.08.033
1568-4946/

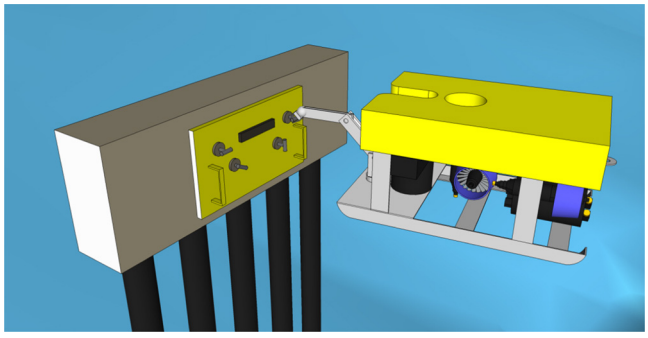

Fig. 1. UVMS approaching hydrocarbon facility control panel.

The conditioning number of the Jacobian was used by Salisbury and Craig [2] and provided a measure of isotropy that indicated the ease of the end effector to move towards any direction from the current configuration. The kinematic conditioning index (KCI) defined as the inverse of the condition number was introduced by Angeles and Lopez-Cajun [3] and could be considered as a measure of the end effector positioning and orientation accuracy.

Though, the Jacobian matrix contains elements for translational and rotational joints as well as for the translational and rotational speeds of the end-effector. Yoshikawa solved that issue by normalising the Jacobian using the upper limits of the joint's and the end-effector's speeds. An alternative approach would be to examine the dexterity index by separating the Jacobian into its translational and rotational components. Cardu et al. [4] proposed a modified dexterity index based on the decomposition of the Jacobian into a translational and a rotational matrix to evaluate separately translational and rotational speeds on the end effector.

Taking into account the directionality of certain robot tasks Dubey and Luh [5] have introduced the manipulator velocity ratio (MVR) to quantify the ability of the end-effector to move towards a given direction. The term of manipulator mechanical advantage (MMA) was also introduced in the same publication, to quantify the end-effector's ability to apply force along a given direction.

Later works examined the use of dexterity to retrieve dexterous poses for mobile vehicle manipulator systems (MVMS). Tchon and Zadarnowska [6] have proposed a local and a global dexterity measure based on an endogenous configuration space approach, to quantify the MVMS dexterity and provide optimal poses for mobile manipulators. Bayle et al. [7] studied the extension of the manipulability introduced by Yoshikawa for a non-holonomic MVMS relating it to the shape of the manipulability ellipsoid.

Regarding UVMS, the importance of dexterity indices in motion coordination, while executing a task, was pointed out by Padir and Nolff [8], where they have studied the use of a dexterity index for the case of two cooperating UVMS. The index was based on the manipulability and it was formed to describe the manipulability of the system of the two vehicles for a rigid object transportation task. Pseudovelocities were introduced to incorporate the coupling constraints into the kinematic equations of the whole system. Jun et al. [9] proposed a method to provide a UVMS with optimal configurations, while executing a given task, based on the task oriented manipulability measure. Asokan et al. [10] proposed a method to retrieve a docking pose for a UVMS while executing a welding task that would maximise the end-effector's workspace manipulability. The dexterity of the manipulator was considered in the optimization criteria for the determination of the docking pose on the structure under constraints. The importance of a well selected docking pose was discussed, since it would allow the vehicle to avoid singular configurations, while performing the tasks and therefore to avoid possible re-docking that would cost in terms of time and energy. Using an extended dexterity index Sotiropoulos et al. [11] have studied the optimal docking pose for a UVMS, on an underwater facility control panel where the vehicle had to execute a series of manipulation tasks. The dexterity measure of Yoshikawa was used to formulate the Area Manipulability Measure (AMM) in order to retrieve an appropriate docking pose for the vehicle that would allow high values of manipulability inside its task area.

The majority of the dexterity indices proposed so far are based on the Jacobian matrix either by calculating its determinant or by performing singular value decomposition (SVD) to determine its singular values. However, even the analytical calculation of the Jacobian matrix, requires a great number of mathematical operations, since each element of the Jacobian includes complex trigonometric functions of the joint variables. The computational issues for the motion planning of UVMS were considered early on by Quinn and Lane [12] and therefore a fast calculation of the indexes could benefit motion planning algorithms in general.

During the last decades, Neural Networks (NN) have been widely used for complex linear and nonlinear function approximation problems. Various NN architectures have been proposed in the literature for robotic applications and especially for the approximation of the robot kinematics model [13]. Numerous studies have been dedicated to the use of an appropriate NN architecture for the solution of the inverse kinematics problem of serial manipulators [14–16] while the forward kinematics problem has been addressed mainly for parallel robot applications [17]. The use of NN in the aforementioned studies, aims to reduce the computational cost for the calculation of the system's kinematics, when an analytical solution is hard to be found. While re-examining the problem of UVMS docking Sotiropoulos et al. [18] have proposed a NN for the fast calculation of the dexterity on the task area, avoiding the calculation of the vehicle's inverse kinematics. The use of a NN provided computational time gain on the calculation of the AMM proposed in [11], however the environment in this case was constrained by the nature of a docking operation and the dimensions of the problem were reduced to the pose of the docking probe (y, z and θ) with respect to the docking plane.

On an underwater mission, the computational potential of UVMS is limited to the system's on-board resources. Given that the vehicle should operate in a rather dynamic and uncertain environment, the UVMS should be able to update the motion plan based on its sensor readings rapidly. The online generated motion plan should ensure efficient task execution, thus appropriate dexterity indices should be considered. Formulating the Jacobian matrix, candidate configurations could be graded according to the dexterity index selected. The calculation of the Jacobian matrix and the dexterity indices derived from it may be a straightforward procedure, though, the computational gain of using NN to approximate dexterity indices rapidly should be investigated. Possible time gain would reduce the computational cost of any motion planning algorithm for efficient task execution accordingly.

In this paper we test two types of NN for the fast calculation of two dexterity indices directly from the configuration of the UVMS. The two types of NN used, are the feed forward and the radial basis function networks. Basic tasks that could be possibly undertaken by a UVMS in an underwater mission are examined and related with the two dexterity indices, the manipulability measure w and the kinematic conditioning index KCI. Both types of NN have been designed and trained to approximate the index value. The full kinematic model of the UVMS is taken under consideration when designing the NN and thus the entire set of dof could be utilised on a dexterous motion planning algorithm.

Moreover, a comparison is performed between the algebraic operations required by the NN approximation and those required by the analytical calculation of the indices. To demonstrate the time gains of the proposed approach the computational time the calculation of the dexterity indices by a numerical method that uses the

analytical expression of the Jacobian and well known algorithms for the calculation of its determinant and SVD, is compared with the NN method.

The contribution of this paper is towards the application of NN in a highly demanding problem such as the optimum motion planning for a high-dof system able to perform an autonomous intervention task. The approximation of widely used dexterity indices by NN would provide a valuable tool for efficient online motion planning algorithms and reduced cycle time during interventions with a minimal computational cost.

The rest of the paper is divided in the following sections: Section 2 presents the kinematics of a UVMS and the association of the dexterity indices to certain underwater tasks. In Section 3 the two types of NN architectures selected for this work are described in detail. In Section 4 the performance of the networks is presented and discussed.

2. Kinematics and UVMS dexterity

2.1. Description of the UVMS

UVMS systems in general are fully actuated systems, though passive by design in roll and pitch angles [19]. A representative example of such a vehicle used mainly for scientific operations is the Jason ROV of the Woods Hole Oceanographic Institution, equipped with six propellers actuated by DC motors that can move the vehicle in any direction [20].

The system described here is composed by a holonomic vehicle and a six dof manipulator attached to its front side resulting on a total twelve dof system. Regarding the UVMS base movement the three translational and three rotational dof are modelled as translational and rotational joints attached to the centre of gravity (CoG) of the vehicle. The vector θ describing the displacements of the vehicle and of the manipulator's joints is given by:

$$\theta = [x, y, z, \theta_x, \theta_y, \theta_z, \theta_{m1}, \theta_{m2}, \theta_{m3}, \theta_{m4}, \theta_{m5}, \theta_{m6}]$$

where x, y and z refer to the base translation, θ_x, θ_y and θ_z refer to the base rotation according to XYZ Euler angles representation and $\theta_{m1},\ldots,\theta_{m6}$ refer to the manipulator's joint displacements.

The movement of every joint (θ_n) could be described with a twist (ξ_n) [21]. The twist for a rotational joint is given by:

$$\xi_n = \begin{bmatrix} -\omega_n \times q_n \\ \omega_n \end{bmatrix} \quad (1)$$

and the twist for a translational joint by:

$$\xi_n = \begin{bmatrix} k_n \\ 0 \end{bmatrix} \quad (2)$$

where ω_n, k_n and q_n denote the unit vector for rotation, the unit vector for translation and a point on the axis of rotation respectively.

The UVMS's model, with the axes of translations k and rotations ω for every joint along with the CoG of the vehicle and the manipulator base point (MB), is illustrated in Fig. 2. The axes of translation and rotation of the UVMS joints at the reference configuration are given by:

$$k_1 = [100]^T, k_2 = [010]^T, k_3 = [001]^T$$

$$\omega_4 = [100]^T, \omega_5 = [010]^T, \omega_6 = [001]^T$$

$$\omega_7 = [001]^T, \omega_8 = [-100]^T, \omega_9 = [-100]^T$$

$$\omega_{10} = [001]^T, \omega_{11} = [-100]^T, \omega_{12} = [010]^T$$

While the points considered on the respective axes are:

$$q_1 = [0, 0, 0]^T,$$

$$q_2 = [0, y_{MB}, l_1]^T,$$

$$q_3 = [0, y_{MB} + l_2, l_1]^T,$$

$$q_4 = [0, y_{MB} + l_2 + l_3, l_1]^T$$

where y_{MB}, is the distance along the y-axis from the CoG to the manipulator base (MB).

2.2. UVMS kinematics

The forward kinematics problem for a robotic system refers to the determination of its end-effector pose for a specified set of joint values using the systems kinematic equations. Rigid body transformations can be represented by the exponential of a twist and the final pose of the end effector could be defined by the product of exponential of the twists for the presented platform-manipulator system:

$$g_{st}(\vartheta) = e^{\xi_1\theta_1} \cdot \ldots \cdot e^{\xi_{12}\theta_{12}} g_{st}(0) \quad (3)$$

$$g_{st}(0) = \begin{bmatrix} R(0) & p(0) \\ 0 & 1 \end{bmatrix} \quad (4)$$

where $g_{st}(0)$ refers to the end-effector pose in the reference configuration of the UVMS, $R(0)=I$ equals to $[3\times3]$ identity matrix that denotes the rotation matrix and $p(0) = [0, y_{MB} + l_2 + l_3, l_1]^T$ the position vector of the end-effector.

The Jacobian matrix relates the joint speed vector to the end-effector speed vector according to Eq. (5)

$$\dot{r} = J^s_{st}(\theta)\,\dot{\theta} \quad (5)$$

The Jacobian could be derived using the adjoint transformations of the twists as:

$$J^s_{st}(\theta) = \left[\xi_1, \ldots, \xi'_n, \ldots, \xi'_{12}\right] \quad (6)$$

$\xi'_n = Ad_{(e^{\xi_1\theta_1}\cdots e^{\xi_{n-1}\theta_{n-1}} g_{st}(0))}$ where, Ad_g denotes the adjoint of the matrix g.

It can be observed that the last joint θ_{12} does not affect the value of the Jacobian by its definition.

2.3. Dexterity indices related to underwater tasks

In this section the engineering meaning of the manipulability index and the kinematic conditioning index is presented, as well as their relation to some critical underwater tasks. Yoshikawa in [1] introduced the manipulability index based on the properties of the Jacobian. Since the Jacobian's elements are the coefficients of both linear and angular velocities, the matrix could be normalized as:

$$J(\theta) = UJ^s_{st}G^{-1} \quad (7)$$

where, $G = diag\left(1/\dot{\theta}_{1max}, \ldots, 1/\dot{\theta}_{12max}\right)$ and $U = diag\left(1/\dot{r}_{1max}, \ldots, 1/\dot{r}_{6max}\right)$ the diagonal matrices that contain the maximum joint velocities and the maximum end-effector velocities respectively.

The manipulability index w is given by:

$$w = \sigma_1 \cdot \sigma_2 \cdot \ldots \cdot \sigma_6 \quad (8)$$

Or alternatively by:

$$w = \sqrt{det\left(JJ^T\right)} \quad (9)$$

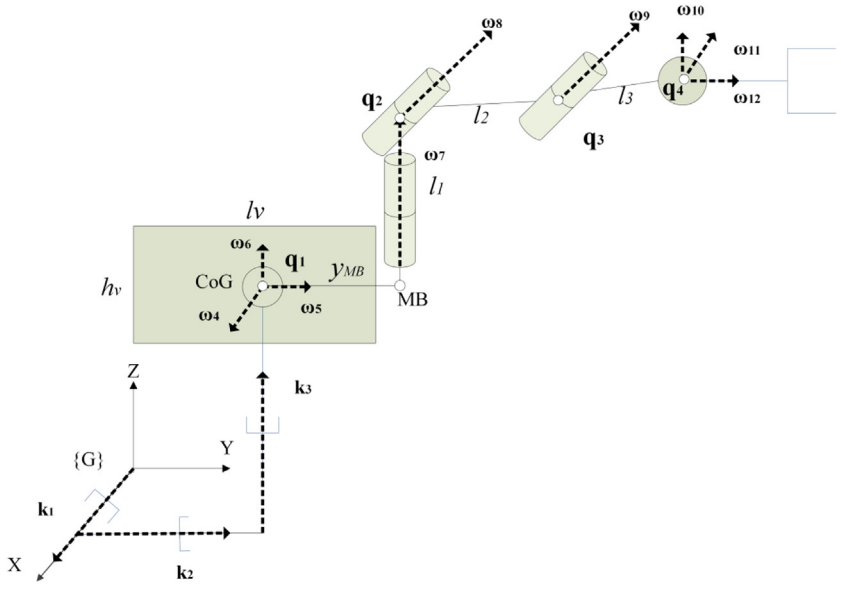

Fig. 2. UVMS at the reference configuration.

where *det* denotes the matrix determinant and $\sigma_1,\ldots,\sigma_6$ denote the singular values of the Jacobian matrix.

The analytical form of the manipulability index could be calculated using symbolic computation software and the analytical form of the Jacobian and its determinant are given in the Appendix A. Considering the eigenvectors $u_1,\ldots,u_6$ of the Jacobian an ellipsoid could be formed that its axes are given by $\sigma_1 u_1,\ldots,\sigma_6 u_6$. The manipulability index is proportional to the volume of the ellipsoid. In a similar manner one could define the force ellipsoid that is reciprocal to the manipulability ellipsoid. Across the manipulability ellipsoid's major axis $(\sigma_1 u_1)$ the end-effector could move with high speed, whereas across its minor axis $(\sigma_m u_m)$ its speed capability is reduced and it could apply higher forces. A zero value of the manipulability index would imply that the end-effector cannot move towards certain directions which could prove a great disadvantage while executing a task that requires high speed along these directions. In Fig. 3, the manipulability ellipsoid is demonstrated graphically by taking into account only the translational speeds of the end effector.

Angeles et al. [3] have proposed the kinematic conditioning index to grade the proximity of the system's configuration to isotropy, where the end-effector could move towards any direction with the same ease. The kinematic conditioning index is given by the ratio of the minimum versus the maximum singular value:

$$KCI = \frac{1}{k(J)} = \frac{\sigma_m}{\sigma_1} \tag{10}$$

For the special case that $\sigma_m = \sigma_1$ the ellipsoid becomes a sphere such as the one illustrated in Fig. 3 that lies inside the original manipulability ellipsoid.

During a UVMS intervention where the end-effector should accomplish a series of subtasks, the value of the manipulability index indicates its ability to move with high velocity on the end-effector, an attribute that could minimize the cycle time of a mission. On the other hand a value of KCI close to one, would indicate that the end-effector could move with the same ease in every direction and therefore such configurations could be utilized during interventions requiring accuracy on the end-effector. From the application's point of view the manipulability index could be associated with underwater operations whose velocity and force are of high importance.

There are several underwater operations that could be analysed depending on their requirements of speed, force/torque and accuracy by the end-effector. Such an analysis would help identify the

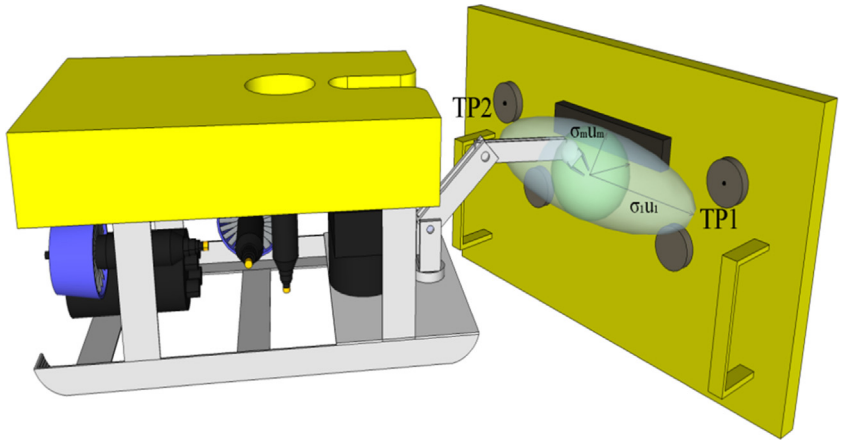

Fig. 3. UVMS close to the intervention area and manipulability ellipsoid.

Table 1
Indices associated with tasks.

Index	Task	Comments
Manipulability Index (w)	1. Manipulation valves/levers 2. Installation and replacement of structural components. 3. Object recovery 4. Debris removal 5. Multiple task execution 6. Cutting 7. Cleaning	Determine the configurations that the end-effector can have the high speed or apply great force.
Kinematic Conditioning Index (KCI)	1. Precise manipulation 2. High precision cutting 3. Welding 4. Small component replacement 5. Non-destructive testing	Evaluate the accuracy of the end-effector

appropriate dexterity index for a given task and rank configurations that could compose a dexterous motion plan.

For example in the area of repair and maintenance, operations such as cutting off damaged parts and cleaning underwater structures using water jet are quite common and require application of constant force by the system. Dexterous configurations could be considered for such a task, though for high precision cutting operations the value of KCI should also be taken into account.

Often UVMS should check and replace jumpers or perform specified non-destructive testing on given components, operations that would benefit from a high level of accuracy by the end effector. Welding operations are also part of the repair and maintenance of underwater installations where the accuracy of the end effector is of great importance, since the speed should be kept constant across the weld and the end effector should keep close to the surface.

In the area of general underwater manipulations, like manipulation of valves and levers on underwater control panels, the end-effector has to apply torque about a certain axis, whereas on debris removal operations the application of excessive force would be required. On the case of underwater archaeological sites and fragile object recovery the accuracy is rather important. Furthermore, in any case where multiple tasks are to be performed in a small task area, the time required from the end-effector to travel from one task to the next should be minimized and therefore configurations with great speed potential should be preferred. Table 1 presents a mapping of selected indices to UVMS tasks such as manipulations, inspections and interventions on underwater structures.

Following this association the importance of dexterity indices in motion planning for task execution becomes rather obvious. However their computational cost should remain at a minimum level in order to be used in practical real world missions, where a motion planning algorithm would examine several hundreds of configurations on every cycle. Even the slightest improvement in terms of computational time for the dexterity indices, would lead to significantly reduced computational time for the planner.

3. Neural networks for dexterity indices approximation

In this paper two types of artificial neural networks are examined for the rapid calculation of the selected dexterity indices. The two types of NN are the feed-forward back-propagation networks (FFBN) and the radial basis function networks (RBFN). Both types of NN are widely used for function approximation problems [13]. Recent studies [22] have demonstrated that hyper-basis function networks (HBFN) could be an alternative to classic RBFN providing scaling to local input dimensions and resulting to compact and reliable networks. However the discussed method could produce unnecessarily large networks for certain problems and the HBFN were not considered further for this particular study.

FFBN can be easily designed having sigmoid neurons on their hidden layer and linear neurons in the output layer. In their training procedure, using back-propagation methods, a significant amount of time is required for network convergence. The convergence time depends on the size of the training set and the number of neurons used for the model.

RBFN are designed by appropriately selecting data from the training dataset and an optimal spread value for the activation of the radial basis function of each neuron, in order to cover the search space effectively. RBFN tend to have more neurons than a comparable FFBN, since sigmoid neurons can have outputs over a large region of the input space, while the radial basis function neurons could only respond to relatively small regions of the input space depending on their spread value (*spread*). The result is that the larger the input space (in terms of number of inputs, and the ranges those inputs vary over) the more neurons required for an RBFN.

There are two general approaches on the training of neural networks, the batch training approach and the sequential training approach. On the sequential approach the network is adapted on an example by example basis whereas on the batch training the entire set of training examples is presented and the network is adjusted on an epoch-by epoch basis.

For on-line applications and towards the minimization of the neurons used by a RBFN, Huang et al. in [23] have introduced the concept of significance for hidden neurons that would allow a neuron to be added on the network only its contribution to the network output is above a certain threshold. Vukovic and Miljkovic in [24] have discussed the use of standard Extended Kalman Filters to deal with outliers in heavy tail noise data. They use a sequential training approach and the network could easily be adapted in case new data become available without the need to memorise the previous data.

Even though sequential training could demonstrate better performance on online applications and pattern classification problems, the batch training approach remains better suited for nonlinear regression problems [25]. In this work given that the example dataset is well defined with no outliers present and since the network would not be updated online during the application, a batch approach has been followed during the training phases that are performed off-line.

3.1. Dataset

By adequately sampling the UVMS configuration space and by having determined the normalised Jacobian matrix of the system using Eqs. (6) and (7), the values of the dexterity indices are computed for each configuration based on Eqs. (9) and (10). It

should be noted that since J does not depend on θ_{12}, both the manipulability index and the kinematic conditioning index do not depend on its value. Therefore the last joint θ_{12} is not considered among the input variables. Furthermore, the value of the determinant of the matrix $[JJ^T]$ does not depend on the first three translational and the first rotational joints of the UVMS due to the base invariance property discussed by Gotlin and Troch [26] and Synodinos et al. [27].

For the *KCI*, a training dataset D_1 is formed using the joint values for each sample configuration that affect the value of the index, as inputs and the corresponding index values as outputs. Following the discussion held before the last joint is not considered among the training inputs. The *i-th* element of the training dataset is given by:

$$D_1^i = \left[\theta_1^i, \ldots, \theta_{11}^i | KCI\right] \tag{11}$$

Regarding w, the training dataset D_2 is formed considering the base invariance property and therefore the inputs contain only the seven joints that affect the value of the index. The *i-th* element of the dataset is given by:

$$D_2^i = \left[\theta_5^i, \ldots, \theta_{11}^i | w\right] \tag{12}$$

The design process for every network consists of training and testing phase. A set of test datasets T_j, (j = *1,2*), were created in a similar manner that would serve to validate the networks during the testing phase. For the sake of simplicity, the subscript j is dropped in the following section where the neural network architectures are described.

3.2. Neural networks structure

Having defined the training and testing datasets, the two neural network architectures are described briefly hereafter. In Fig. 4, a FFBN with an input layer, one hidden layer (*Layer 1*) and an output layer (*Layer 2*) is illustrated. Given an input γ the activation function of the first layer f_{TS} is a tangential sigmoid function given by Eq. (13), while the activation function of the second layer f_L is a linear function given by Eq. (14).

$$f_{TS}(\gamma) = \frac{e^{\gamma} - e^{-\gamma}}{e^{\gamma} + e^{-\gamma}} \tag{13}$$

$$f_L(\gamma) = \gamma \tag{14}$$

Given a training input vector $D_{in} = [\theta_1, \ldots, \theta_r]$ the *j-th* output of the hidden layer X_H^j would be:

$$X_H^j = f_{TS}\left(\sum_{i=1}^{r} (W_1^{i,j} \cdot D_{in}^i) + b_1^j\right) \tag{15}$$

where, r denotes the number of system joints considered in the training dataset and are equal to the input neurons.

The network's simulated output S_{out} would be:

$$S_{out} = f_L\left(\sum_{i=1}^{k} (W_2^i \cdot D_H^i) + b_2\right) \tag{16}$$

where S_{out} = *KCI* or w.

The second type of the NN architecture is composed of an input layer, a hidden layer (*Layer 1*) and an output layer (*Layer 2*). A typical RBFN is illustrated in Fig. 5. The activation function f_{RB} is a radial basis function given by:

$$f_{RB}(\gamma) = e^{-\gamma^2} \tag{17}$$

Again, given a training input vector $D_{in} = [\theta_1, \ldots, \theta_r]$, the *j-th* output of the hidden layer X_H^j would be:

$$X_H^j = f_{RB}\left(\left(\sum_{i=1}^{r} D_{\text{in}}^i - W_1^{j,i}\right) b_1^j\right) \tag{18}$$

The bias vector b affects the output of the radial basis function by amplifying or compressing it. The biases b of the hidden neurons are set to 0.833/*spread* that would give an output of 0.5 or larger for any distance that equals or is less than +/−*spread*. The network's simulated output S_{out} (S_{out} = *KCI* or w) is computed using Eq. (16).

3.3. FFBN design procedure, training and testing

The design process for the FFBN, is an iterative procedure for the determination of the minimum number of neurons in the hidden layer. It is composed of two coupled phases: the training and the testing.

Starting with a single neuron network the number of neurons is increased by one each iteration of the design phase and the derived NN is trained. During the training process the weights and the biases of the network are updated until the mean square training error (e_{mse}) falls below the desired training error (E_{mse}) or the training epochs reach the maximum limit (L_{epochs}). The e_{mse} for each stage of the training phase is given by:

$$e_{mse} = \frac{1}{q}\sum_{i=1}^{q}\left(S_{out}^i - D_{out}^i\right)^2 \tag{19}$$

where, S_{out}^i is the simulated network output given the training input vector D_{in}^i, D_{out}^i is the corresponding training dataset dexterity index value and q denotes the number of training vectors in the training dataset.

The network is trained using the Levenberg-Marquardt back propagation algorithm embedded in Matlab Neural networks toolbox. The Levenberg-Marquardt algorithm is designed to approach second-order training speed without having to compute the Hessian matrix and it is among the fastest methods for training moderate-sized feed-forward neural networks [28]. The algorithm is based on the computation of the network's Jacobian matrix J_n to update the weights, an approach that is less complex than the computation of the Hessian matrix on Newton methods. The update rule is given by:

$$W_{k+1} = W_k - \left[J_n^T J_n + \mu I\right]^{-1} J_n^T e \tag{20}$$

The Levenberg-Marquardt algorithm performs as a combination of a steepest descent algorithm and a Gauss-Neuton algorithm and switches between the two during the training process. For small values of the coefficient μ the equation approaches the Gauss-Neuton algorithm while for very large values it approaches gradient descent methods with a small step size. As described by Hagan and Menhaj in [29] the Levenberg-Marquardt algorithm is much more efficient than conjugate gradient methods and variable learning rate algorithms for networks that consist of no more than a few hundred neurons.

Following the training phase of the FFBN, testing is performed using the test dataset, to simulate the network. The simulated network outputs S_{out} are compared with dexterity index values (T_{out}) on the test dataset and the testing error (e) is given by:

$$e = \max_{i=1:t}\left|\frac{T_{out}^i - S_{out}^i}{T_{out}^i}\right| \tag{21}$$

where t denotes the number of test configurations.

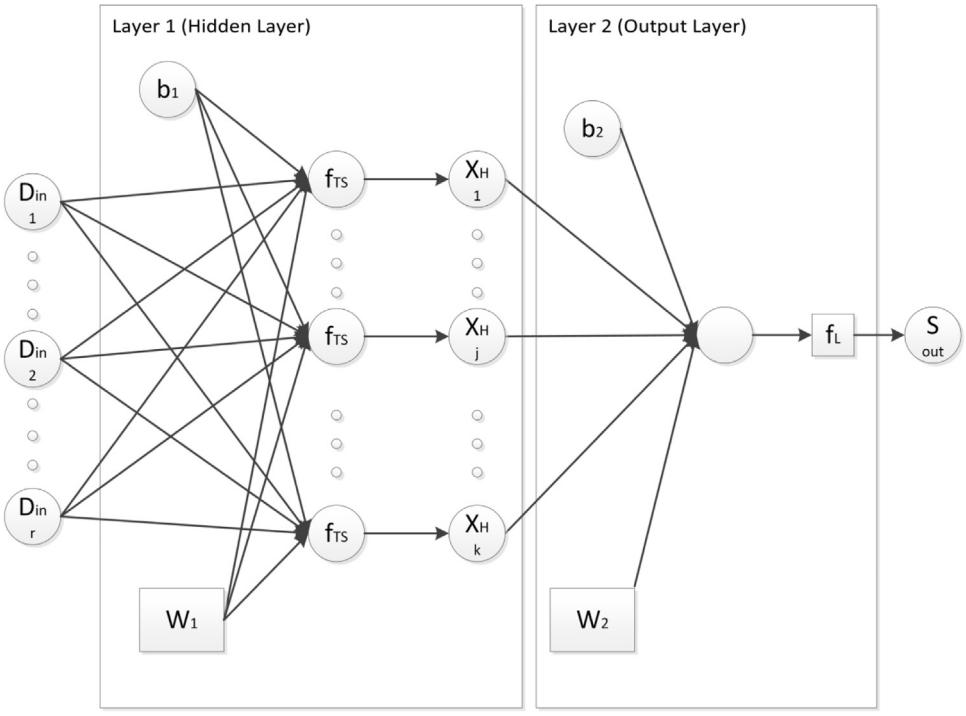

Fig. 4. FFBN architecture.

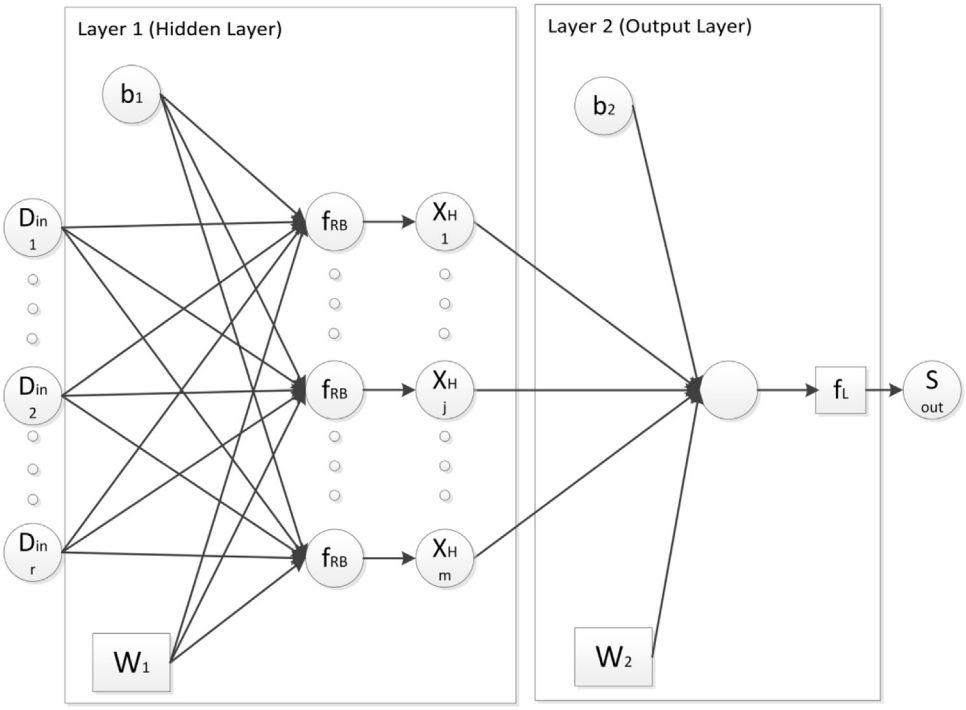

Fig. 5. RBFN architecture.

The design procedure terminates when the value of e for the given network falls below the maximum allowed testing error (E), or if the number of neurons reaches the design iteration limit (L_{iter}).

3.4. RBFN design process, training and testing

The design process for the RBFN aims to minimise the network's number of neurons in the hidden layer by finding an optimal spread value. The approach is based on a similar procedure as the one described before, though a second iteration is required. Given the training dataset D and an initial spread value, a RBFN is designed and trained. The training dataset D is used to simulate the network and compute the e_{mse}. The testing phase initiates when the e_{mse} falls below the training error threshold (E_{th}) to acquire a network with the minimum number of neurons for the given *spread* value. In principle the value of E_{th} is higher than the value of E_{mse}.

During the testing procedure if the value of e falls below E, the design process is terminated. In the opposite case, if the e_{mse} is

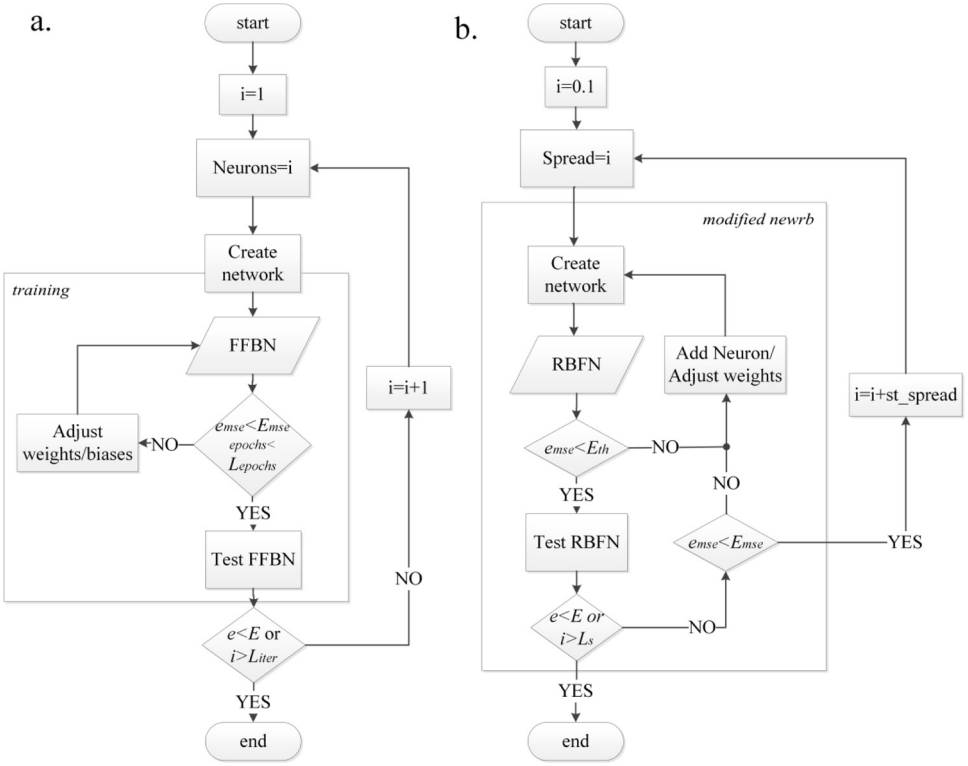

Fig. 6. a) FFBN design process flow chart b) RBFN design process flow chart.

Table 2
Vehicle parameters.

UVMS parameters
h_v = 1 m
l_v = 1.5 m
w_v = 1 m
r_{min} = 0.2 m
r_{max} = 1 m

above the desired error E_{mse}, a neuron is added in the hidden layer with weights equal to the values of the input vector with the biggest output error and the updated network is simulated again. If not, the *spread* value is increased by a predefined amount (*st_spread*) and the procedure starts over until the spread value reaches the maximum spread limit (L_s). The procedure is based on a modified version of the *newrb* function (a built-in Matlab function) where the testing stage is incorporated into the training process.

The flowcharts for the combined design and training process for both FFBN and RBFN are illustrated in Fig. 6.

4. Performance and comparison of neural networks results

4.1. Problem parameters

In order to compare the two neural network architectures, a UVMS was considered with its parameters given in Table 2. $r_{\max}$ and $r_{\min}$ denote the radiuses of the maximum and the minimum boundary of the manipulator's workspace, whereas h_v, l_v and w_v denote the height, the length and the width of the vehicle respectively. The parameters used in the design procedure are given in Table 3.

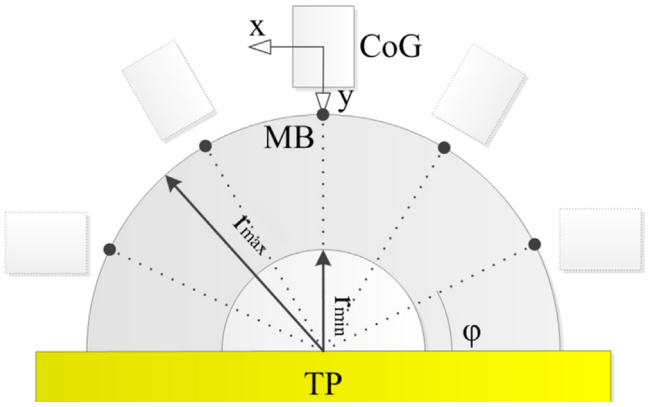

Fig. 7. Sample area around a task point (TP) on x-y plane.

4.2. Data sampling

For the first six joints that represent the pose of the UVMS's base, there are no physical constraints such as the ones imposed on the manipulator's joints. However, the pose of the base should be such that would allow the intervention point to remain inside the manipulator's workspace during the entire task execution.

Given a random task point (*TP*) as the one in Fig. 7, only configurations that allow the manipulator's base (*MB*) to remain inside the illustrated area were considered in the search space. The illustrated search area is defined by $r_{\max}$ and r_{min}. Therefore, during data sampling the ranges of the translational joints θ_1, θ_2 and θ_3 of the UVMS that correspond to the *CoG* displacement were restricted accordingly. In Fig. 7, $\phi = 30^\circ$ denotes the sampling angle inside the search area, where the positions of the base joints of the system were considered. The remaining configuration space was sampled in a range of 30°, from −15° to 15° with a step of 5°, per manipulator joint in an attempt to reduce the amount of data. Nevertheless the findings

Table 3
Design procedure parameters.

Design parameters	RBFN-*w*	RBFN-*KCI*	FFBN-*w*	FFBN-*KCI*
E (testing error)	0.02	0.02	0.02	0.02
E_{mse} (training error)	10^{-6}	10^{-6}	10^{-3}	5×10^{-6}
E_{th} (error threshold)	10^{-2}	10^{-4}	–	–
L_{iter} (iteration limit/number of neurons)	–	–	100	100
L_{epochs} (epochs limit)	–	–	100	100
L_s (spread limit)	10	10	–	–
T (number of test configurations)	1000	1000	1000	1000

of this work could be extended to cover the entire configuration space. For the case of the manipulability index approximation, only the joints that have an impact on the index's value are considered, namely joints θ_5 to θ_{11}.

It should be noted that after the training of the network the *TP* could correspond to any point inside the UVMS workspace and the trained network could approximate the dexterity index for every configuration once the dataset is extended.

4.3. Neural networks designed to approximate the manipulability index (w)

Both neural network architectures proved capable of approximating the indices within the acceptable error E during the design process, though as expected the RBFN procedure led to networks larger than the FFBN. The best RBFN found had 149 neurons in its hidden layer, while the corresponding FFBN managed to approach the index using only 8 neurons in its hidden layer.

Considering Eqs. (13)–(18), one could compute the number of operations required by each network to approximate the dexterity index. Given that the computational time for multiplications and additions differs depending on the available hardware and the accuracy required, in this paper these are considered as equivalent operations and the total number of operation presented is based on this assumption. Though, the number of addition and multiplication operations is also presented separately. For the case of w, the total number of operations for the FFBN network rises to 184 from which 80 additions and 104 multiplications, while for the RBFN the total number of operations required is 4172, from which 1788 additions and 2384 multiplications.

The performance of the networks during the design process for the RBFN is illustrated in Fig. 8. In Fig. 9 the training convergence to the best network derived from the design procedure is illustrated. As the spread value increases more neurons have a significant impact on the network's output and the network is able to generalise and have better performance on the test dataset.

Fig. 10 demonstrates the design process for the FFBN, while Fig. 11 shows the training procedure for the best network configuration. In this case the small number of neurons would provide a network that would generalise better.

The performance of each new network depends highly on the initialisation of the weight's matrices. This could explain why the networks that have six and seven neurons in the hidden layer give inferior results than the one with eight neurons that was finally selected, while the networks with 4 and 5 neurons in the hidden layer give good results.

4.4. Neural networks designed to approximate the kinematic conditioning index (KCI)

The types of neural networks used to approximate the KCI were trained using a significantly bigger dataset. As a consequence it can be observed from the results that, regarding the RBFN a larger spread value (*spread = 8.9*) was required for the network to achieve the desired performance while for the FFBN a larger number on neurons (25 neurons), compared to the number of neurons of the FFBN used to approximate w, had to be added in the hidden layer for the network to converge. It can be noted that both types of networks managed to reach the testing error E during the design procedure. The total number of operations in this case for the FFBN network rises to 775 from which 350 additions and 425 multiplications, while for the RBFN to 3976 from which 1704 addition operations and 2272 multiplication operations.

Table 4
Number of operations required per index.

	Analytical Computation	FFBN	RBFN
w	21810	184	4172
KCI	21594	775	3976

Figs. 12 and 14 demonstrate the design procedure for the RBFN and FFBN respectively while Figs. 13 and 15 illustrate the training process for the best network in each case.

On the design process and for spread values higher than 1.5, there are RBFN with testing error less than 5%. Though, the limitation on the number of neurons (L_{iter}) on the hidden layer does not permit these networks to reach E during testing.

Having designed the NN, the total number of operations required for the best NN of each case is compared with the operations required by the analytical computation of the dexterity indices in Table 4. It should be noted that the number of operations for the analytical computation of KCI is approximated, as stated in the Appendix A.

It can be easily observed that the number of operations required by the analytical approach is significantly higher than the one required by both NN architectures. As expected, the FFBN approach requires less operations since the number of neurons in every case is considerably smaller (Table 5).

4.5. Computational time results

Apart from the obvious computational gain, due to the considerable difference in the number of operations required by the analytical computation and the neural network approximation of the indices, a set of running time tests were performed to further validate the method. Since an analytical expression for the SVD does not exist, the two selected indices were calculated based on the analytical expression of the Jacobian and a pair of well-known algorithms for the calculation of the determinant and the singular values. The determinant of the Jacobian is calculated using LU decomposition in Matlab as the product of the diagonal elements of the upper triangular matrix of $[JJ^T]$. The method is based on the Gaussian elimination method that is among the fastest proposed so far [30]. In order to perform the singular value decomposition for the calculation of KCI, Maltlab's built in function dgesvd is used that is based on Lapack routines. The method is considered among the fastest and most robust algorithms [31] known.

The run-time testing procedure was designed to provide the mean running time t_{mr} of a hundred runs for each method and index. A test set of 1000 inputs was used for testing. The runs were

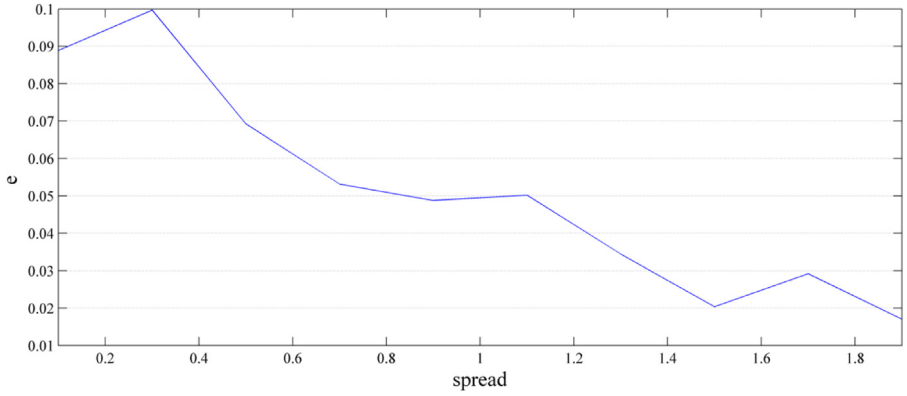

Fig. 8. Design process RBFN-manipulability.

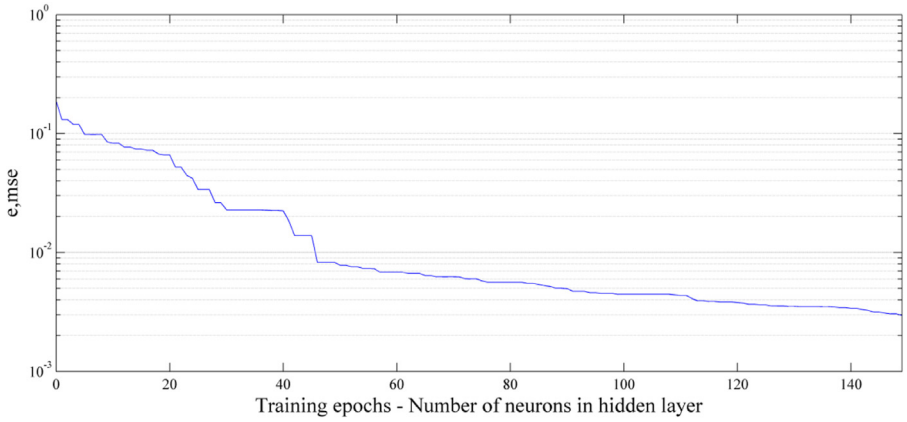

Fig. 9. Training process RBFN-manipulability.

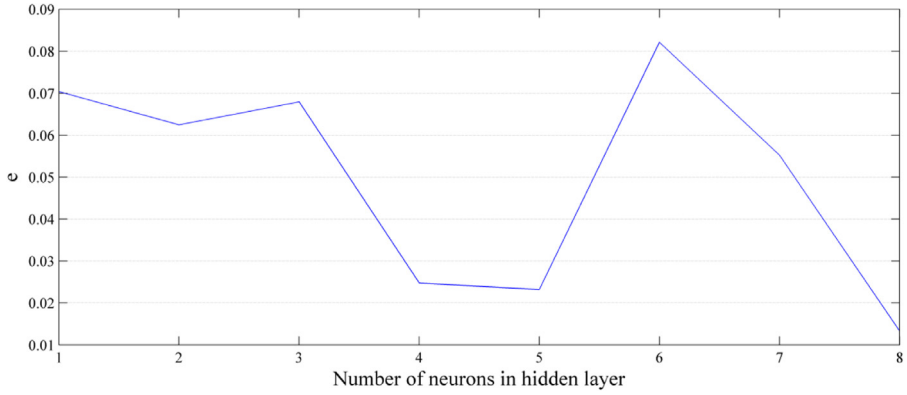

Fig. 10. Design process FFBN-manipulability.

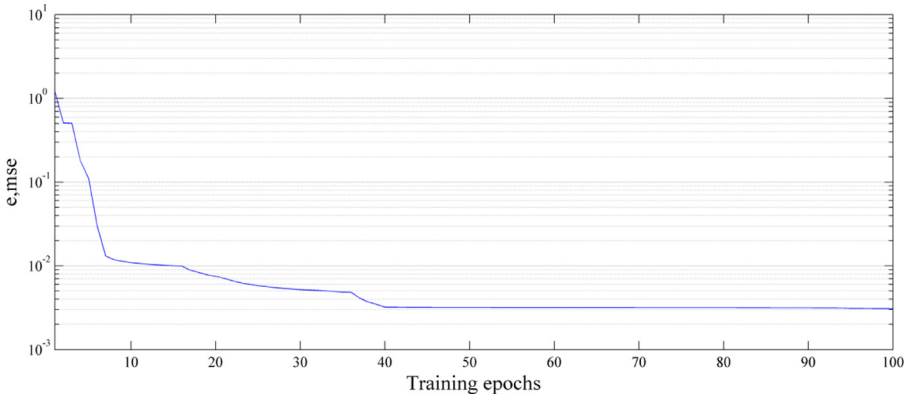

Fig. 11. Training process FFBN-manipulability.

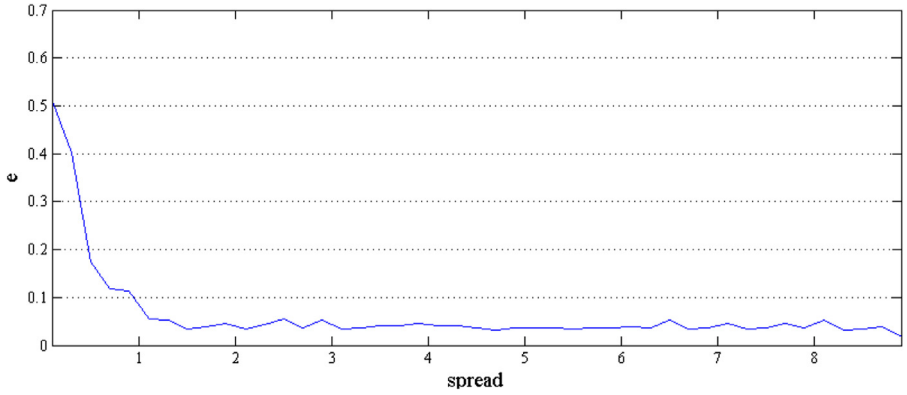

Fig. 12. Design process RBFN-KCI.

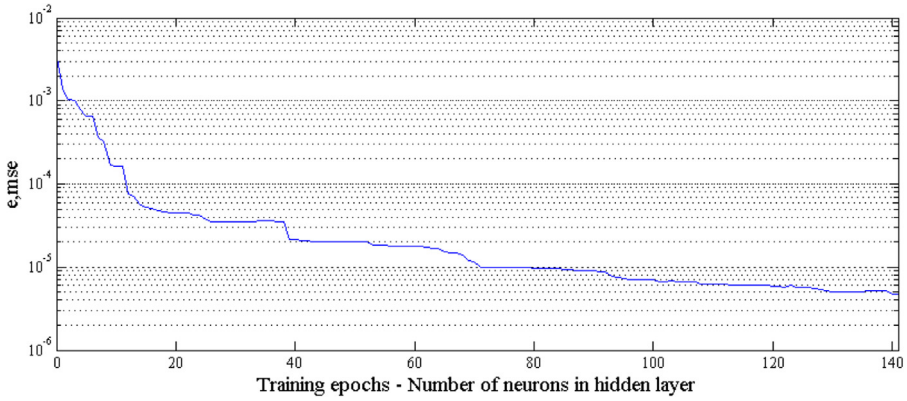

Fig. 13. Training process RBFN-KCI.

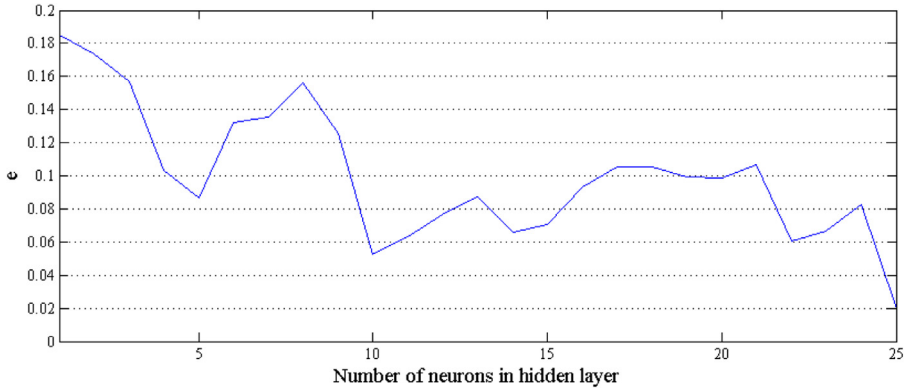

Fig. 14. Design process FFBN-KCI.

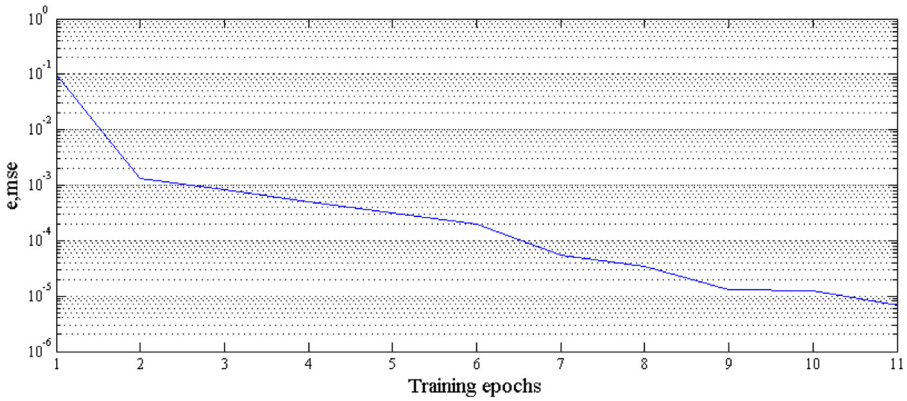

Fig. 15. Training process FFBN-KCI.

Table 5
Comparison of the proposed methods' mean running times.

	Computation based on the analytical Jacobian t_{mr} (s)	*Feed-forward Back-Propagation Network* t_{mr} (s)	*Radial Basis Function Network* t_{mr} (s)
w	0.7043	0.0057	0.0152
KCI	0.7283	0.0069	0.0172

performed on an Intel Core i7-3930K @ 3.8 Ghz processor, with 16 Gb of RAM running 64-bit Windows 7 Professional operating system.

Following the run-time tests it becomes obvious that both neural network approaches are faster than the analytical one, by a factor of a hundred the FFBN and by a factor of ten the RBFN. Moreover, the comparison among RBFN and FFBN shows that FFBN is the fastest, a fact that was expected since RBFN tend to require more neurons that the FFBN for the same problem parameters. However a post-training pruning phase might benefit the RBFN in order to reduce the neurons in the hidden layer and subsequently their computational time.

It should be noted that the running time results are in accordance with the numerical operations required by each method (Table 4). The minor inconsistencies observed are mainly due to the initialization of the networks and their variables. Regarding KCI analytical computation the method is iterative and therefore the convergence plays an important role on its exact computational time.

Both network architectures can provide a good approximation of the dexterity indices given a UVMS configuration, within 2% error. The two selected dexterity indices are used as an indicator of whether a configuration is near singularity or isotropy, therefore, high accuracy is not required and an error of 2% is considered acceptable.

Concluding, the apparent choice for a method to rapidly approximate a dexterity index, such as the ones discussed in this paper, would be a FFBN. Considering the high importance of computational time in real world operations where several hundreds of configurations have to be evaluated instantly, a speedup of 10–100 for this specific process becomes rather significant. On a typical motion planning algorithm this could lead to savings of more than a second per iteration and thus bringing it closer to a real time performance. These time savings could allow a motion planning algorithm to utilise high dexterity configurations and produce efficient plans for UVMS during task execution.

5. Conclusions

In this paper common underwater tasks are associated with well-known dexterity indices and two types of neural network are designed to rapidly approximate the dexterity index given a UVMS configuration. In the design process and during the testing phase it can be observed that both FFBN and RBFN can approximate the dexterity indices within an error of no more than 2%. The dexterity is not considered as an absolute criterion on motion planning but rather as an indicator of appropriate dexterous configurations depending on the task, therefore, such a small error is acceptable for an underwater mission motion planning algorithm.

The number of algebraic operations required for the analytical calculation of the selected dexterity indices was compared with number of operation required by the NN approximation. The comparison proved that the NN derived by the design process require considerably less computations than the analytical calculation. During the run-time testing procedure, the difference of the mean running time t_{mr}, among the computational solution based on the exact form of the Jacobian and both neural network types is proven to be significant. Moreover the FFBN is proved even faster than the RBFN in every case mainly due to the less neurons utilised in the hidden layer. Even though a post-training pruning phase could benefit the RBFN by reducing the number of neurons in their hidden layer, the difference in their time performance remains quite significant. Based on the t_{mr} performance the FFBN prove to be the best option among the networks proposed. The computational time reduction (10–100 times faster) yields better performance for a real-time motion planning algorithm for a UVMS that is to perform an intervention and dexterity is taken under consideration.

The proposed method is applicable on any type of UVMS and could be extended in other dexterity indices too, creating a library of index approaching neural networks. The potential design of neural networks in hardware form could lead to even further computational time savings. The method could easily be extended into space robotic systems by eliminating the passivity of pitch and roll angles or even mobile systems by constraining the respective dof.

In our future plans the integration of the FFBN method in an online motion planner for a UVMS is considered, in order to enhance its performance towards real-time navigation. The development of a series of NN to approach other known dexterity indices and their association with additional underwater tasks is also considered.

Appendix A.

The calculation of the numerical operations required to compute the Jacobian matrix J and the two dexterity indices, was based on the analytical expression of J retrieved by the Symbolic toolbox of Matlab. Since several different operations are required, some assumptions had to be made to calculate the total number of operations. Therefore in this work, we assume that additions and multiplications require the same computational time. In fact the difference between the two kinds of operations depends on the accuracy achieved and the hardware available. The sine and cosine functions are calculated taken into account the Matlab formula and they could be approached by 3 multiplications/divisions and 1 addition/subtraction. The sine and cosine functions are given by:

$$sin\,(x) = \frac{e^{ix} - e^{-ix}}{2i} \tag{22}$$

$$cos\,(x) = \frac{e^{ix} + e^{-ix}}{2} \tag{23}$$

The exponential of a number is considered to cost the same as a multiplication operation.

The analytical expression of the Jacobian is given in Appendix B, whereas the expression for the determinant of the $[JJ^T]$ matrix is given in Appendix B. From the analytical expressions presented and given the assumptions made beforehand it comes up that a total of 21162 operations are required to calculate the Jacobian matrix, from which 5889 addition operations and 15273 multiplication operations.

To calculate the manipulability measure w the determinant of the matrix $[JJ^T]$, where J^T denotes the transpose of J, has to be computed. To multiply the two matrices 432 multiplications and 396 additions are required. The calculation of the determinant of the 6×6 $[JJ^T]$ matrix requires 3600 multiplications and 719 additions. Therefore, the total operations required to analytically calculate the manipulability index, w, are 19305 multiplications and 7005 additions, 26310 operations in total considering the operation required to compute the Jacobian.

Regarding the calculation of the kinematic conditioning index KCI, an exact analytical solution does not exist since singular value decomposition is performed on the Jacobian using iterative algorithms. According to the literature [31], the fastest algorithms proposed so far, for a matrix mxn, where $m < n$, requires approximately m^2n operations to perform singular value decomposition. For a 6×12 Jacobian matrix, SVD results in 432 operations taking into account that this is an approximate number of operations since this is an iterative procedure. Therefore the approximated number of operations required for the calculation of the KCI would be 21594.

Appendix B. Supplementary data

Supplementary data associated with this article can be found, in the online version, at http://dx.doi.org/10.1016/j.asoc.2016.08.033.

References

[1] T. Yoshikawa, Foundations of Robotics, The MIT Press, Cambridge Massachusetts, 1990.
[2] J.K. Salisbury, J.J. Craig, Articulated hands, Int. J. Robot. Res. 1 (1) (1982) 4–17, http://dx.doi.org/10.1177/027836498200100102.
[3] J. Angeles, C.S. Lopez-Cajun, Kinematic isotropy and the conditioning index of serial robotic manipulators, Int. J. Rob. Res. 11 (6) (1992) 0–571.
[4] P. Cardou, S. Bouchard, C. Gosselin, Kinematic-sensitivity indices for dimensionally nonhomogeneous jacobian matrices, IEEE Trans. Rob. 26 (1) (2010) 166–173.
[5] R. Dubey, J.Y.S. Luh, Redundant robot control using task based performance measures, J. Robot. Syst. 5 (5) (1988) 409–432, http://dx.doi.org/10.1002/rob.4620050502.
[6] K. Tchon, K. Zadarnowska, Kinematic dexterity of mobile manipulators: an endogenous configuration space approach, Robotica 21 (5) (2003) 521–530, http://dx.doi.org/10.1017/s0263574703005022.
[7] B. Bayle, J.-Y. Fourquet, M. Renaud, Manipulability of wheeled mobile manipulators: application to motion generation, Int. J. Robot. Res. 22 (7-8) (2003) 565–581, http://dx.doi.org/10.1177/02783649030227007.
[8] T. Padir, J.D. Nolff, Manipulability and maneuverability ellipsoids for two cooperating underwater vehicles with on-board manipulators, in: Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference, on 7–10 Oct. 2007, 2007, pp. 3656–3661.
[9] B.H. Jun, Pan-Mook Lee, J. Lee, Manipulability analysis of underwater robotic arms on ROV and application to task-oriented joint configuration, in: OCEANS '04. MTTS/IEEE TECHNO-OCEAN, vol. 1543, 09–12 Nov. 2004, 2004, pp. 1548–1553.
[10] T. Asokan, G. Seet, M. Lau, E. Low, Optimum positioning of an underwater intervention robot to maximise workspace manipulability, Mechatronics 15 (6) (2005) 747–766, http://dx.doi.org/10.1016/j.mechatronics.2004.12.003.
[11] P. Sotiropoulos, N.A. Aspragathos, F. Andritsos, Optimum docking of an unmanned underwater vehicle for high dexterity manipulation, Int. J. Comput. Sci. 38 (1) (2011) 8–56.
[12] A.W. Quinn, D.M. Lane, Computational issues in motion planning for autonomous underwater vehicles with manipulators, in: Autonomous Underwater Vehicle Technology, 1994. AUV '94., Proceedings of the 1994 Symposium, 19–20 Jul. 1994, 1994, pp. 255–262.
[13] L. Fausett, Fundamentals of Neural Networks: Architectures, algorithms, and Applications, Prentice-Hall, Inc., 1994, 2016.
[14] A.T. Hasan, N. Ismail, A.M.S. Hamouda, I. Aris, M.H. Marhaban, H.M.A.A. Al-Assadi, Artificial neural network-based kinematics Jacobian solution for serial manipulator passing through singular configurations, Adv. Eng. Software 41 (2) (2010) 9–367.
[15] Y. Kuroe, Y. Nakai, T. Mori, A new neural network learning of inverse kinematics of robot manipulator, vol. 5, 27 Jun–2 Jul 1994, in: Neural Networks, 1994. IEEE World Congress on Computational Intelligence, 1994 IEEE International Conference, 5, 2016, pp. 2819–2824, http://dx.doi.org/10.1109/ICNN.1994.374678.
[16] R. Köker, C. Öz, T. Çakar, H. Ekiz, A study of neural network based inverse kinematics solution for a three-joint robot, Rob. Auton. Syst. 49 (3–4) (2004) 7–234.
[17] M. Dehghani, M. Ahmadi, A. Khayatian, M. Eghtesad, M. Farid, Neural network solution for forward kinematics problem of HEXA parallel robot, in: American Control Conference, 11–13 June 2008, 2008, pp. 4214–4219.
[18] P. Sotiropoulos, N.A. Aspragathos, F. Geffard, High Dexterity Docking of an UUV by Fast Determination of the Area Manipulability Measure of the Arm Using ANN 2016 Paper presented at the IFAC SYROCO.
[19] B. Siciliano, O. Khatib, Handbook of Robotics, Springer, 2008.
[20] WHOI, Jason ROV Specifications. http://www.whoi.edu/page.do?pid=10755.
[21] R. Murray, Z. Li, S. Sastry, A Mathematical Introduction to Robotic Manipulation, CRC, 1994.
[22] N. Vukovic, Z. Miljkovic, A Growing and Pruning Sequential Learning Algorithm of Hyper Basis Function Neural Network for Function Approximation, Neural Networks, vol. 46C, Elsevier, 2013, pp. 210–226, October.
[23 G.B. Huang, P. Saratchandran, N. Sundararajan, A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation, IEEE Trans. Neural Netw. 16 (1) (2005) 57–67.
[24] N. Vukovic, Z. Miljkovic, Robust sequential learning of feedforward neural networks in the presence of heavy-tailed noise, Neural Netw. 63 (2015) 31–47, http://dx.doi.org/10.1016/j.neunet.2014.11.001, Elsevier.
[25] S. Haykin, Neural Networks: a Comprehensive Foundation, 2nd ed., Prentice Hall PTR, Upper Saddle River, NJ, USA, 1998.
[26] K. Gotlih, I. Troch, Base invariance of the manipulability index, Robotica 22 (4) (2004) 455–462.
[27] A. Synodinos, N.A. Aspragathos, Frame Invariance of the dynamic manipulability measure, MULTIBODY DYNAMICS 2011, ECOMAS Thematic Conference (2011).
[28] Matlab.www.mathworks.com.
[29] T. Hagan, M. Menhaj, Training feedforward networks with the marquardt algorithm, IEEE Trans. Neural Netw. 5 (6) (1994) 989–993.
[30] B. Dingle, Calculating Determinants of Symbolic and Numeric Matrices a Starting Point, Texas A&M University, 2005, 2016.
[31] G. Plassman, A Survey of Singular Value Decomposition Methods and Performance Comparison of Some Available Serial Codes, NASA Center for AeroSpace Information (CASI), 2005.