Neurocomputing 166 (2015) 229–243

Contents lists available at ScienceDirect

Neurocomputing

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

Using evolutionary neural networks to predict spatial orientation of a ship

Tomasz Praczyk

Polish Naval Academy, Institute of Naval Weapon, Gdynia, Poland

ARTICLE INFO

Article history:
Received 10 November 2014
Received in revised form
18 January 2015
Accepted 22 March 2015
Communicated by Manoj Kumar Tiwari
Available online 17 April 2015

Keywords:
Evolutionary neural networks
Prediction
Ship spatial orientation

ABSTRACT

The ability to precisely predict behavior of a ship can be useful for different ship systems, for example, dynamic positioning, video or artillery systems. To automatically maintain a ship position and heading, the dynamic positioning system has to know future behavior of the ship as exactly as possible. The same applies to the ship video and artillery systems which to continuously track a target have to predict both the location of the target and orientation of the ship deck in relation to the target in successive points in time.

In this paper, evolutionary neural networks are proposed as ship behavior predictors. To perform the task, they are supplied with the information about ship spatial orientation (Euler angles) acquired from inertial navigational systems. In experiments reported in the paper, both monolithic and modular recurrent neural networks were tested. To build them, a neuro-evolutionary method called Assembler Encoding with Evolvable Operations was applied. As the point of reference for the networks two other prediction methods were used: the first is Linear Regression with Correction, i.e. the most effective method in preliminary experiments, whereas the second is Autoregressive Integrated Moving Average, which seems to be currently the most general tool for forecasting a time series.

1. Introduction

Ship motion in waves has to be taken into consideration by most ship systems. In some cases, knowledge about the current state of the ship is insufficient to effectively perform a task, sometimes, the systems need the information of how the ship may behave in the future. For example, during underwater works, when a ship supports divers and is rigidly connected with some underwater infrastructure, it is necessary to precisely maintain ship position and heading. To this end, the dynamic positioning systems are used which control ship propellers and thrusters based on two types of information, i.e. the information about the current and historical states of the ship acquired from different sensors and the information concerning predicted states provided by a model of the ship. Estimations of the future states make the dynamic positioning systems more effective allowing them to respond not only to the current dangers but also to the ones which may occur in the following points in time.

In order to effectively track an object, ship vision and artillery systems are stabilized. To this end, the information about ship angular velocities acquired from gyroscopes is used. However, sometimes, momentary state of a ship measured by the gyroscopes is insufficient to precisely control the ship camera/gun. To improve accuracy of tracking, a ship model can be used whose task is to predict future behavior of the ship.

In order for the ship model to be useful for the ship vision/artillery systems, all model predictions have to be made very fast and they have to be very accurate. Dynamics of both the tracked object and the ship in longer periods is very difficult to predict so the only solution is to predict for short periods. To this end, all calculations associated with prediction have to be made very fast. High speed of the prediction is also significant because of speed requirements for the camera/gun which have to be able to act with a high frequency.

As for accuracy, it is also a key parameter of the prediction. This is because even an error of aiming the camera/gun equal to 0.1° means the error equal to a number of meters near a distant object.

A serious problem is also that predictions have to provide information about ship states which are very distant from the point of the last measurement. To track an object, the tracking system has to make measurements, collect information about the current and historical states of the object and the ship, use this information to predict a ship next state and to determine an appropriate setting for the camera/gun, use a hydraulic/electric drive to move the camera/gun and finally use it. The result is that the time interval between camera/gun usage and a prediction on the point in time of the camera/gun usage is long with respect to the ship dynamics which hinders accurate prediction.

The other problem is that evaluation of each prediction based on ship state measurements is delayed. Before a ship state

http://dx.doi.org/10.1016/j.neucom.2015.03.075
0925-2312/

corresponding to a prediction can be measured, a number of other predictions are performed for future camera/gun applications, which means that the feedback information verifying accuracy of prediction can be obtained after a number of other predictions, not immediately after the prediction.

Currently, the ships are usually modeled as rigid bodies whose behavior is described in the form of differential equations ([13] and many other papers in the field of mechanics). In order for the ship model to be complete, a wave model is also necessary. Both models require dimensions of a modeled ship, weight, parameters of propulsion, parameters of wave and others. Some parameters are approximate characteristics of ships and waves which are difficult or impossible to measure. Moreover, different speeds of the ship often require its different models while the various sea areas require different wave models. The advantage of this approach is that it does not need measurements on a real object to construct approximate models, for example, for use on simulators, estimated parameters of the modeled object are sufficient. However, when the purpose is to build highly accurate model, all the parameters of the object and wave have to be precisely set which requires multiple measurements at sea and tedious tuning process.

Another simple approach to model the ship behavior is to apply the regression methods for that purpose. The modeled object together with the sea environment is treated as a black-box, there are no parameters which relate directly to the former and the latter, parameters which as mentioned above are sometimes difficult, problematic or even impossible to determine accurately. In this case, subsequent measurements of the ship spatial orientation are used to estimate a regression function which defines an approximate behavior of the ship in a short period of time. The function adjusted to the measurements is the ship model, the parameters of the function relate not to the object but to its measured behavior.

The first step in the use of this approach has been taken within the framework of the research reported in [19]. All the research were carried out under the assumption that each motion of a ship which affects its spatial orientation can be described with a sinus function – the change of roll, and pitch of a ship moving under the influence of sea waves is a periodic change with some maximum and minimum which means that it can be approximated by means of sinusoid. During simple experiments, two classic regression methods, i.e. linear and nonlinear (sinusoid) regression, and one method designed especially for the purpose of predicting ship behavior, i.e. linear regression with correction (LRC), were tested. The task of each of them was to predict the roll angle of a ship which changed according to seven model sinusoid functions with different parameters and higher harmonic components. The results of the tests appeared to be promising, they, generally, showed that accurate prediction of the ship behavior by means of regression methods is possible in short periods of time.

Time series prediction and tools used in this field are a next solution which can be used to model behavior of the ship. To model, analyze and predict time series, linear statistical methods, i.e. autoregressive (AR), moving average (MA), and autoregressive moving average (ARMA), are often employed. Roughly speaking, they are based on the assumption that each new signal is a noisy linear combination of the last few signals and independent noise terms. These methods have been successfully used in a wide range of applications in different areas (e.g. [1–3,7,21]), however, there are problems, especially real-life problems (for example the problem addressed in the paper) which are nonlinear, and in consequence, need other than linear predictors. In this case, artificial neural networks (ANNs) seem to be a valuable alternative to the statistical methods. To date, many attempts have been made to apply ANNs to time series problem. For example, in [4,9], authors propose Fuzzy Neural Networks (FNNs) as time series predictors, Multi-layered Perceptron (MLP) and Radial Basis Function neural networks (RBFs) are suggested among other things in [5,8], a form of Kohonen network is applied in [11], whereas different types of Recurrent Neural Networks (Elman, NARX) are proposed in [10,12]. Although to train neural time series predictors, classical gradient training algorithms are usually used, attempts are also made to employ global optimization methods like Particle Swarm Optimizer (PSO), Genetic Algorithms (GA) and Evolutionary Strategies (ES) [20] for that purpose.

Successes of ANNs in predicting nonlinear time series shown in all the papers mentioned above contributed to a decision to use them also to model ship behavior. For that purpose, Recurrent Evolutionary Neural Networks (RENNs) build by means of neuro-evolutionary method called Assembler Encoding with Evolvable Operations (AEEO) [18] were applied. The RENNs differ from the Elman and NARX recurrent networks [10,12] in three points: (i) they evolve according to AEEO, whereas the latter networks are trained with the classical gradient methods, (ii) the architecture of RENNs depends entirely on the evolutionary process, which means, for example, that feedbacks can appear in the networks between any neurons, meanwhile the Elman and NARX networks have a fixed topology-feedbacks connect only selected neurons, (iii) a part of RENNs have a modular architecture, they consist of cooperating sub-networks, whereas, the Elman and NARX networks are all monolithic.

To test what accuracy of predicting ship behavior can be achieved by means of RENNs, experiments based on measurements of real ship behavior were carried out. The measurements were performed with the use of inertial navigational systems (INS) and additionally, for the purpose of improving reliability of measurements, with the use of accurate satellite systems (GNSS RTK). To predict the spatial orientation of the ship, in addition to different RENNs, two other methods were also applied: LRC, i.e. the most effective regression method from the earlier experiments, and Autoregressive Integrated Moving Average (ARIMA[1]-generalization of ARMA), i.e. the most widely used approach to time series forecasting.

The paper is a report on the experiments, it is organized as follows: Section 2 shortly defines the prediction problem, Section 3 presents all the prediction methods, Section 4 shortly describes the neuro-evolutionary method used to evolve the neural networks, Section 5 shows and discusses the results and Section 6 summarizes the paper.

2. Definition of the prediction problem

The prediction problem, as assumed in the paper, relates to a situation when it is necessary to estimate a future state of a process, say y, based on state measurements $y(t_0), y(t_1), \ldots, y(t_{n-1}), y(t_n)$ (for example, subsequent measurements of roll or pitch) in the current and earlier points in time, n is the number of measurements. In the case of a ship and its spatial orientation, the following extra assumptions are also made: (i) the interval between individual state measurements is the same and equal to Δt_m, (ii) the prediction is made based on only the two last measurements $y(t_{n-1}), y(t_n)$, (iii) the interval between the last measurement made in t_n and the point in time t_p for which the prediction is made is a number of times greater than Δt_m and is equal to Δt_p (horizon of prediction).

To measure the ship orientation, INSs are assumed to be used. They can provide the information about the ship orientation with a very high frequency and at regular intervals which means that we can assume the same and very short interval Δt_m for every two measurements. The experiments reported in [19] showed,

[1] In the experiments MATLAB implementation of ARIMA was used.

moreover, that the most accurate prediction on sinusoid-shape functions which are assumed to be the most suited model of ship rotational movements and the progressive movement along the axis leading from the ship to the Earth center is made when only two up-to-date measurements are used in the calculations. With regard to the interval Δt_p and its value greater than Δt_m, it is a consequence of the duration of all calculations and actions necessary to move the camera/gun at a proper position. First, it is necessary to determine a current ship orientation (fusion of data from INS angular rate sensors, accelerometers and magnetometers), next, to estimate a future ship orientation and an observed object/target position (prediction), then, to determine azimuth and elevation for the camera/gun and set the device according to the calculated angle parameters (executive systems have to move the camera/gun to a proper position), and finally, to use the camera/gun. All these calculations and actions require more time than the INSs need for simple readings of their sensors.

3. Methods used to predict ship spatial orientation

In the experiments reported in the paper, seven different methods were used to predict the ship spatial orientation: linear regression with corrections (LRC), Autoregressive Integrated Moving Average (ARIMA), two variants of recurrent evolutionary neural network (RENN(1–2)), and three variants of modular recurrent evolutionary neural network (MRENN(1–3)). All the seven methods are detailed further.

3.1. Linear regression with correction

In the LRC, the predicted ship state $\hat{y}(t_p)$ in point in time t_p is calculated in two stages. In the first stage, parameters a and b of a line which according to the least squares method fit the best to successive measurements $(t_i, y(t_i))$ are fixed. The parameters are calculated by formulas (1)–(3) or (4)[2] and they are used in the first stage to determine the predicted value $\tilde{y}(t_p)$ according to the classic linear regression:

$$\tilde{y}(t_p) = at_p + b \tag{1}$$

$$a = \frac{n\sum_i^n t_i y(t_i) - \sum_i^n t_i \sum_i^n y(t_i)}{n\sum_i^n t_i^2 - \left(\sum_i^n t_i\right)^2} \tag{2}$$

$$b = \frac{\sum_i^n y(t_i) - a\sum_i^n t_i}{n} \tag{3}$$

$$\tilde{y}(t_p) = \frac{y(t_n) - y(t_{n-1})}{\Delta t_m}(\Delta t_m + \Delta t_p) + y(t_{n-1}) \tag{4}$$

In the next step, the value $\tilde{y}(t_p)$ is corrected by the value of estimated error $\Delta\tilde{y}(t_p, a, b)$ which is calculated as follows:

$$\Delta\tilde{y}(t_p, a, b) = \frac{\sum_i^K \Delta y(a_i, b_i) G((a_i, b_i), (a, b))}{\sum_i^k G((a_i, b_i), (a, b))} \tag{5}$$

$$G((a_i, b_i), (a, b)) = \exp\left(-\frac{\|(a_i, b_i), (a, b)\|^2}{2\sigma^2}\right) \tag{6}$$

where a_i,b_i are parameters of regression lines calculated during previous predictions, $\Delta y(a_i, b_i)$ is an error measured for parameters a_i,b_i, σ is a parameter, K is the number of pairs (a_i, b_i) calculated in the previous predictions.

To estimate error $\Delta\tilde{y}(t_p, a, b)$, accurate errors $\Delta y(a_i, b_i)$ are necessary. They are measured for different a and b, for many earlier points in time in which the linear prediction is performed. Using historical data to improve indications of the linear predictor means that this solution adapts to conditions at sea and to a place on the sinusoid in which the ship is located. On the sinusoid, there are both nearly linear areas which require smaller corrects and nonlinear ones which need a greater intervention of the correction system. Linearity and nonlinearity of these areas or identification of the place on the sinusoid where the ship is currently located is determined based on values a and b [19].

As already mentioned, each prediction solution dedicated for cooperation with the control system of a ship-camera/gun should be on the one hand very accurate but on the other hand it also should be highly quick. Time-consuming methods which can predict a future state with a very high accuracy are completely useless because the result of their operation is simply produced too late. The linear regression works with a very high speed which is due to the fact that it needs a little number of calculations to produce value $\hat{y}(t_p)$. LRC is different in this regard. In this case, to determine $\hat{y}(t_p)$, the linear regression supported by corrections calculated based on the set of historical data is applied. The accuracy of corrections added to linear predictions depends on whether the set of the historical data contains errors recorded for values a and b close to the current values of a and b. Generally, from the point of view of the accuracy of corrections, the situation with many historical data is better than situation with few data. Too few historical data results in the situation when in many cases, i.e. for many a and b, corrections are equal to zero (see (5) and (6)) and the method is reduced to the simple linear regression. In order for the correction component to have the influence on predictions, it is necessary to make calculations based on a greater number of example errors memorized in the system. The problem is, however, that the historical data cannot be too numerous because the more the data is in the system, the slower are the calculations. In order to ensure a high speed and accuracy of predictions, the method uses a special mechanism whose responsibility is to appropriately organize the set of historical data. First, it is careful not to make the set too numerous, second, it cares to uniformly distribute data in the space axb, and third, it also ensures adaptability of the set, and in consequence, the whole prediction system to changing conditions at sea. The last task can be achieved by continuous data updates, that is, by replacing older data with up-to-date data [19].

Ultimately, after determining $\Delta\tilde{y}(t_p, a, b)$ according to (5) and (6), $\hat{y}(t_p)$ is calculated in the following way [19]:

$$\hat{y}(t_p) = \tilde{y}(t_p) + \Delta\tilde{y}(t_p, a, b) \tag{7}$$

3.2. Autoregressive Integrated Moving Average

ARIMA is currently one of the most widely used models to time series analysis. Since it is well described in the literature and even in the Internet, the paper presents only a short description of the model.

ARIMA is a generalization of ARMA model that consists of two components, i.e. AR and MA model. AR model of order p, i.e. AR(p) is a discrete time linear model of the form:

$$y(t_n) = \alpha_1 y(t_{n-1}) + \cdots + \alpha_p y(t_{n-p}) + \epsilon_n \tag{8}$$

where $\alpha_1,\ldots,\alpha_p$ are the parameters of the model, and ϵ_n is an error term.

In turn, MA model of order q, i.e. MA(q) is defined in the following way:

$$y(t_n) = \epsilon_n + \cdots + \beta_1\epsilon_{n-1} + \cdots + \beta_q\epsilon_{n-q} \tag{9}$$

where $\beta_1,\ldots,\beta_q$ are the parameters of the model.

[2] For $n=2$, we can assume that $t_{n-1}=0$, $t_n=\Delta t_m$ and in consequence $t_p=\Delta t_m+\Delta t_p$.

In consequence, ARMA(p,q) can be defined as follows:

$$y(t_n)=\alpha_1 y(t_{n-1})+\cdots+\alpha_p y(t_{n-p})+\epsilon_n+\cdots+\beta_1\epsilon_{n-1}+\cdots+\beta_q\epsilon_{n-q} \tag{10}$$

or in a shorter form as follows:

$$\left(1-\sum_{k=1}^{p}\alpha_k L^k\right)y(t_n)=\left(1+\sum_{k=1}^{q}\beta_k L_k\right)\epsilon_n \tag{11}$$

where L is the lag operator defined as $Ly(t_n)=y(t_{n-1})$, for all t; the powers, positive and negative, of the lag operator are denoted by $L_k: L^k y(t_n)=y(t_{n-k})$, for all t.

Ultimately, ARIMA(p,d,q) model with integration element I of order d, i.e. I(d), can be presented in the following form:

$$\left(1-\sum_{k=1}^{p}\alpha_k L^k\right)(1-L)^d y(t_n)=\left(1+\sum_{k=1}^{q}\beta_k L_k\right)\epsilon_n \tag{12}$$

where $(1-L)^d$ is a power of d (d is the integer) of difference operator $(1-L)$ of integrated part of ARIMA model; for example $(1-L)^2 y(t_n)=y(t_n)-2y(t_{n-1})+y(t_{n-2})$.

3.3. Recurrent evolutionary neural network

In RENN, prediction of the ship orientation is performed by a recurrent neural network. The network has one or two input neurons (depending on RENN variant), one output neuron and a number of hidden neurons (see Fig. 1). The first variant of RENN, say RENN1, uses the network with a single input neuron supplied with the following signal: $y(t_n)-y(t_{n-1})$, whereas RENN2, that is, the second variant of the method, applies the network with two input neurons, the first of which is supplied with the same signal as in RENN1 whereas the second provides the network information about a point on the sinusoid at which the ship is located, that is, whether it is near the maximum where the angular velocity is low and the change in spatial orientation is small or is somewhere in between maximum states where orientation changes and velocities are considerably greater. This information takes the form of the parameter a of a regression line determined for a number of subsequent state measurements. In RENN, the purpose of the first input neuron is to inform the network about a momentary state of the ship which can be noisy whereas the second input neuron (RENN2) gives the network more reliable information acquired from a larger number of measurements.

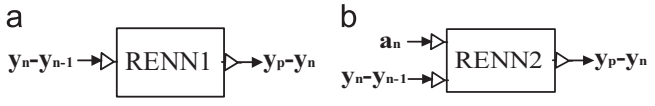

Fig. 1. (a) RENN1 recurrent neural network (b) RENN2 recurrent neural network (feedbacks are hidden inside the boxes).

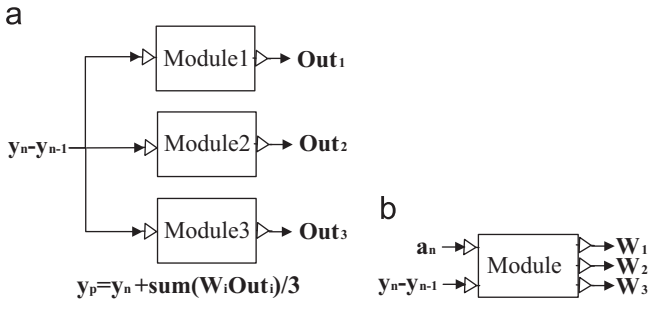

Fig. 2. (a) MRENN1 modular network. (b) MRENN1d feed-forward module.

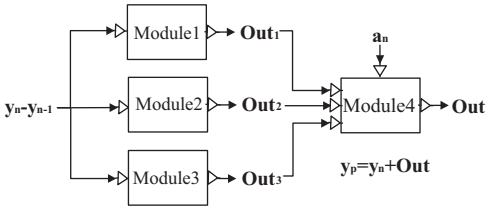

Fig. 3. MRENN2 modular network.

A single output neuron of the network produces the following signal: $\hat{y}(t_p)-y(t_n)$, which means that to obtain the predicted value $\hat{y}(t_p)$ the output signal has to be increased by value of the last measurement $y(t_n)$.

The inner architecture of the network, that is, the number of hidden neurons, their weights and network connectivity depends on a designer and a learning method. The designer decides about cardinality of neurons whereas the learning method about their weights and interconnections. To train all neural networks used in the experiments, also the ones presented further, a neuro-evolutionary method called Assembler Encoding with Evolvable Operations [18] was applied.

3.4. Modular recurrent evolutionary neural network

In all variants of MRENN, prediction of the ship spatial orientation is performed by modular neural networks, each of which consists of separate RENN modules. Moreover, in MRENN1 and MRENN2, that is, in the first and second variant of MRENN, there is also an extra feed-forward module network which is responsible for fusion signals from the recurrent modules. In all variants of MRENN, each single module included in the modular network is trained separately. Input and output layers of the module networks are adjusted to their tasks whereas the inner architecture (weights and connectivity) is always a result of the training process. As in RENN, the output signal of all MRENN modular networks is in the following form: $\hat{y}(t_p)-y(t_n)$

MRENN1 is, in general, a weighted average of output signals from individual modules (see Fig. 2a):

$$Out^{MRENN1}=1/N\sum_{i}^{N}W_i Out_i^m. \tag{13}$$

where Out^{MRENN1} is an output signal of the MRENN1 modular network, Out_i^m is an output signal of ith RENN module, N is the number of modules. The weights are determined in a different way:

- MRENN1a: all weights are equal to 1 which means that this variant is an arithmetic average of module outputs $W_i=1$

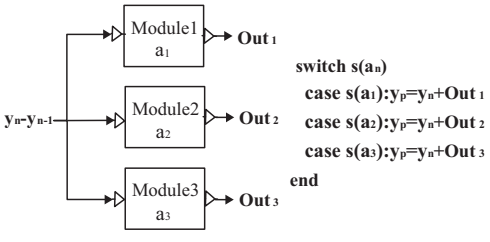

Fig. 4. MRENN3 modular network (a_1, a_2, a_3 are centers of clusters assigned to individual RENN modules).

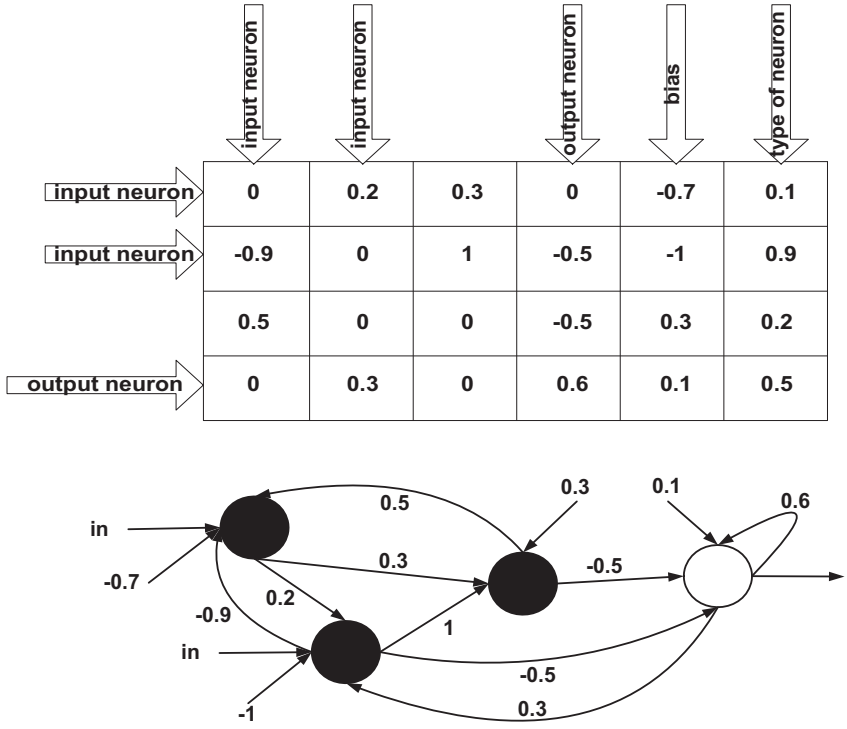

Fig. 5. The way of encoding ANN in the form of Network Definition Matrix (NDM) [17].

Parameters:
T - threshold
P_{max}^{AEEO} - maximum number of neurons in resulting ANN
Input:
set of ANN-operations
$L_{ANN-oper}$ - number of ANN-operations
Output:
NDM - network NDM

```
InitiateNDM(0)//all items in NDM are set to 0
FOR i = 1 , i ≤ P_max^AEEO
    FOR j = 1 , j ≤ P_max^AEEO + 1
        FOR l = 1 , l ≤ L_ANN-oper
            ns_l = FANN_1^l(i,j) //negotiation strength
            //FANN_k^l(i,j) - kth output of lth ANN-operation
            //i and j are inputs to network
        END
        win = Index(ns) //index of winner ANN-operation
        IF FANN_2^win(i,j) > T
            NDM_i,j = FANN_3^win(i,j)
        END
    END
END
```

Fig. 6. Using ANN-operations to form NDM in AEEO [18].

- MRENN1b: the weights depend on module results (fitness) achieved during the learning process,

$$W_i = \frac{F_i}{\sum_k^N F_k} \tag{14}$$

where F_i is the fitness of ith module,

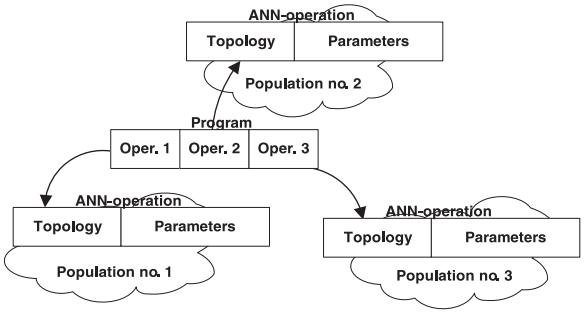

Fig. 7. Evolution of programs in AEEO [18].

- MRENN1c: the weights depend on module results achieved during operation,

$$W_i^{(1)}(t_n) = \frac{1 - e_i(a(t_n))}{\sum_k^N 1 - e_k(a(t_n))} \tag{15}$$

or

$$W_i^{(2)}(t_n) = \begin{cases} W_i^{(1)}(t_n) & \text{if } W_i^{(1)}(t_n) > T \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

$$e_i(a) = 1/P \sum_{k=1}^{P} h(m_i(t_{n-k}), a(t_{n-k}), a) \tag{17}$$

$$h(m, a1, a2) = \begin{cases} m & \text{if } s(a1) = s(a2) \\ 0 & \text{otherwise} \end{cases} \tag{18}$$

where $a(t)$ is a parameter of a regression line calculated in point t, $m_i(t)$ is a prediction error of ith module measured in point in

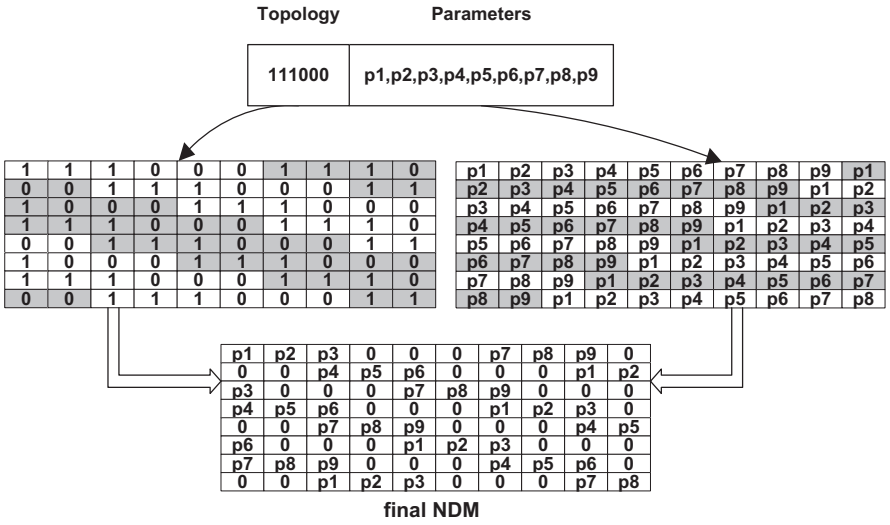

Fig. 8. The way of building NDM representation of ANN-operation [18].

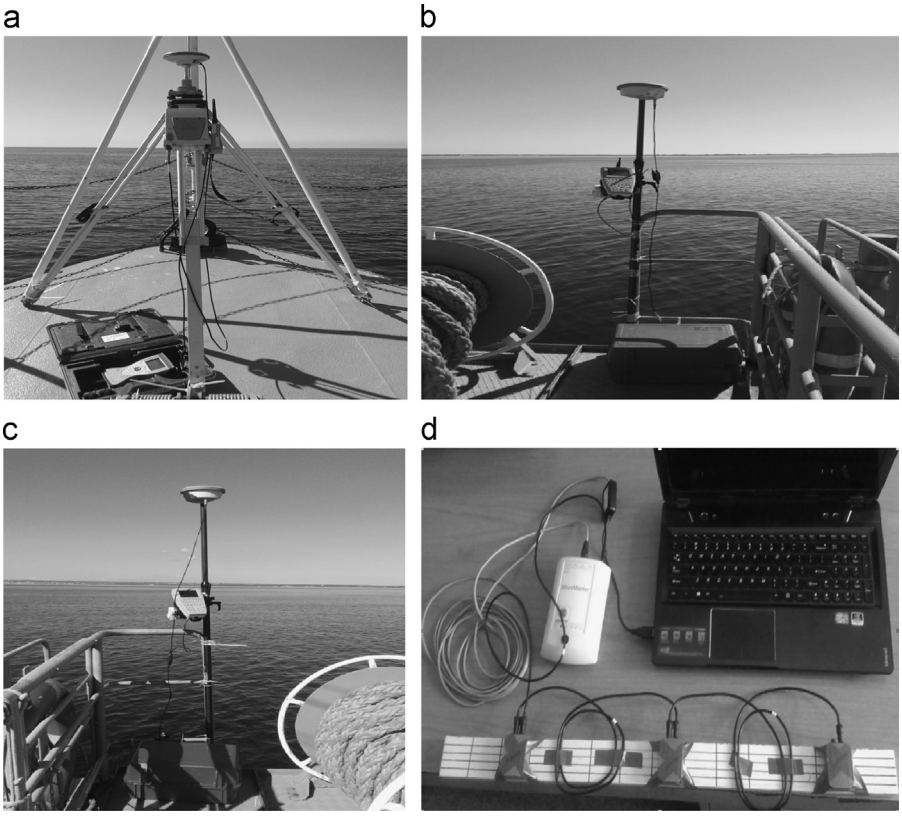

Fig. 9. GNSS RTK satellite system located on the ship (a) bow, (b) port side, (c) starboard side, (d) three INS MEMS (Micro-Electro-Mechanical Systems) located inside the ship (each INS worked for different settings of Kalman filter, all INS measurements were averaged and then corrected based on data from GNSS).

time t, $m \in \langle 0, 1\rangle$, $s(a)$ is the number of a cluster assigned to a, the whole a-parameter space is divided into clusters, $s(a)=1,..,C$, where C is the number of clusters, P is the number of predictions which are taken into account when calculating weights, T is a threshold.

- MRENN1d (see Fig. 2b): the weights depend on output signal of additional feed-forward neural network $W_i = Out_i$, Out_i is the ith output of the neural network which has N outputs and two inputs: $In_1 = y(t_n) - y(t_{n-1})$, $In_2 = a(t_n)$.

In MRENN2, each modular network consists of two layers, the first of which, as in all variants of MRENN, is composed of separate RENN modules, whereas the second contains a single feed-forward output network with $N+1$ inputs: $In_1 = a(t_n)$, $In_i = Out_{i-1}$, $i = 2, .., N+1$, and one output which is output of the entire modular network (see Fig. 3).

In turn, MRENN3 modular networks (see Fig. 4) include only the RENN modules, each of which, unlike the RENN modules from the remaining variants of MRENN, is responsible for a separate task, that is, for prediction in different conditions identified by means of a-parameter of the regression line. To determine output of a MRENN3 modular network, first, the a-parameter is calculated which characterizes the current ship state, next, a single RENN module is selected corresponding to the a-parameter, and finally,

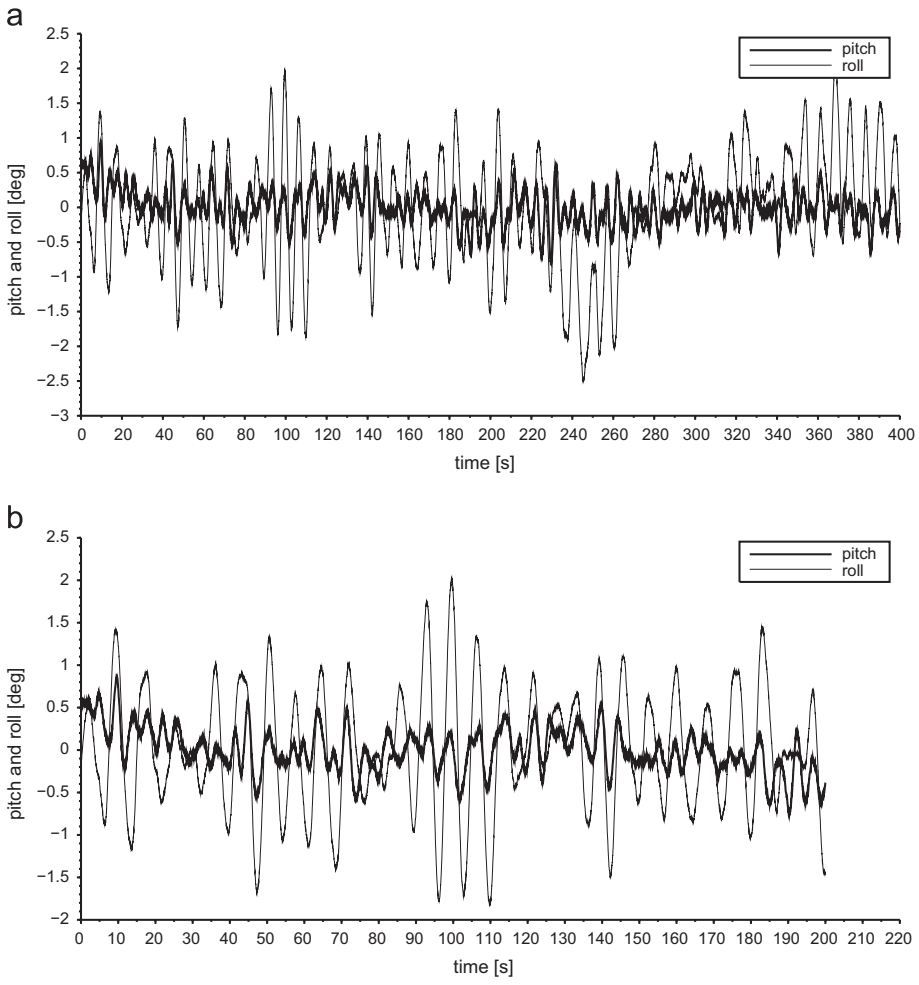

Fig. 10. Learning part of recordings (a) *learn1*, (b) *learn2*.

the output signal of the selected module is set as the output of the entire modular network. In order for RENN modules to be able to perform separate tasks, they are trained on separate training sets, each of which corresponds to a different *a*-parameter.

4. The method used to train neural networks – Assembler Encoding with Evolvable Operations

Assembler Encoding with Evolvable Operations (AEEO) originates from a generative neuro-evolutionary method called Assembler Encoding (AE) [16,17]. In AE, an artificial neural network (ANN) is represented in the form of a linear program consisting of operations with predefined implementations and data. The task of the program is to create a Network Definition Matrix (NDM) defining a single ANN. To form the program and, in consequence an ANN, Cooperative Co-Evolutionary Algorithm (CCEGA) [14,15] is used. The genetic algorithm generates operations (parameters of operations; as mentioned above, implementations are defined beforehand) and data which combined together form a program. Each program builds NDM which is then transformed into an ANN. To this end, the matrix has to store all the information necessary to construct a network. This information is included both in the size and individual items of the matrix, scaled always to the range $\langle -1, 1\rangle$. The size of NDM determines a maximum number of neurons in an ANN whereas individual items of the matrix define weights of interneuron connections, i.e. an item NDM $[i,j]$ determines a link from neuron i to neuron j. Apart from the basic part, NDM also contains additional columns that describe parameters of neurons, e.g. type of neuron (e.g. sigmoid, radial, linear), and bias (see Fig. 5) [17].

In AEEO, the operations with predefined implementations are replaced with ANNs-operations with the same task as in the classic variant of the method. Moreover, the data used previously by the operations are completely removed from the programs which in AEEO include exclusively the ANNs-operations. The ANNs-operations operate together on one NDM which encodes one final ANN, once the matrix is completely formed it is then transformed into the network. To perform the task, ANNs-operations have two inputs, a number of hidden neurons, and three outputs. The inputs indicate an item in NDM updated by the operation (a number of row and column) whereas the outputs are used for three different purposes, i.e. to determine a negotiation strength of each ANN-operation, to determine whether the item should by modified or should not, and to determine a new value for the item. To indicate which ANN-operation should determine the value of a given item, negotiation outputs of all ANNs-operations are compared. An ANN-operation with the greatest output value is entitled to modify the item. In order for the item to be updated, the second output of the selected ANN-operation is tested. If the output value is greater than an assumed threshold the item gets a value from the third output of the ANN-operation (see Fig. 6). Otherwise, the item remains intact and the corresponding connection between neurons in the final ANN is not established [18].

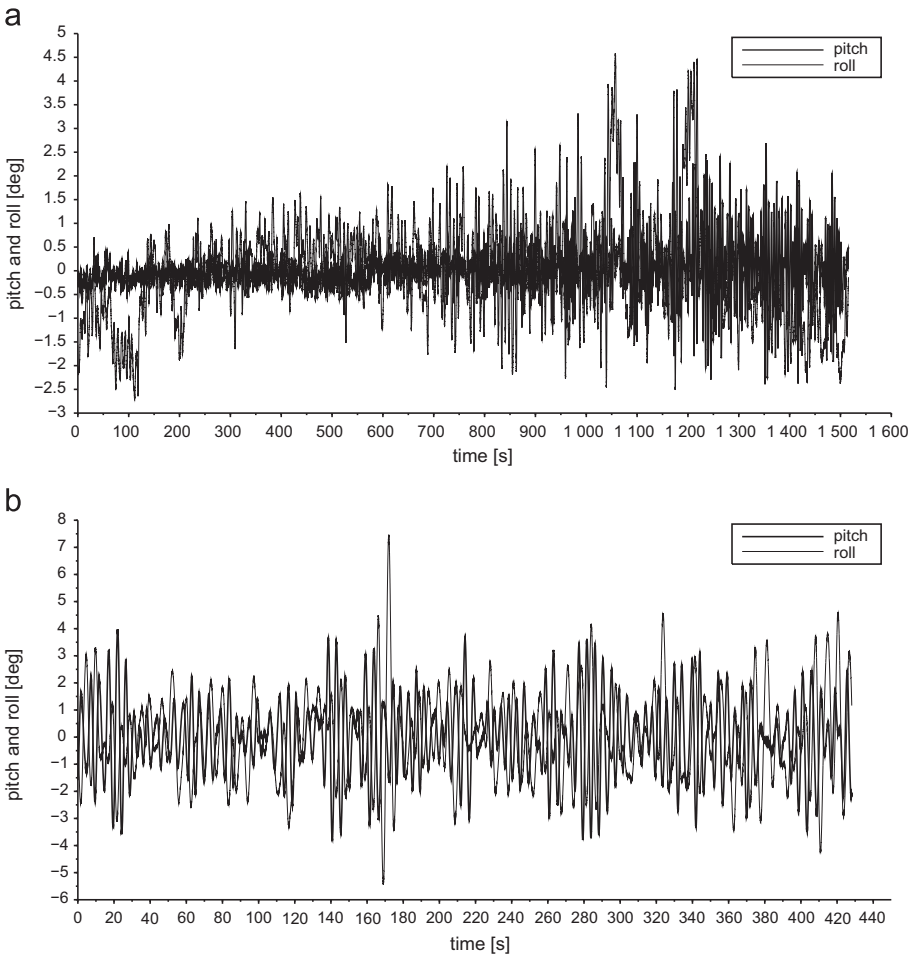

Fig. 11. Generalization part of recordings (a) *gen1*, (b) *gen2*.

Table 1
Parameter setting for AEEO ((∗)-number of evolutionary generations without progress which were necessary to add a new population with ANN-operations).

Parameter	Value
Max number of evolutionary generations	60 000
(∗)	10 000
Probability of crossover	0.7
Probability of cut-splice	0.1
Probability of mutation	0.06
Size of tournament	1
Number of elite individuals	1
Number of subpopulations (ANN-operations)	1–5
Size of subpopulations	50
No. of integer genes in chromosomes	7–20
Type of neurons in ANN-operations	Sigmoid, radial, linear, sinusoid, cosinusoid
Hidden neurons in ANN-operations	4

ANNs-operations as operations and data in AE evolve according to CCEGA. In CCEGA, each part of a solution evolves in a separate population. To form a complete solution, selected representatives (usually, the best ones) of each population are combined together. Application of this evolutionary scheme in AEEO consists in evolution of individual ANNs-operations in separate populations. The number of ANNs-operations in a complete program corresponds to the number of populations (see Fig. 7). Each population delegates exactly one representative ANN-operation to each program. During the evolution, programs expand gradually. In the beginning, they contain only one ANN-operation from one existing population. When the evolution stagnates, i.e. lack of progress in fitness of generated solutions is observed over some period, a set of populations containing ANNs-operations is enlarged by one population. This procedure extends all programs by one ANN-operation. Each population can also be replaced with a newly created population. Such situation takes place when the influence of all ANNs-operations from a given population on fitness of generated solutions is definitely lower than the influence of ANNs-operations from the remaining populations (a population can be replaced when, for

Table 2
Parameter setting for LRC, ARIMA (detailed parameter setting for ARIMA is given in appendix), RENN, and MRENN.

Parameter/method	Value
K-see formula (5)/LRC	100
σ-see formula (6)/LRC	Tuned individually
p,*d*,*q*/ARIMA	20,0,0 or 30,0,0
Type of neurons/RENN and MRENN	Sigmoid
Hidden neurons in RENN networks/RENN	4(RENN1),6(RENN2)
Hidden neurons in modules/MRENN	4(MRENN1,2),2–3(MRENN3)
Hidden neurons in feed-forward modules/MRENN	4
Number of modules/MRENN	5–10
P-see formula (12)/MRENN	1000
Number of clusters in α-parameter space/MRENN	10

Table 3
Prediction errors of pitch and roll for *gen1* and *gen2* (ARIMA1 – ARIMA(20,0,0), ARIMA2 – ARIMA(30,0,0)) – three the best results in each column (three for max and three for average errors) are bolded.

Prediction error (deg)	Roll *gen1*	Roll *gen2*	Pitch *gen1*	Pitch *gen2*
Max NOP	0.72181	1.10592	0.67228	1.3156
Average NOP	0.09647	0.29502	0.1015	0.22826
Max LRC	0.39941	0.50368	0.4827	0.73787
Average LRC	0.05777	0.09127	0.06952	0.12152
Max ARIMA1	0.20795	0.22585	0.25265	0.24691
Average ARIMA1	**0.03289**	0.03905	0.04564	0.04546
Max ARIMA2	0.19982	0.21338	0.26449	0.23589
Average ARIMA2	0.03299	**0.03568**	0.04789	0.04149
Max RENN1	0.18382	**0.18732**	0.19894	0.22207
Average RENN1	**0.0307**	0.03705	0.03606	0.04303
Max RENN2	0.19876	0.19921	0.21032	0.23494
Average RENN2	0.0359	0.03945	0.04033	0.04518
Max MRENN1a	**0.18191**	**0.18622**	**0.18782**	**0.21729**
Average MRENN1a	0.03295	0.03619	**0.03302**	**0.03959**
Max MRENN1b	**0.18265**	0.18762	**0.18811**	**0.21697**
Average MRENN1b	0.03447	**0.03167**	**0.03287**	**0.03957**
Max MRENN1c	0.19336	0.19211	0.19849	0.22223
Average MRENN1c	0.03854	0.03843	0.03954	0.04199
Max MRENN1d	0.2135	0.19974	0.20955	0.24832
Average MRENN1d	0.04211	0.03921	0.04184	0.0431
Max MRENN2	0.2212	0.21528	0.20527	0.23953
Average MRENN2	0.04537	0.04236	0.04065	0.04115
Max MRENN3	**0.17522**	**0.18032**	**0.18157**	**0.20294**
Average MRENN3	**0.02997**	**0.03098**	**0.03134**	**0.03424**

example, fitness of a population, measured as the average fitness of all ANNs-operations from the population, is definitely lower than the fitness of the remaining populations) [18].

In individual populations, the evolution proceeds according to Canonical GA [6]. At the genotypic level, each ANN-operation is represented in the form of a variable length chromosome consisting of two parts. The first short part defines topology of an ANN-operation whereas the second part includes parameters for the network (see Fig. 8). Construction of an ANN-operation proceeds in three phases. First, topology of an ANN-operation is determined based on the information contained in the first part of the chromosome. To this end, binary values from this part are directly copied into NDM, a single bit corresponds to a single item in the matrix. When the number of bits is insufficient to completely fill in the matrix, the whole sequence of bits is used again. In the following phase, the parameters from the second part of the chromosome are successively introduced into NDM. In this case, only items equal to one are modified. The remaining items, i.e. items equal to zero, remain intact. As before, transfer of the parameters is performed in a loop until all the elements in NDM have a value assigned. In the last phase, NDM is transformed into an ANN-operation [18].

5. Experiments

To test all the prediction methods presented in Section 3, experiments on real data were carried out. The data were acquired during two-day Baltic sea voyage on a navy ship. During the voyage, two parameters of ship spatial orientation were measured, i.e. pitch and roll. To measure the parameters, both INSs and accurate satellite systems (GNSS RTK) were applied (see Fig. 9). The task of INSs was to record the parameters with a high frequency equal to 50 Hz ($\Delta t_m = 0.02$ s) whereas the satellite systems were mainly used for verification purposes. The difference in measurements of both systems indicated recorded data which should not be taken into account during the experiments with prediction methods (Fig. 9).

Generally, two different recordings were selected, and then used in the experiments, out of all the recorded data. The first recording was measured for the wind of force 1–2 on the Beaufort scale and the speed of the ship equals to 10–12 knots whereas the

Table 4
Distances between ship camera/gun line of sight and observed object/target located 3000 m away from ship (ARIMA1 – ARIMA(20,0,0), ARIMA2 – ARIMA(30,0,0)) – three the best results in each column (three for max and three for average errors) are bolded.

Distance to object/target (m)	*gen1*	*gen2*
Max NOP	37.81	58.03
Average NOP	5.1	15.55
Max LRC	20.96	26.46
Average LRC	3.04	4.83
Max ARIMA1	10.94	11.88
Average ARIMA1	1.73	2.05
Max ARIMA2	10.52	11.22
Average ARIMA2	1.73	**1.87**
Max RENN1	**9.64**	9.83
Average RENN1	**1.62**	1.96
Max RENN2	10.45	10.48
Average RENN2	1.89	2.07
Max MRENN1a	**9.44**	**9.71**
Average MRENN1a	**1.55**	**1.74**
Max MRENN1b	9.7	**9.81**
Average MRENN1b	1.71	1.88
Max MRENN1c	10.2	10.51
Average MRENN1c	1.83	1.95
Max MRENN1d	11.22	10.52
Average MRENN1d	2.21	2.06
Max MRENN2	11.63	11.32
Average MRENN2	2.38	2.22
Max MRENN3	**9.21**	**9.49**
Average MRENN3	**1.57**	**1.63**

second recording for the wind force 3–4 and the ship speed 18–22 knots. Both recordings were divided into two parts, i.e. the learning and generalization part. Each part consisted of the following number of samples (subsequent measurements of pitch and roll): 20 000 – the learning part of the first recording (called *learn1* – Fig. 10(a)), 10 000 – the learning part of the second recording (called *learn2* – Fig. 10(b)), 75 972 – the generalization part of the first recording (called *gen1* – Fig. 11(a)), and 21 406 – the generalization part of the second recording (called *gen2* – Fig. 11(b)). The learning data were used to train neural networks, LRC and ARIMA whereas the generalization ones to ultimately verify abilities and assess accuracy of all the prediction methods.

As Figs. 10 and 11 show the amplitude of pitch and roll in the learning data amounted to $\pm 2^\circ$ whereas in the generalization ones even $\pm 8^\circ$. The disproportions result from the fact that the recordings had not been analyzed before they were split into the parts. The assumption was that if the learning and generalization parts come from the same recordings they should be similar in terms of the shape and amplitude. As it appeared, it was not the case, with the effect that conditions in which the prediction methods were trained and verified were different. Certainly, such a situation could have a negative influence on the effectiveness of the prediction methods operating on the generalization data, however, on the other hand, it also allowed the prediction methods to be tested in conditions considerably different from those encountered during the learning process.

As already mentioned, before all the prediction methods were tested on the generalization data, first, they were prepared to the task based on the learning data. To estimate an optimal value of LRC σ parameter, the method was run many times for different σ values. The values that led to the lowest prediction error for the learning data were then used during generalization tests. A similar procedure was applied in relation to ARIMA and its parameters p, d, and q. First, the learning data were used to estimate optimal parameters of ARIMA (α_i and β_j parameters, $i = 1..p, j = 1..q$) for different combinations of p, d, and q ($p = 0..30, d = 0..2, q = 0..30$). Then, the first thousand samples of generalization data were applied to identify the best setting for the method, the setting which was next used when ARIMA prediction capabilities were ultimately verified.

To train the neural networks, as mentioned before, AEEO was applied. Each individual network, regardless of the method, was prepared separately, which means that each of them arisen as a result of a separate evolutionary process. The prediction of pitch and roll was performed by different predictors – one predictor was responsible for pitch, the other one for roll. In RENN, MRENN1 and MRENN2, each network was trained based on complete sequences of learning data, while in MRENN3, separate sub-sequences were derived from the entire sequences, each sub-sequence corresponded to other ship state characterized by a parameter, and each module was trained on a sub-sequence whose parameter a matched a parameter assigned to the module.

To construct different networks, a number of AEEO runs were necessary. First, thirty RENN networks prepared to work independently were build, each of them evolved in a separate AEEO run. All these networks were, next, split into thirty groups of maximum ten networks (5–10 networks – see Table 2), the assignment to the groups were random. The groups were used to construct thirty layers consisting of RENN modules for MRENN1 and MRENN2 networks. The layers were the same for both variants of MRENN. To build feed-forward modules for MRENN1d and MRENN2, additional sixty AEEO runs were necessary, thirty runs per variant. MRENN3 networks were constructed independently, first, ten RENN modules for each a-parameter cluster were produced, then, as in the previous case, the modules were randomly assigned to thirty MRENN3 networks. Parameter setting for AEEO, and for all the prediction methods are presented in Tables 1 and 2.

In order to evaluate neural networks arisen during the evolutionary process, the following fitness function was applied:

$$F(network) = 1/L \sum_{i}^{L} (\hat{y}_i - y_{i+10})^2. \quad (19)$$

where

- $\hat{y}_i$ is a prediction performed when ith learning sample is measured,
- y_i is the ith learning sample,
- L is the number of samples in a learning sequence,
- $\hat{y}_i - y_{i+10}$ corresponds to $\Delta t_p = 0.2$ s (for comparison $\Delta t_m = 0.02$ s) – evaluation of a prediction is given after ten subsequent predictions.

Since the classical model of the ship to be the research platform on which the measurements were carried out does not exist, and is, generally, very difficult to build (model which treats the ship as a rigid body), as a point of reference for all the prediction methods presented in the paper, a solution was also applied that assumes invariability of ship state within Δt_p, which means that it calculates a predicted state as follows: $\hat{y}_i = y_i$. Since, in practice, this solution does not perform the prediction, it was called NOP, that is, no prediction. In spite of the fact that it is not, of course, a perfect point of reference to evaluate usefulness of all the proposed methods, it has one key advantage, namely, it may help to estimate what can happen in the case of no prediction.

5.1. Experimental results

Generally, the experiments summarized in Tables 3 and 4 showed that neural prediction tools proposed in the paper considerably reduce the influence of sea waves on the ship camera/gun, and in consequence, improve their effectiveness. As it turned out, the error of

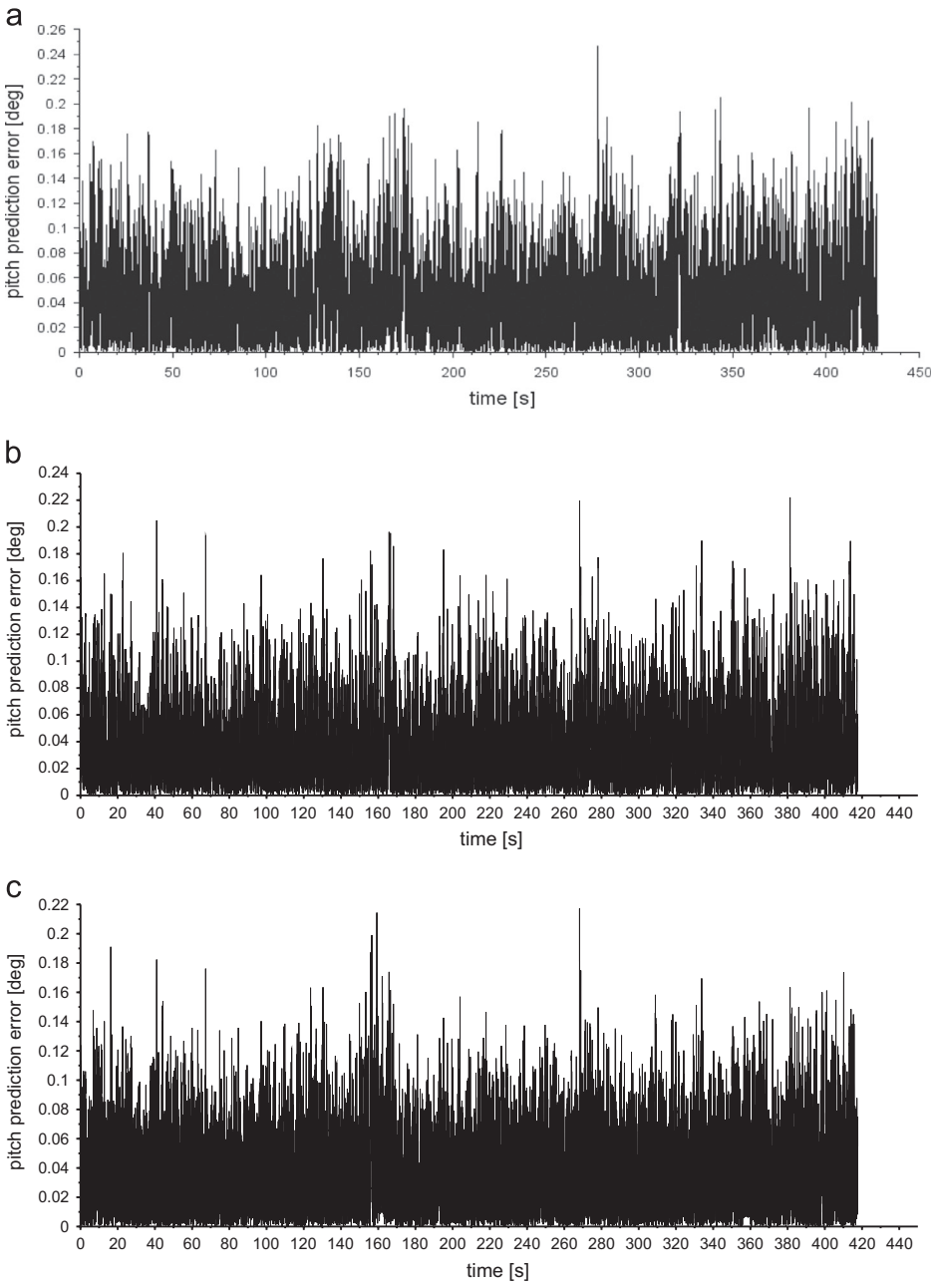

Fig. 12. Errors in pitch prediction for *gen2*. (a) ARIMA(20,0,0), (b) RENN, (c) MRENN1a.

predicting the ship orientation, and in effect, pointing the camera/gun at the object/target, can be reduced even six times in relation to the NOP (see MRENN3 results in the tables and Figs. 12 and 13).

When assessing the prediction methods presented in the paper from the point of view of their further application, it is necessary to state that accuracies achieved by them, it particularly applies to the neural methods and ARIMA, seem to be high in relation to conditions at sea for which the predictions were performed. Average error of focusing the camera/gun on the object/target located 3000 m away from the ship equal to one and half meter and the maximum error less than ten meters appear to be very small and sufficient for the camera/gun to effectively carry out their tasks. Of course, the change in pitch and roll is not the only factor affecting effectiveness of ship systems, the ship is also subject to progressive motions and it also changes yaw. In this case, however, the impact on the ship systems is much smaller, for example, the change of ship location within 0.2 s, the ship going at a speed of 20 knot, amounts to about 10 cm. This 10 cm is an error that would have been made, regardless of the distance to the object/target, when the only ship motion would be the move forward. In the case of yaw, which never changes rapidly at sea, the influence is even smaller.

The other serious problem for the ship vision/artillery systems is that the tracked object/target is not stationary and its motion also should be modeled. To this end, radar and/or vision systems supported by a specialized software implementing different patterns of object motion are used. As in the case of ship motion, also this time we deal with errors in prediction, which in contrast to errors associated with ship progressive motions and change of yaw, are considerably larger. It is, however, a separate issue which is not the subject of this paper.

When comparing the prediction methods presented in the paper, it can be noted that all the ones implemented in the form of evolutionary neural networks are more than twice as accurate in

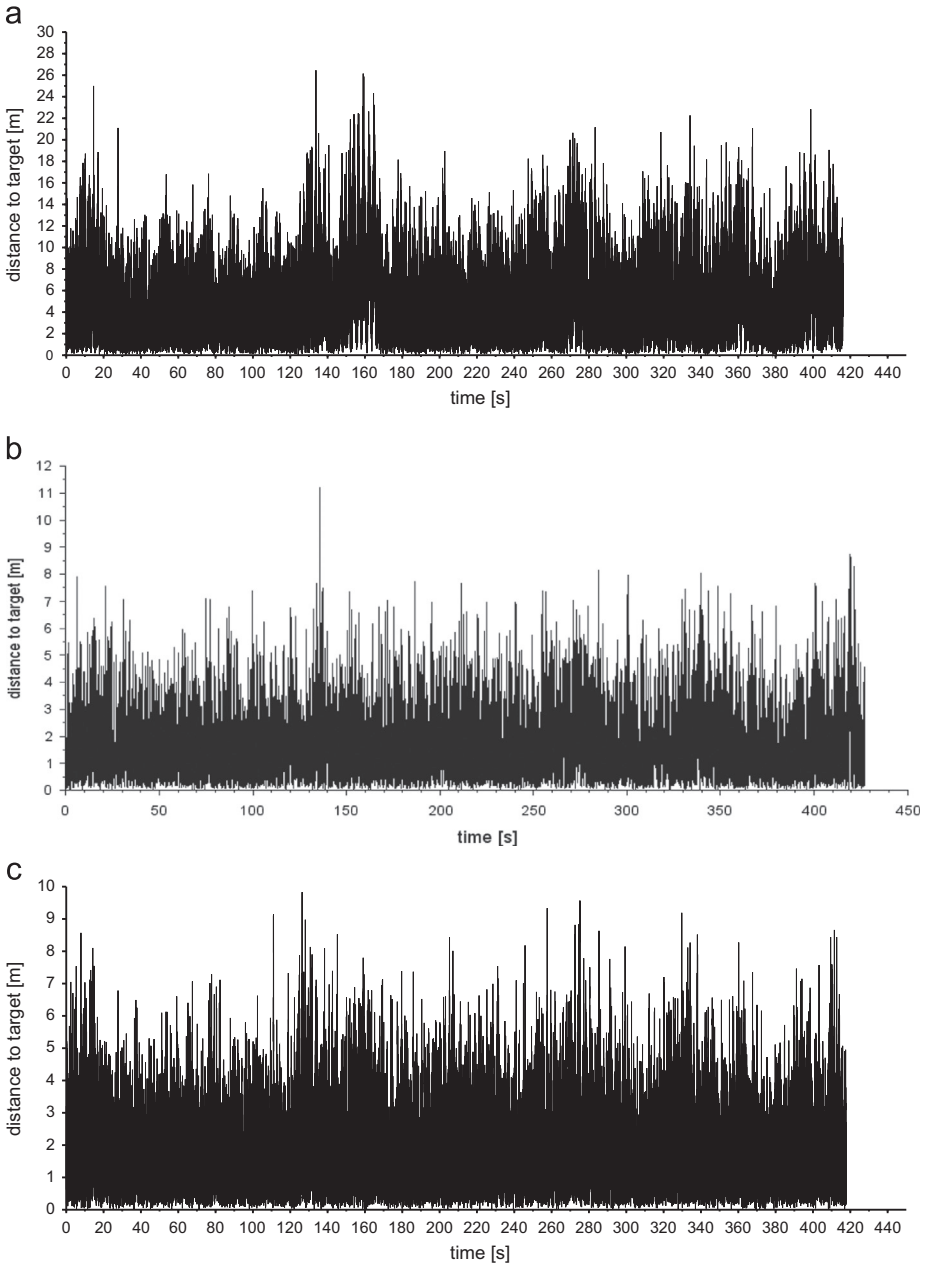

Fig. 13. Distances between camera/gun line of sight and observed object/target located 3000 m from ship for *gen2* (a) LRC, (b) ARIMA(30,0,0), (c) RENN2.

their predictions as LRC. For example, maximum distance between camera/gun line of sight and observed object/target when using LRC is more than 20 m, whereas for RENN1, MRENN1ab and MRENN3, it is below 10 m. As regards ARIMA predictions, they are more accurate than LRC, however, still less precise than most predictions made by neural networks. Differences in accuracy between ARIMA and neural predictors are definitely smaller than in the case LRC, however, for the most effective networks mentioned above they are noticeable.

Results of the neural predictors are, generally, very similar which may be due to application of the same recurrent RENN networks in most predictors (except MRENN3). High effectiveness of RENN1 modules used in modular networks made the latter insensitive to fusion methods, any method of combining accurate module predictions resulted in accurate final predictions. The only case when results of RENN1 networks were improved through usage of modularity is MRENN1a and MRENN3. In the case of MRENN1a, the cause may be the fact that this solution works as a simple filter which eliminates all possible partial noisy predictions through averaging signals from many modules. In other fusion approaches, weights assigned to individual modules are different, and in consequence, the impact of one modules on the final prediction is greater than remaining ones. The effect is that wrong weight assignments lead to worse results than the simple averaging of weights.

In MRENN1b, the weights are fixed permanently during the training process based on evaluations of individual RENN modules. Since, as already mentioned, the training data differed from those used for verification purposes, the evaluations of the modules, and in consequence their weights, could be inappropriate in some situations.

In turn, in MRENN1c, the evaluation of the modules, and thereby, the assignment of weights to each of them is performed continuously during the prediction, and to this end, the prediction errors are used. Moreover, the modules are weighted separately for each specified ship state. This approach, however, turned out to

be less effective than the previous one, and difficulties in reliable evaluation of the modules for each state are the most likely cause of such a situation. To evaluate usefulness of a module in a given ship state, average prediction errors committed by the module in states similar to the given state were used. To determine which states are similar, the ship state space was manually divided into clusters of the same size. The consequence of the manual division was, however, imprecise evaluation of modules, a module which was both accurate and inaccurate for states grouped in one cluster received an average evaluation which was not a good indication for the allocation of weights.

Applying neural networks as the fusion or weighting modules did not change the situation, what is more, modular networks equipped with extra fusion/weighting modules turned out to be the least effective of all the neural solutions. It seems that the most likely cause of this state of affairs is, in this case, the difference between what the fusion/weighting modules have learned during the training process and what they later encountered during the operation. The task of fusion/weighting modules was to combine outputs of RENN modules adequately to the current ship state. Since ship states encountered during the operation were partly different from those used to train the fusion/weighting modules, the latter were not appropriately prepared to make accurate predictions in all new states, their generalization capacities appeared to be insufficient.

As the experiments showed, MRENN3 turned out to be the most accurate prediction solution. It seems that a high accuracy of this method was accomplished by dividing the whole prediction problem into simpler sub-problems, in this case, each recurrent module was responsible for prediction in other ship state. In consequence, each of them had a separate easier task to perform, was trained on a separate training data, was simpler in terms of architecture than RENN modules used in other modular solutions, was specialized in a dedicated task, and in effect, was highly accurate.

6. Conclusion

The paper reports experiments on applying different prediction methods, mostly the neural ones, to predict spatial orientation of a ship. Generally, the experiments showed that even in difficult conditions at sea it is possible to predict behavior of a ship to the extent that allows ship vision/artillery systems to be effectively used. The accuracies accomplished in the experiments are high enough to conclude that solutions proposed in the paper can be a valuable alternative or a support for existing systems operating on the basis of data acquired from expensive, very accurate gyroscopic devices. Instead of applying the expensive gyroscopes to perform predictions, the proposed approach suggests to use cheap MEMS INSs which are able to provide a reliable angle information in short periods thanks to fusion signals from noisy gyroscopes, accelerometers, and a three-axis magnetometer. As the experiments revealed, this information can be then used by a neural model of a ship to accurately predict its behavior.

Acknowledgments

The paper is supported by the Project no. O ROB 0046 03 001 granted by the Polish National Center of Research and Development.

Appendix A. Parameter setting for ARIMA

See Figs. A1–A4.

ARIMA(20,0,0) Model:
Conditional Probability Distribution: Gaussian

Parameter	Value	Standard Error	t Statistic
Constant	0.00301975	0.000174205	17.3344
AR{1}	1.02645	0.00708296	144.919
AR{2}	0.0699841	0.0101141	6.91944
AR{3}	-0.0426138	0.010055	-4.23809
AR{4}	0.00235211	0.0100488	0.23407
AR{5}	-0.0215231	0.01008	-2.13523
AR{6}	0.0925199	0.0101214	9.14098
AR{7}	-0.10788	0.0100897	-10.6921
AR{8}	0.0388432	0.0101024	3.84495
AR{9}	-0.04383	0.0102136	-4.29136
AR{10}	0.0562056	0.00999767	5.62187
AR{11}	-0.0830091	0.0100298	-8.27625
AR{12}	0.114139	0.010047	11.3606
AR{13}	-0.107795	0.00998303	-10.7978
AR{14}	0.0670372	0.0100342	6.68089
AR{15}	-0.0459741	0.0101563	-4.52667
AR{16}	0.0460613	0.0100254	4.59448
AR{17}	-0.0645448	0.010099	-6.39121
AR{18}	0.0507347	0.0101217	5.01247
AR{19}	-0.0292758	0.0101419	-2.88662
AR{20}	-0.0219663	0.0070649	-3.10922
Variance	5.31747e-05	5.167e-07	102.912

Fig. A1. Snapshot from MATLAB including ARIMA(20,0,0) parameters determined based on data from *learn1*.

ARIMA(20,0,0) Model:
Conditional Probability Distribution: Gaussian

Parameter	Value	Standard Error	t Statistic
Constant	0.00279457	0.000248187	11.26
AR{1}	1.04887	0.0101486	103.351
AR{2}	0.0169978	0.0145893	1.16508
AR{3}	-0.0151391	0.0143308	-1.0564
AR{4}	0.0181998	0.0142289	1.27907
AR{5}	-0.0617033	0.0144571	-4.26803
AR{6}	0.13629	0.014673	9.2885
AR{7}	-0.120756	0.0144202	-8.37408
AR{8}	0.0244389	0.0145982	1.6741
AR{9}	-0.0175934	0.0145657	-1.20786
AR{10}	0.0604018	0.0142083	4.25115
AR{11}	-0.101402	0.0144089	-7.03746
AR{12}	0.113568	0.014433	7.86866
AR{13}	-0.100948	0.0143322	-7.04342
AR{14}	0.0661411	0.0143997	4.59323
AR{15}	-0.0513183	0.0144664	-3.54743
AR{16}	0.0672009	0.014327	4.6905
AR{17}	-0.103997	0.0146109	-7.11777
AR{18}	0.0479446	0.0145131	3.30354
AR{19}	-0.0260739	0.0144245	-1.80761
AR{20}	-0.00462226	0.00993765	-0.465126
Variance	5.14929e-05	7.06005e-07	72.9356

Fig. A2. Snapshot from MATLAB including ARIMA(20,0,0) parameters determined based on data from *learn2*.

ARIMA(30,0,0) Model:

Conditional Probability Distribution: Gaussian

Parameter	Value	Standard Error	t Statistic
Constant	0.00337834	0.000189864	17.7934
AR{1}	1.02438	0.00707759	144.735
AR{2}	0.0656232	0.0101023	6.49584
AR{3}	-0.03712	0.0100511	-3.69312
AR{4}	-0.00286292	0.0100364	-0.285254
AR{5}	-0.0170806	0.0100829	-1.69401
AR{6}	0.0865498	0.0101391	8.53627
AR{7}	-0.100864	0.010115	-9.97175
AR{8}	0.0310758	0.010125	3.0692
AR{9}	-0.035651	0.0102577	-3.47553
AR{10}	0.0465774	0.0100839	4.61898
AR{11}	-0.0734659	0.0101154	-7.26278
AR{12}	0.104125	0.0101174	10.2917
AR{13}	-0.0973594	0.0100541	-9.68353
AR{14}	0.0540971	0.0101136	5.34893
AR{15}	-0.0353957	0.0102421	-3.45591
AR{16}	0.0361158	0.0101353	3.56337
AR{17}	-0.0547135	0.0102321	-5.34726
AR{18}	0.0398999	0.010233	3.89913
AR{19}	-0.020426	0.0102307	-1.99653
AR{20}	0.0118628	0.010188	1.1644
AR{21}	-0.0097948	0.010214	-0.958961
AR{22}	0.0144601	0.010167	1.42225
AR{23}	-0.0337237	0.0100692	-3.3492
AR{24}	0.0281589	0.010168	2.76937
AR{25}	-0.0272059	0.0101594	-2.67791
AR{26}	0.0364387	0.010079	3.61532
AR{27}	-0.0481842	0.00995309	-4.84113
AR{28}	0.00823675	0.0100777	0.817321
AR{29}	-0.0299999	0.0101244	-2.96313
AR{30}	0.0316835	0.00705579	4.49042
Variance	5.27993e-05	5.13685e-07	102.785

Fig. A3. Snapshot from MATLAB including ARIMA(30,0,0) parameters determined based on data from *learn1*.

ARIMA(30,0,0) Model:

Conditional Probability Distribution: Gaussian

Parameter	Value	Standard Error	t Statistic
Constant	0.0031012	0.000265736	11.6702
AR{1}	1.0511	0.010139	103.669
AR{2}	0.00825314	0.014639	0.563779
AR{3}	-0.00660926	0.0143484	-0.460627
AR{4}	0.00874461	0.0142484	0.613725
AR{5}	-0.0517065	0.0144895	-3.56855
AR{6}	0.122563	0.0147266	8.32251
AR{7}	-0.106919	0.0144895	-7.37903
AR{8}	0.0112828	0.0146662	0.76931
AR{9}	-0.00537126	0.014634	-0.367038
AR{10}	0.0448611	0.0143132	3.13424
AR{11}	-0.0856121	0.0145594	-5.88021
AR{12}	0.0991005	0.0145167	6.82663
AR{13}	-0.0885619	0.014441	-6.13266
AR{14}	0.0543593	0.0144925	3.75086
AR{15}	-0.0382942	0.0146303	-2.61745
AR{16}	0.0504793	0.0144937	3.48284
AR{17}	-0.0862771	0.0148326	-5.81672
AR{18}	0.0336442	0.0147055	2.28787
AR{19}	-0.0176557	0.01458	-1.21096
AR{20}	0.0322107	0.0146596	2.19724
AR{21}	-0.042415	0.0148416	-2.85785
AR{22}	0.0646891	0.0147906	4.37367
AR{23}	-0.0709757	0.0145296	-4.8849
AR{24}	0.0421606	0.0148205	2.84475
AR{25}	-0.0089753	0.0147399	-0.60891
AR{26}	0.00208418	0.014538	0.14336
AR{27}	-0.034665	0.0142086	-2.43973
AR{28}	0.0289157	0.0145372	1.98908
AR{29}	-0.0448955	0.0144462	-3.10778
AR{30}	0.0306084	0.0100118	3.05723
Variance	5.10479e-05	6.99564e-07	72.971

Fig. A4. Snapshot from MATLAB including ARIMA(30,0,0) parameters determined based on data from *learn2*.

References

[1] O. Anava, E. Hazan, S. Mannor, O. Shamir, Online learning for Time Series prediction, in: JMLR: Workshop and Conference Proceedings, vol. 2013, 2013, pp. 1–13.
[2] G. Box, G. Jenkins, G. Reinsel, Time Series Analysis: Forecasting and Control, 3rd edition, Prentice-Hall, Englewood Cliff, N.J, 1994.
[3] P. Brockwell, R. Davis, Time Series: Theory and Methods, 2nd edition, Springer, New York, 2009.
[4] O. Castillo, P. Melin, Hybrid intelligent systems for time series prediction using neural networks, fuzzy logic, and fractal theory, IEEE Trans. Neural Netw. 13 (6) (2002) 1395–1408.
[5] D. Fernando, A. Jayawardena, Runoff forecasting using RBF networks with OLS algorithm, J. Hydrol. Eng. 3 (3) (1998) 203–209.
[6] D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison Wesley, Reading, MA, 1989.
[7] J. Hamilton, Time Series Analysis, Princeton University Press, Princeton, New York, 1994.
[8] Y. Kajitani, K.W. Hipel, A.I. Mcleod, Forecasting nonlinear time series with feed-forward neural networks: a case study of Canadian lynx data, J. Forecast. 24 (2005) 105–117.
[9] V. Kodogiannis, A. Lolis, Forecasting financial time series using neural network and fuzzy system-based techniques, Neural Comput. Appl. 11 (2002) 90–102.
[10] J.F. Kolen, S.C. Kremer, A Field Guide to Dynamical Recurrent Networks, Wiley-IEEE Press, New York, 2001.
[11] A. Lendasse, D. Francois, V. Wertz, M. Verleysen, Nonlinear Time Series Prediction by Weighted Vector Quantization, ICCS 2003, Lecture notes on Computer Science, vol. 2657, 2003, pp. 417–426.
[12] T. Lin, B.G. Horne, P. Tino, C.L. Giles, Learning long-term dependencies in NARX recurrent neural networks, IEEE Trans. Neural Netw. 7 (6) (1996) 1424–1438.
[13] J. Matusiak, Dynamics of a Rigid Ship, Aalto University, freely available book, 2013. https://aaltodoc.aalto.fi/handle/123456789/10265.
[14] M. Potter, The design and analysis of a computational model of cooperative coevolution (Ph.D. thesis), George Mason University, Fairfax, Virginia, 1997.
[15] M.A. Potter, K.A. de Jong, Cooperative coevolution: an architecture for evolving coadapted subcomponents, Evolut. Comput. 8 (1) (2000) 1–29.
[16] T. Praczyk, P. Szymak, Decision system for a team of autonomous underwater vehicles - preliminary report, Neurocomputing 74 (17) (2011) 3323–3334.
[17] T. Praczyk, Solving the pole balancing problem by means of assembler encoding, J. Intell. Fuzzy Syst. 26 (2014) 857–868.
[18] T. Praczyk, Using augmenting modular neural networks to evolve neuro-controllers for a team of underwater vehicles, Soft Comput. 18 (2014) 2445–2460. http://dx.doi.org/10.1007/s00500-014-1221-0.
[19] T.Praczyk, P. Szymak, Prediction of spatial orientation of 35 mm gun ship, Conference Proceedings of 5th International Conference on Scientific Aspects of Unmanned Mobile Object (SAUMO), Deblin, 2013.
[20] A. Rodrigues, P. de Mattos Neto, J. Albuquerque, S. Bocanegra and T. Ferreira, Forecasting chaotic and non-linear time series with artificial intelligence and statistical measures, modelling simulation and optimization, in: Gregorio Romero Rey and Luisa Martinez Muneta (Eds.), InTech, Rijeka, Croatia, ISBN: 978-953-307-048-3, Available from: http://www.intechopen.com/books/modelling-simulationand-optimization/forecasting-chaotic-and-non-linear-time-series-with-artificial-intelligence-and-statisticalmeasures (2010).
[21] R. Shumway, D. Stoer, Time Series Analysis and Its Applications, Springer, New York, 2005.

Tomasz Praczyk received his M.Sc. degree from Cybernetics Faculty, Military Technical Academy, in Warsaw, 1991–1996. Computer Center of Polish Navy, 1996–2002. He did his Ph.D. degree from Polish Naval Academy, 2002. He worked as an Assistant Professor at Polish Naval Academy, Computer Science and Naval Weapon Institute, 2002. He did his post-doctoral studies at Military Technical Academy, Cybernetics Faculty and his interests include diving, artificial intelligence, and naval weapon.