Original Article

Institution of MECHANICAL ENGINEERS

Journal of MECHANICAL ENGINEERING SCIENCE

Proc IMechE Part C:
J Mechanical Engineering Science
2014, Vol. 228(8) 1444–1458

DOI: 10.1177/0954406213507704
pic.sagepub.com

SAGE

Neural networks for prediction of robot failures

Ali Diryag, Marko Mitić and Zoran Miljković

Abstract

It is known that the supervision and learning of robotic executions is not a trivial problem. Nowadays, robots must be able to tolerate and predict internal failures in order to successfully continue performing their tasks. This study presents a novel approach for prediction of robot execution failures based on neural networks. Real data consisting of robot forces and torques recorded immediately after the system failure are used for the neural network training. The multilayer feedforward neural networks are employed in order to find optimal solution for the failure prediction problem. In total, 7 learning algorithms and 24 neural architectures are implemented in two environments – Matlab and specially designed software titled BPnet. The results show that the neural networks can successfully be applied for the problem in hand with prediction rate of 95.4545%, despite having the erroneous or otherwise incomplete sensor measurements invoked in the dataset. Additionally, the real-world experiments are conducted on a mobile robot for obstacle detection and trajectory tracking problems in order to prove the robustness of the proposed prediction approach. In over 96% for the detection problem and 99% for the tracking experiments, neural network successfully predicted the failed information, which evidences the usefulness and the applicability of the developed intelligent method.

Keywords

Neural networks, prediction, execution failures, mobile robot, obstacle detection, trajectory tracking

Date received: 1 April 2013; accepted: 12 September 2013

Introduction

Nowadays, one of the most desirable features of every robotic system is the ability to adapt to the real-world changing conditions. This is especially important for robots working in the hazardous and dangerous surroundings where unwanted events frequently interfere in task accomplishment. Likewise, failure prediction is equally important in these environments in which repairs are often infeasible and failures can have disastrous consequences.[1] In industrial robotics, failure prediction and fault tolerance are helpful in reduction of a system downtime in the following manner: by tolerating failures robot's lifespan is increased, and by identifying faulty components or subsystems to speed up the repair process.[2] Also, the reliability of a product manufacturing and increased human safety is ensured by implementing fault tolerance and failure prediction unit in the robotic system.

Failure tolerance has been addressed in various applications for robot manipulators. Usually, redundancy approach in actuation,[3] sensors[4] or joints[5] is used. Likewise, different methods are employed for solving the failure detection problem such as second-order sliding-mode algorithm,[6] robust non-linear analytic redundancy technique[7] or partial least squares approach.[8] In mobile robotics, the fault detection is treated by implementing a torque filtering technique,[9] multiple model adaptive estimation method[10] or an interacting approach.[11] Recently, several interesting studies have been reported regarding the failure prediction problem. In Ref. 12, the method that utilizes the concept of augmented global analytical redundancy relations to handle failures with both parametric and non-parametric nature is presented. Additionally, multiple hybrid particle swarm optimization algorithm is employed in order to realize multiple failures prediction. Twala addressed the robot execution failure prediction using incomplete data in Ref. 13. Here, this prediction is formulated as a classification problem which is solved by developing a novel probabilistic approach. Likewise, the work given in Ref. 14 presents

Production Engineering Department, Faculty of Mechanical Engineering, University of Belgrade, Belgrade, Serbia

Corresponding author:
Zoran Miljković, Production Engineering Department, Faculty of Mechanical Engineering, University of Belgrade, Kraljice Marije 16, 11120 Belgrade 35, Serbia.
Email: zmiljkovic@mas.bg.ac.rs

the performance comparison of base-level and meta-level classifiers on the same problem. The results show the superiority of Bagged Naïve Bayes classifier across different settings. However, none of the aforementioned studies incorporate learning techniques in order to improve presented solutions. In this study, neural networks (NNs) are employed for prediction of robot execution failures in order to solve the non-linear dependencies between input and output variables.

NNs are a well-known tool used as a solution for various engineering problems.[15] They can understand the relationship or mapping between input and output variables during the training process using different learning algorithms. The applications of this machine learning method are very diverse: It can be used for prediction of vehicle reliability performance[16] or in education to predict professional movements of graduates.[17] In robotics, this artificial intelligence technique is often applied for control of a mobile robot[18,19] or a robot manipulator.[20,21] For failure problems, the NNs are employed in the assembly tasks,[22] prediction of failure rates of large number of the centrifugal pumps[23] or in the robust scheme for robot manipulators.[24] However, despite various mentioned applications, the robot failure prediction based on the soft computing methods has not been reported in the literature so far. This study delivers a novel approach using multilayer feedforward NNs as a solution for this problem and also presents performance comparison of different learning algorithms and architectures. In addition, the original prediction method is successfully implemented and tested on a real mobile robot in indoor environment for solving obstacle detection and trajectory tracking problems.

The rest of this paper is organized as follows. The problem definition and main achievements are stated in 'Problem statement and paper contributions' section. 'Materials and Methods' section introduces execution dataset and software used in this study, while 'Prediction procedure' section describes entire learning process. 'Results and discussion' section shows experimental outcomes and interpretation of the obtained data. Final remarks are given in 'Conclusions' section.

Problem statement and paper contributions

The problem treated in this study refers to the failure detection in a robot system; more specifically, this work treats the robotic failure prediction problem using NNs and a set of recorded sensor measurements. Consider a robotic system working in a structured or unstructured environment exposed to severe conditions such as increased working hours, changeable working demands, possibility of collision with known/unknown objects and/or presence of human workers near the robot workspace. In these cases it is crucial to ensure maximum safety and smallest deviation from the nominal operating mode by recognizing irregularities in robot behaviour. The prediction of robot failures is equally important, since this can provide a continuous and undisturbed work using a backup emergency control commands.

In order to successfully predict execution failures, some sort of safety unit must be employed in the robotic system. In this case, the NNs are used in the control system as an element for predicting misbehaviour based on the corrupted internal/external measurements. For example, one can consider obstacle detection problem and an irregular work of several infrared sensors. Given the set of correct sensor values for a particular case (for example, obstacle on the left side of the robot), the robot with the installed NN-based safety element can predict if one or more sensors are malfunctioning. After this, the incorrect sensor measurements can be ignored or replaced with their initial (i.e. nominal) value. In that way using this prediction approach, the system is enabled to work in unobstructed regime and to successfully detect different obstacles. Likewise, the trajectory tracking problem can be treated in the same manner; for example, the NN-based unit can be used to predict irregular behaviour in wheel control domain. Consider that mobile robot wheels command unit is not working properly all the time, and that in certain control iterations it gives unexplainable large/small commands for tracking the specific trajectory. In this case, NNs can predict these irregularities, with the aim to invoke a nominal control value in the command dataset. In this manner, the bad wheel command is replaced with the desired (calculated) value, and the robot motion is continued without difficulties.

In order to summarize the findings in this paper, the contributions are explicitly stated next. The main achievements of this study are threefold:

1. To the authors' best knowledge, this is the first paper that involves NNs in prediction of robot execution failures using sensor data. Real forces and torques recorded during execution of a specific task are used to train NNs in order to predict one of four possible working cases (normal, collision, front collision and obstruction). The erroneous data are also implemented in the NN training set in order to fully investigate robustness of the proposed approach.
2. Various NN learning algorithms and architectures are tested in order to find optimal solution. In total, 6 algorithms and 24 neural architectures are tested in the Matlab environment. Additionally, another prediction tool is used in this study – specially designed software titled BPnet[25] which employs most common feedback method for minimizing the error between input and output variables – backpropagation (BP) technique.[26]

3. Real-world experiments are conducted on a *KheperaII* mobile robot in indoor environment for solving obstacle detection and trajectory tracking problems with the aim to additionally verify the proposed method. The experimental results confirmed that NNs successfully predict robot failures and that this approach ensures nominal working mode during the entire task execution.

Materials and methods

Robot execution data

As mentioned before, this paper considers the NN prediction ability concerning robot failures so as to successfully detect and classify failures and to dependently track and monitor the action execution. The data used in this study are obtained from a real system and refers to the evolution of forces (F) and torques (T) during execution of a specific task. In order to correctly evaluate and compare various NN algorithms and architectures, the failures in approach to grasp position are considered. Each feature in the dataset represents a force or torque value measured immediately after failure detection. Total number of instances is 88, and each instance consists of 15 sensor measurements (i.e. samples) collected at regular time intervals. Three values of forces and torques are found in each sample; therefore, one instance has 90 different features (i.e. the values of F and T). These data are publicly available via well-known machine learning repository.[27]

In the failure dataset, four different robot situations (i.e. data classes) can be identified: normal, collision, obstruction and front collision with the distribution of 24%, 19%, 18% and 39%, respectively. The identification of particular class is based on the values and/or relationships between measured forces and torques. As an example, in Figure 1 the Fx and Tx in one instance for each robot situation are presented. It is obvious that the values are very different, which is especially suitable for NN prediction purposes.[15]

Neural networks

NNs are inspired by biological neurological system and are composed of simple processing elements called artificial neurons or nodes capable of performing massive parallel computations for data processing and knowledge representation.[15,28] The neurons are able to communicate between themselves and to exchange information through the biased or weighted connections. The NNs have been found to be both reliable and effective when applied to applications involving prediction, classification and clustering.[29] The most frequent areas of NNs applications are production/operations (53.5%) and finance (25.4%).[30]

NNs often find usage in cases when dealing with noise in data, in the situations when data contains complex relationships between many factors or when other mathematical techniques or methods are not adequate.[31] By adjusting weights iteratively between the neurons in different layers, the network is able to find hidden rules between the data. The main advantages of NNs are their information processing abilities such as non-linearity, high parallelism, robustness, fault and failure tolerance, learning, ability to handle imprecise information and their capability to generalize.[32] Therefore, the prediction of robot execution failures appears to be an appropriate assignment for NNs.

Each neuron in NN is active or non-active based on the adding value and activation function value. Adding value is determined by summarizing all inputs to the particular cell modified by their weighting coefficients, while activation function affects amplitude of the neuron output. After defining these

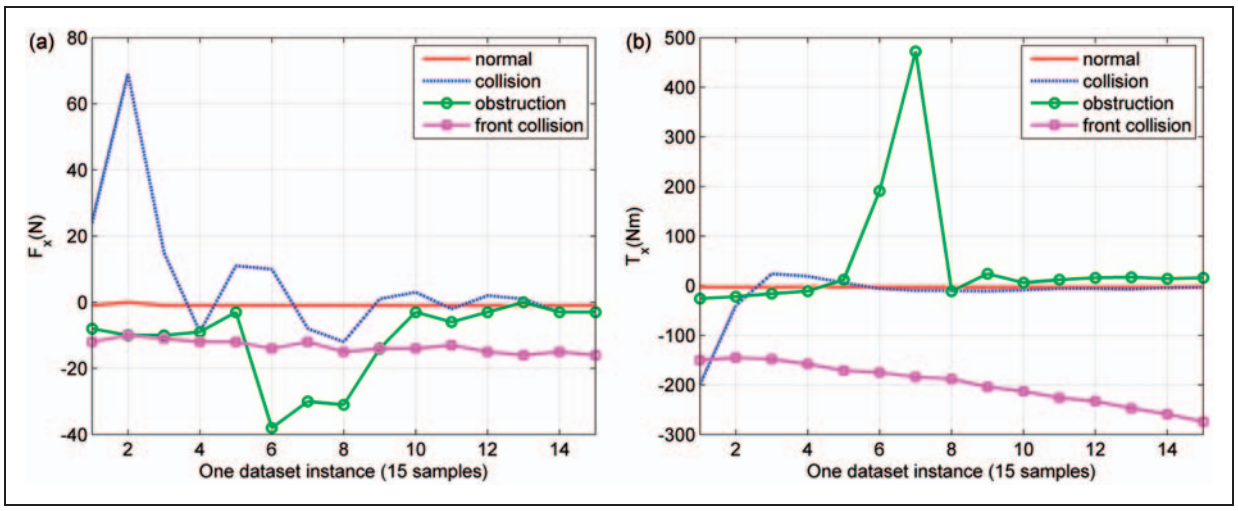

Figure 1. An example of force and torque value in one dataset instance: (a) Fx, (b) Tx.

neuron components, the training process in a supervised manner is set to start. Firstly, an input to the each neuron in the first (i.e. input) layer must be defined. The weights between an input neuron and the neurons in hidden layer indicate the degree of importance between these units. Thus, the strength of connections between neurons is given by the numerical value between –1 and 1 which represents aforementioned weight number. Secondly, the output value for each neuron is calculated by using weighted input through the activation function. If that value is larger than the neuron internal threshold, the processing unit is activated; otherwise, there is no output from that particular neuron. After the calculation of outputs from every neuron in the network, the error between the output values in the last (i.e. output) layer and the pre-defined desired output is calculated. Then, that error is propagated backwards from the output to the input layer in order to determine new network weights that will decrease the difference between the desired and actual output. This iterative procedure is finished when these values are close enough, i.e. when they are below the pre-defined learning threshold. After the training step is over, a validation and testing are active next. In the validation phase, the length of NN training, learning parameters and number of units in hidden layers are optimized. The testing phase represents network performance evaluation on a new sample, and the result is taken as the assessment of the NN. Finally, the network with the optimal performance is used as a solution for the problem in hand.

BPnet software

In this study, the NN-based robot failure prediction is conducted in the Matlab NN toolbox and in the BPnet software. We tested the NNs using different learning algorithms in Matlab and using most common feedback methods (i.e. the BP) in BPnet software.

BP is an iterative gradient-descent algorithm designed to minimize the mean squared error (MSE) between the actual output and the desired output as specified in the training set.[17,26] The software BPnet employs BP technique and is developed in the Laboratory for Industrial Robotics and Artificial Intelligence at the Faculty of Mechanical Engineering in Belgrade, primarily for the needs of implementing sensor–motor coordination of learning robot and camera calibration.[25] Wide ranges of applications using BPnet are established; for example, it is involved in the domain of intelligent robot control[33] as well as for predicting professional choices of secondary school graduates.[17] In this work, the basic idea for BPnet engagement was to test our method using two independent software packages. In that way, the obtained results are more credible regarding prediction problem solved by NNs. Likewise, the software proved to be very useful in previous applications in the robotic domain,[25,33] which represents an additional argument for its utilization.

BPnet software was developed in *Visual Basic* programming language.[34] User friendly interface of the software enables that NN topology and initial weighting coefficients are easily defined. The starting window (Figure 2(a)) shows basic information about the software. After the 'proceed' button is pressed and the project is named, four different steps for defining BP NN in BPnet software are available. Primarily, the number of layers and number of neurons in each layer are defined in the 'configuration' step. By using three buttons located on the top of the window (Figure 2(b)), every NN architecture can be easily set. The end of this step is conducted using the 'check' mark on the left side. Likewise, by pressing on the 'x' button misentered neurons in each layer can be deleted. The weighting coefficients are defined in the next module – in the 'connections' step. Initial weighting coefficients are defined by default and are shown in Figure 2(c). These can be varied in order to obtain their optimal value for the problem in hand. This stage is completed in the same manner as previous by confirming weighting coefficients and bias values using the 'check' mark. Third step implies implementation of input/output training pairs. In this module we can also open earlier work or save a new training data in a text file. Finally, the training and testing phase is conducted in the 'train' module. Here, we can define expected (i.e. goal) middle absolute error (MAE) as well as network learning parameters. Testing is also conducted here by invoking the new input/output pairs after the training phase is over. An example of the BPnet engagement during the training process with NN architecture 6–10–4 (6 neurons in input layer, 10 and 4 in hidden and output layer, respectively) is shown in Figure 2(d). One can notice that the BP algorithm successfully decreased the NN error below the previously defined value after ~25,000 iterations.

Prediction procedure

In order to evaluate the NN performance, the collected data need to be distributed on training and testing set. Generally, different variables are represented in different order of magnitude; thus, in order to ensure that every data unit receives the same influence in the training procedure, it needs to be normalized. In this study, the data are divided in training and testing randomly in a ratio of 7:3. In other words, the Matlab training set includes 70% of 1320 sensor measurements (88 instances ×15 samples in each instance), randomly chosen over the entire dataset. The rest of the data is used to test the NN network. In addition, a small portion of data is replaced with erroneous samples; 0–3% of real sensor measurement is randomly replaced in the training and testing process. For BPnet software, the

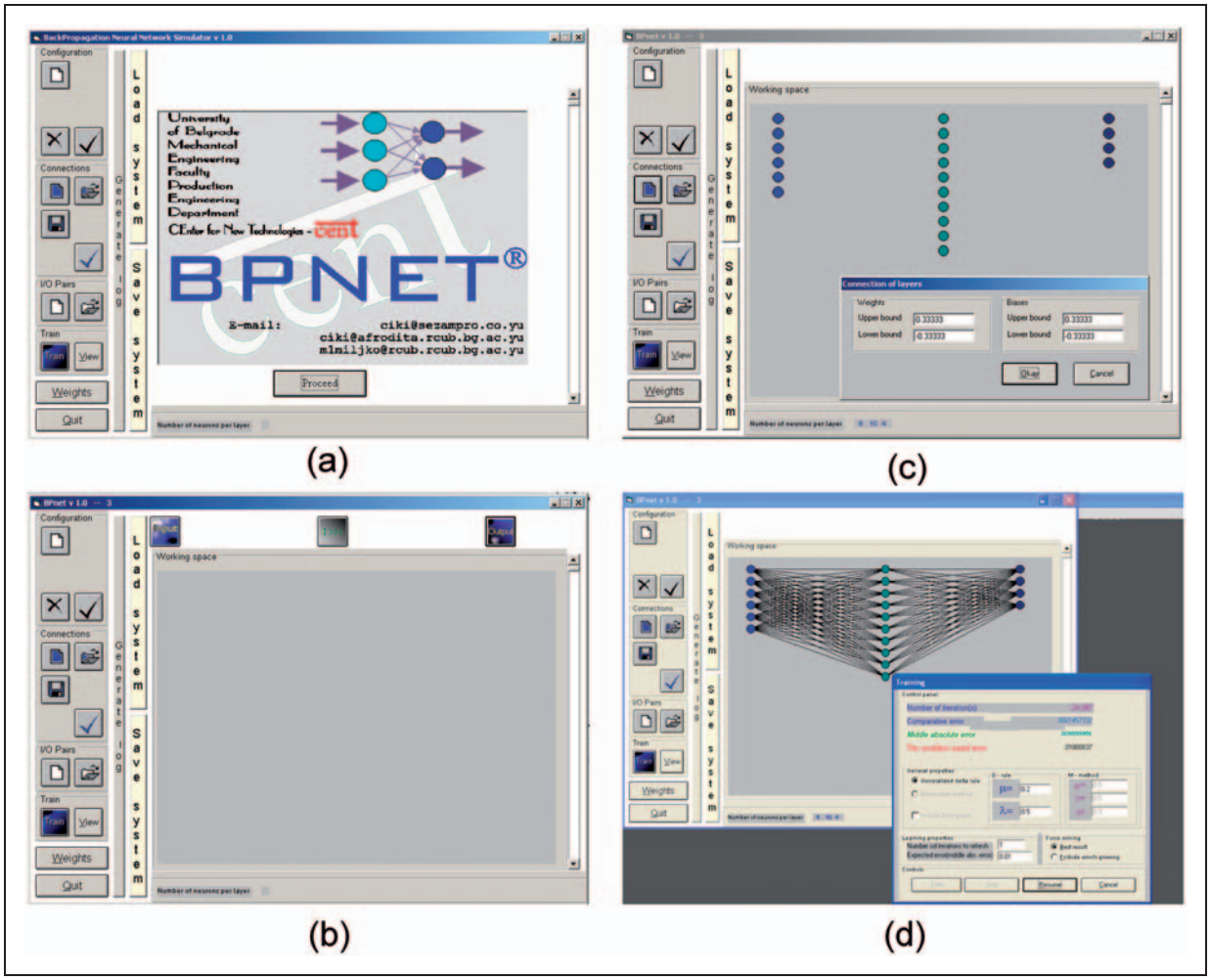

Figure 2. BPnet software: (a) starting window, (b) 'configuration' module, (c) 'connections' module and (d) training process.

training and testing set are much smaller in order to significantly reduce the computational cost and to speed up the NN training. In total, we use 64 randomly chosen sensor measurements, divided into the training and testing data in the same ratio. The bad data are implemented here in the same manner as before.

Before the start of the training phase, the data are scaled in the [–1 1] interval. This is done so as to enhance the network performance and to speed up the learning process. The following equation is used in this purpose[15]

$$x_{\text{scaled}} = \bar{x}_{\min} + \frac{x - x_{\min}}{x_{\max} - x_{\min}} \cdot (\bar{x}_{\max} - \bar{x}_{\min}), \qquad (1)$$

where x_{scaled} denotes scaled data value, $\bar{x}_{\min}$ and $\bar{x}_{\max}$ are minimum and maximum values in chosen range (i.e. –1 and 1, respectively) and $x_{\min}$ and $x_{\max}$ represent minimum and maximum values in the scaling dataset, and x is value to be scaled.

In this study, multilayer feedforward NNs with different learning algorithms are used. The tested architectures include one, two, three or four hidden layers. The NN input parameters are six scaled values (i.e. one sample) of forces and torques from the dataset. The output neurons represent four possible cases that correspond to the particular input: normal, collision, obstruction or front collision. Therefore, the prediction problem is formulated as a classification problem (similarly to Refs. 13, 17), which is solved by developing a novel approach using NNs. In Matlab implementation, sigmoid and linear activation functions are used in hidden and output layer, respectively. The leaning parameter is set to be 0.5 for all networks. In BPnet software, sigmoid function was used as an activation function with delta rule as the learning rule and with momentum $\lambda = 0.2$ and learning parameter $\mu = 0.2$. An example of a NN with five neurons in first and two neurons in second hidden layer is shown in Figure 3.

To obtain the optimal NN, we need to test various architectures. So far, there is no explicitly determined rule or pattern for selection of the number of the hidden layers reported in the literature. Likewise, the selection of number of neurons is not universally determined. This is usually done empirically, although there are different advices for solving this problem.[15] In this study 24 different architectures were investigated, including the networks with one, two or three hidden layers. The network structure marked as

$8-4-2$ means that there are eight neurons in the first hidden layer, four in the second hidden layer and two in the third hidden layer. As mentioned, the NN input and output are single column vectors since they represent scaled values of recorded sensor measurements and corresponding robot situations. Employed network architectures are listed in Table 1.

After determining the architectures listed in Table 1, several learning algorithms are employed in order to investigate the best possible NN behaviour. The specific problem under consideration represents the main problem in algorithm selection (i.e. the problem mainly influences the performance of the learning algorithms). Thus, the same algorithm can have different performance depending on the considered task. Therefore, we tested in Matlab all the main algorithms that proved to have best performance over classification, pattern recognition or non-linear function approximation in order to find optimal solution for the problem of robot failure prediction. Likewise, we tested one of the most popular gradient descent algorithm outside of Matlab so as to discover the best NN-based prediction method. These seven algorithms used in Matlab or BPnet environment with corresponding acronyms are listed in Table 2. Note that these acronyms and NN ordinal numbers will be used in the next section. As stated before, in Matlab we use sigmoid and linear activation function in the hidden and output layer, respectively. In BPnet software, sigmoid activation function is used in all layers.

The stopping criterion for NN training in Matlab software is defined in terms of goal MSE or maximum number of learning iterations. These values are defined to be 10^{-5} MSE and 1000(maximum iterations). In the case of BPnet, we set the goal MAE to be 0.01, which is the minimal value that can be

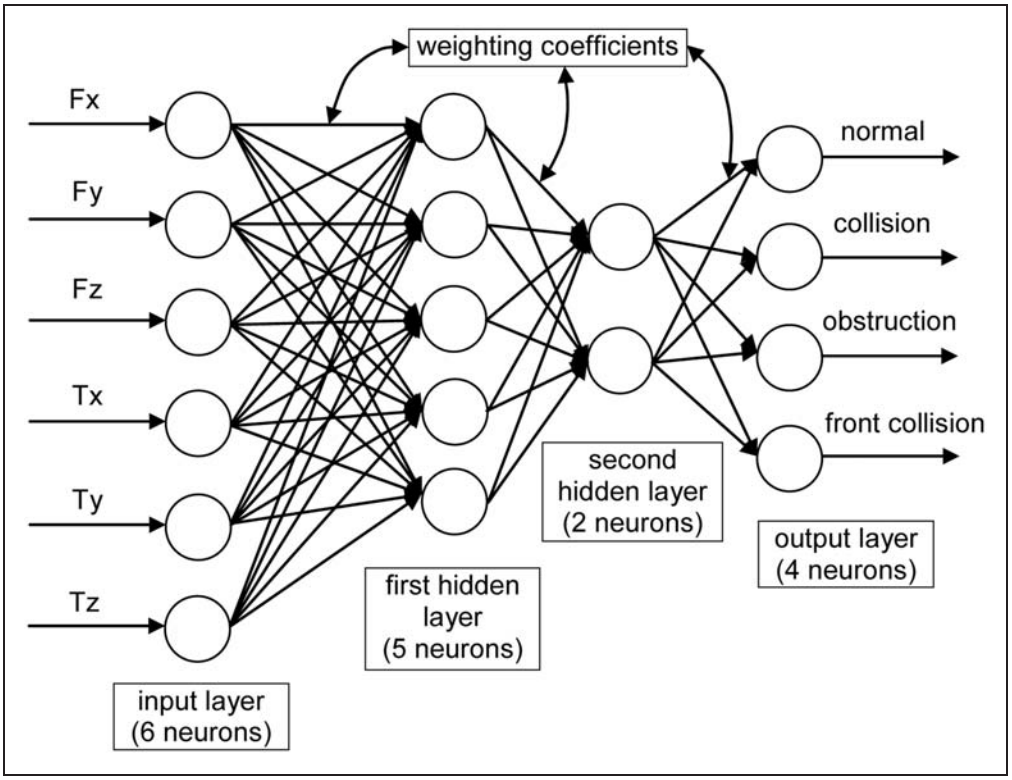

Figure 3. Structure of the proposed multilayer feedforward NN.

Table 1. NN architectures used in experiment.

No.	NN architecture	No.	NN architecture	No.	NN architecture	No.	NN architecture
1	1	7	1–1	13	2–2–2	19	3–3–3–3
2	2	8	2–2	14	3–2–2	20	4–3–3–3
3	3	9	3–2	15	4–3–2	21	5–4–3–3
4	5	10	5–2	16	5–3–2	22	8–5–4–3
5	8	11	8–4	17	8–3–2	23	10–8–4–3
6	10	12	10–4	18	8–4–2	24	10–8–5–4

defined in this software. Nevertheless, the experimental results show that this difference in MSE has no crucial influence in the overall prediction outcome.

The NN prediction performance is evaluated using the MSE (equation (2)) and MAE (equation (3)) on test data in Matlab and BPnet, respectively. The NN ability to predict execution failures is tested several times for each architecture and learning algorithm. The best results are obtained and presented in the following section.

$$MSE\_test = \frac{1}{n}\sum_{i=1}^{n}(\text{NN\_output} - \text{target\_output})^2 \tag{2}$$

$$MAE\_test = \frac{1}{n}\sum_{i=1}^{n}\left|\text{NN\_output} - \text{target\_output}\right| \tag{3}$$

Table 2. Learning algorithms used in the experiment.

No.	Learning algorithm	Software	Acronym
1	Levenberg–Marquardt	Matlab	LM
2	Bayesian regularization	Matlab	BR
3	Resilient backpropagation	Matlab	RP
4	Scaled conjugate gradient	Matlab	SCG
5	BFGS quasi–Newton backpropagation	Matlab	BFG
6	Variable learning rate backpropagation	Matlab	GDX
7	Gradient descent backpropagation	BPnet	BP

Results and discussion

Prediction of failures using recorded forces and torques

The verification of NN prediction performance is conducted using Intel® Core™2 Duo 2.1 GHz processor laptop computer with 2.96 GB RAM on Windows XP platform. The Matlab 2009a (v. 7.8.0.347) is employed for algorithm implementation and testing. In order to find optimal NN, all architecture and algorithms are tested several times, and the best performance is presented in this study. Furthermore, the dataset is corrupted with erroneous values in random manner (these values are 0–3% in the entire set). Likewise, in order to test every NN structure in an appropriate way, we utilize six NN architectures separately for one, two, three and four layer networks (the total number of architectures is 24 as given in Table 1). As mentioned before, the prediction in this study is treated as the classification problem, as used in many studies.[13,17] Note that the acronyms listed in Table 2 are used to denote corresponding learning algorithm.

Results of MSE on test data (30% of the dataset) for Levenberg–Marquardt (LM) and Bayesian regularization (BR) algorithm are depicted in Figure 4(a) and (b), respectively. The NN architectures in the figures represent network number given in Table 1. It is obvious that the MSE for LM has the decreasing trend when number of neurons and layers increases. In other words, the larger number of neurons and layers has positive influence on the training process. The best MSE result for LM algorithm is recorded for network number 23 (see Table 1) and is 0.0023. In the case of BR algorithm the similar conclusion can

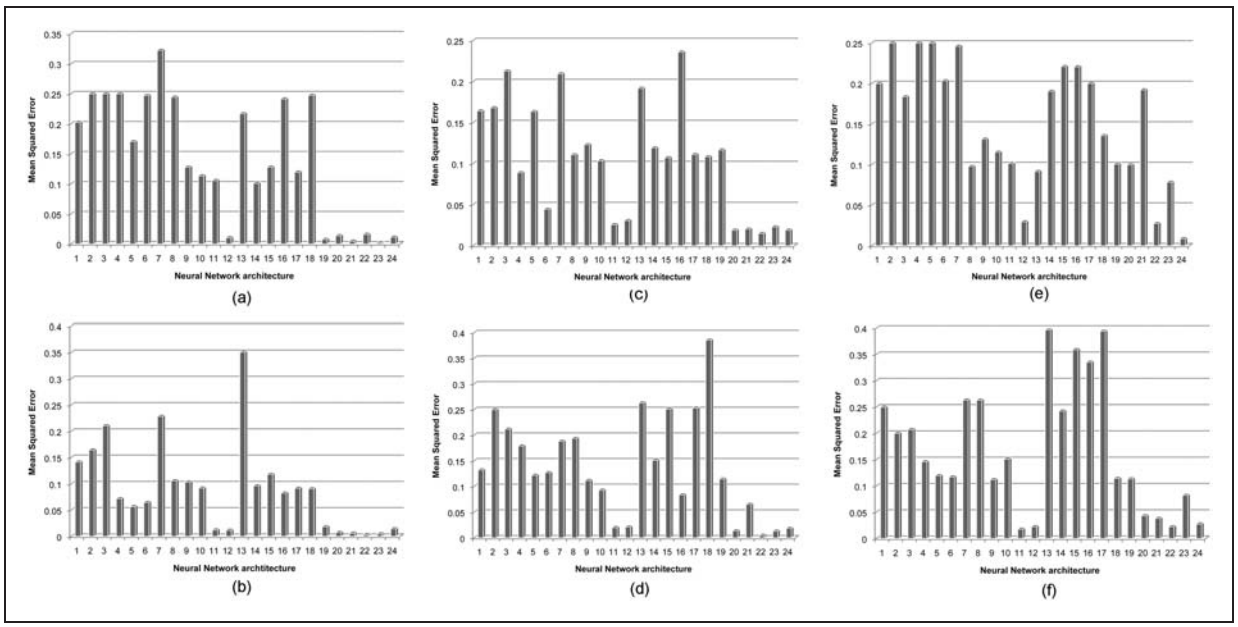

Figure 4. NN testing results: (a) LM algorithm, (b) BR algorithm, (c) RP algorithm, (d) SCG algorithm, (e) BFG algorithm and (f) GDX algorithm.

be obtained. Overall, the NNs with four hidden layers show the best performances. Particularly, in the case of BR algorithm, the network number 22 (Table 1) has the smallest test MSE of 0.0036.

Figure 4(c) and (d) indicates MSE results for resilient backpropagation (RP) and scaled conjugate gradient (SCG) algorithms, respectively. As in the previous case, the best results are for NNs with four hidden layers: For RP algorithm the smallest MSE is 0.0151 (for NN number 22, Figure 4(c)), while the SCG has the best MSE of 0.005(also NN 22, as shown in Figure 4(d)). Likewise, the two NNs that give good performance are $8-4$ and $10-4$. Similarly to the result of LM and BR algorithm, these networks under numbers 11 ($8-4$NN) and 12 ($10-4$ NN) show smallest errors among two layer networks. In other words, these NNs also show results that are promising for prediction purposes.

Finally, the MSE test results for BFGS quasi–Newton backpropagation (BFG) and variable learning rate backpropagation (GDX) algorithms are shown in Figure 4(e) and (f), respectively. In BFG case, the optimal network is the one with the four hidden layers (NN number 24), while the smallest MSE for GDX algorithm is NN with two hidden layers (NN number 11). Unlike previous cases, the NNs with the BFG algorithm do not exhibit overall best results with four hidden layers. In Figure 4(f), the NNs with two hidden layers show best GDX algorithm performance. However, the MSE results presented in this section only indicate the optimal NN outcome, since the prediction is determined based on the largest value in the network output.

After the testing phase, the prediction rate was calculated for each learning class individually. Additionally, the average rate for all algorithms and all NN architectures is determined. Since the NN inputs are randomly generated, the prediction performance over the entire testing dataset is presented. The results obtained in Matlab environment are given in Table 3.

Looking at Table 3 one can notice several important conclusions (best results are underlined and circulated). Firstly, some results reported in Figure 4 do not correspond to the results given in Table 3. For example, the second largest MSE for BR algorithm (Figure 4(b)) is found for network number 7 (architecture 1–1), while the corresponding prediction rate in Table 3 is 56.0606 which is fourth lowest result for that algorithm. This is because we evaluate prediction

Table 3. NN prediction rate in Matlab (in percentage).

NN architecture	LM	BR	RP	SCG	BFG	GDX	Average rate (architecture)
1	42.9293	46.2121	44.4444	51.2626	19.9495	23.2323	38.005
2	27.5253	40.1515	50	37.6263	24.2424	52.7677	38.7189
3	35.6061	37.8788	16.1616	45.9596	38.1313	16.6667	31.734
5	23.9899	72.7273	71.4646	37.8788	24.4949	52.5253	47.1801
8	42.1717	78.5354	37.1212	56.5657	21.2121	63.1313	49.7896
10	23.7374	72.2222	83.0808	63.1313	31.5657	57.0707	55.1347
1–1	51.2626	56.0606	53.2828	53.7879	52.2727	57.5758	54.0404
2–2	57.5758	69.4444	71.2121	54.0404	73.7374	60.3535	64.3939
3–2	67.9293	76.5152	73.7374	66.9192	70.2020	87.1212	73.7374
5–2	70.7071	73.4848	72.9798	69.4444	69.4444	69.4444	70.9175
8–4	92.4242	92.9293	92.6768	93.1818	72.2222	92.1717	89.2677
10–4	91.9192	91.9192	91.4141	93.1818	91.9192	94.1919	92.4242
2–2–2	55.3030	35.8586	56.0606	55.3030	73.9899	36.1111	52.1044
3–2–2	89.8990	74.7475	69.1919	68.9394	53.7879	55.3030	68.6448
4–3–2	73.2323	71.7172	73.2323	54.7980	58.8384	37.1212	61.4899
5–3–2	56.0606	75	56.3131	77.0202	56.0606	33.8384	59.0488
8–3–2	71.7172	79.0404	70.2020	56.8182	49.2424	33.8384	60.1431
8–4–2	56.0606	77.2727	76.7677	42.4242	71.4646	70.2020	65.6986
3–3–3–3	89.6465	89.8990	69.1919	72.2222	67.6768	69.4444	76.3468
4–3–3–3	90.6566	92.1717	91.1616	91.1616	71.4646	85.6061	87.037
5–4–3–3	94.6970	91.1616	91.6667	78.2828	53.7879	86.6162	82.702
8–5–4–3	93.6869	94.4444	92.6768	92.6768	93.4343	89.3939	92.7189
10–8–4–3	94.4444	94.4444	91.1616	91.6667	72.9798	75.2525	86.6582
10–8–5–4	94.1919	95.4545	93.6869	92.1717	93.9394	89.3939	93.1397
Average rate (algorithm)	68.9509	74.1372	70.3704	66.5194	58.5858	62.0156	/

based on the largest network output, i.e. we assume that the NN predicted correctly if the node that gives the largest output corresponds to the neuron with the target value 1. Nevertheless, this is not significant, since the NNs with the smallest MSE show the best prediction rate for each learning algorithm (Table 3). Secondly, despite the aforementioned, in most cases obtained MSE corresponds to the algorithm prediction rate. The evident MSE increasing or decreasing trend reported is also found in the prediction table. Thirdly, the results confirmed that the NN can successfully predict robot execution failures. The highest prediction rate of 95.4545 was found for the network number 24 (10–8–5–4), which is better than the results obtained with the base-level and meta-level classifier reported in Ref. 14. In spite of added erroneous data, the NN BR method outperforms the Naïve Bayes, Decision Trees and Support Vector Machine based algorithms.[14] These results evidence the applicability and the effectiveness of the NN in the case of failure prediction related task.

Finally, the results of BPnet engagement are shown in Figures 5 and 6. The same network architectures listed in Table 1 are considered in this case. MAE results for each architecture are shown in Figure 5. Note that the error is similar for most of the tested NN. The best result is obtained for the same NN architecture as in Matlab software – MAE for 10–8–5–4 network is 0.009962. The worst result shows NN with architecture 1–1. In order to validate the prediction ability, network with smallest MAE is tested next. Figure 6 shows testing input and output values. It can be observed that the trained NN successfully predicts the 'normal' case from the scaled input forces and torques. In other words, the generated output vector in Figure 8 corresponds to the randomly selected sensor measurements from the testing dataset. Generally, the BP algorithm performs well – overall, the prediction rate for all networks is 70.8333%. The best prediction rate refers to the network number 24 (10–8–5–4) – approximately 75%, while the worst NN (1–1) successfully predicted 51% of all considered cases. These results indicate that the BP, as well as other tested algorithms, can successfully be applied for robot execution failure prediction. Furthermore, as in the previous cases, the NN shows robustness regarding implemented erroneous values of forces and torques in the training/testing dataset.

Future research on this topic may include several directions. Further testing using different NN types and additional data is desirable in order to further improve prediction result. Radial basis function NN, recurrent NN and/or echo state NN could be investigated for the particular prediction purposes. Likewise, the information other than failures in approach to grasp position (as used in this study) could be added to the training/testing dataset.

Real-world experiments

In order to verify the proposed approach in a fair manner, several experiments in real time are conducted on a non-holonomic mobile robot in laboratory model of manufacturing environment. The robotic setup consists of *KheperaII* mobile robot

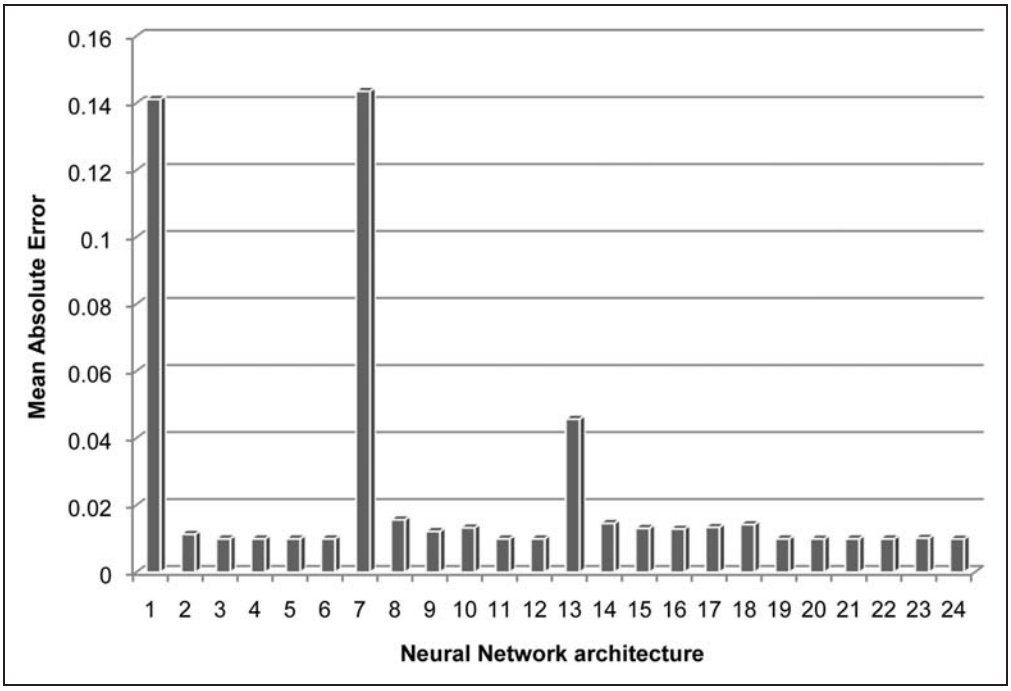

Figure 5. BPnet training results.

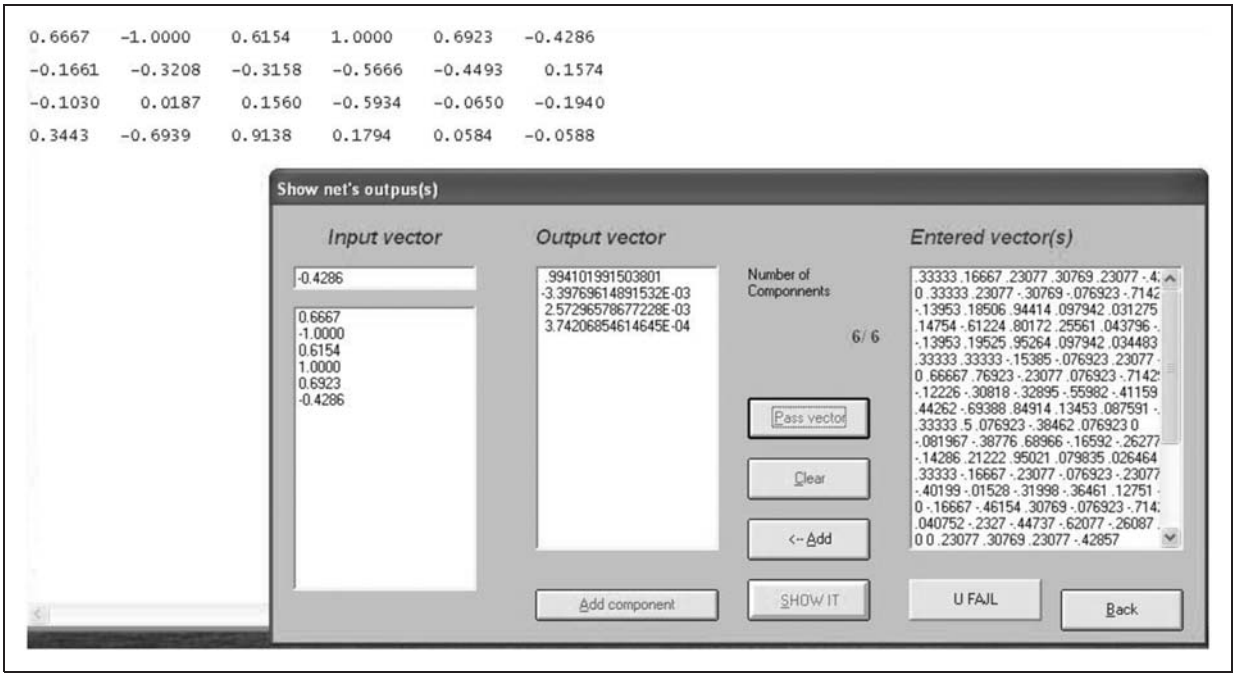

Figure 6. NN testing in BPnet software.

and six integrated infrared sensors. The data manipulation and robot control are carried out using Intel™ i5-2320 3 GHz processor desktop computer with 4 GB RAM on Windows 7. Technical description of a mobile robot used in experiments is given in Table 4.

First experiment refers to an obstacle detection problem.[35] Six infrared sensors mounted directly on the front end of the robot are used to detect obstacle in two characteristic positions – on the left side and on the right side of the robot. The NN training data are gathered by placing the platform in several positions for each case (Figure 7). In Figure 7 one can notice that the robot is placed closer and further away from the object in different positions for both cases. This is done in order to show robustness of our approach regarding various combinations of obtained sensor measurements. Additionally, the failed data are gathered in two ways: (i) manually, by blocking the chosen infrared sensor(s), and (ii) in control commands, by replacing correct values with the failed ones. All these information are implemented in the training and testing set for NN divided in a ratio of 7 : 3 (respectively), as in the previous case with the recorded forces and torques.

Since the earlier experiments showed that BR algorithm (Table 2) is the most suitable for this prediction problem, this one is used in all real-world experiments. Also, we tested the overall most successful architecture (see Table 3), i.e. the network number 24 in Table 1 (10–8–5–4). The activation functions are the same as earlier – sigmoid in hidden and linear in output layer. The NN input represents six infrared sensor measurements with values from 0 (obstacle is far) to 1020 (obstacle is near), and the output is a value 1 if the failure is recognized, and 0 otherwise. These values are scaled in accordance with equation (1). Similar to the previous cases, the failure prediction problem is formulated as a classification problem[13,17] using sensor information. An example of correct and incorrect sensor measurements for both cases is represented in Table 5, in which the 'X' mark denotes failed sensor measurement. The first two measured values correspond to the sensor positions on the left side of the robot, while the last two values correspond to the sensors placed on the right side of the platform. In accordance to the aforementioned, one can conclude that the larger sensor values indicate that the object is closer to the robot (Table 5).

The incorrect sensor values are included in the training set in random manner, meaning that the number and the index location of the failed

Table 4. Robot configuration specifications.

Robot elements	Technical description
Processor:	Motorola 68331, 25 MHz
RAM:	512 kB
Flash:	512 kB
Motion:	2 DC servo motors with incremental encoders
Speed:	Max. 0.5 m/s; Min. 0.02 m/s
Power:	Power adapter or rechargeable NiMH batteries
Communication:	Standard serial port, up to 115 kbps
Size:	Diameter: 0.07 m; Height: 0.03 m
Payload:	Approx. 250 g

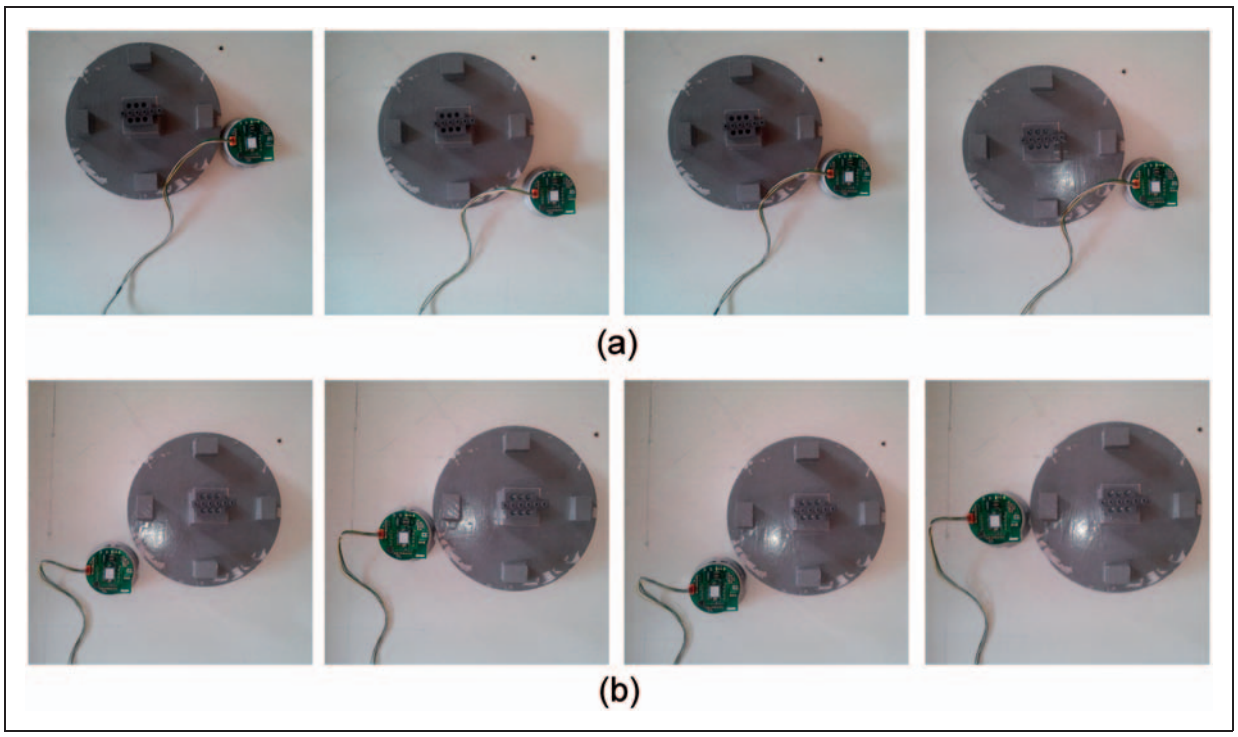

Figure 7. Robot positions in the detection experiment: (a) obstacle on the left side, (b) obstacle on the right side.

Table 5. Example of correct and failed sensor information.

	Correct infrared values	Failed (incorrect) infrared values
Obstacle on the left side	$IR = [420 \quad 704 \quad 92 \quad 68 \quad 56 \quad 50]^T$	$IR = [X \quad 554 \quad 52 \quad 64 \quad 56 \quad 95]^T$
Obstacle on the right side	$IR = [72 \quad 68 \quad 48 \quad 100 \quad 984 \quad 1020]^T$	$IR = [55 \quad X \quad 63 \quad 105 \quad X \quad 921]^T$

measurement are randomly generated. We tested the NN algorithm and selected architecture several times for each detection problem with up to three failed values included in the input vector. The result showed that in over 96% of all tested cases NN successfully recognized the failed sensor measurement. In addition to this, incorrect measurement(s) are replaced with the expected value(s), so that the object location (left or right of the robot) is successfully recognized.

Second experiment is related to the trajectory tracking problem.[36] As mentioned earlier, one can consider the case in which the robot wheels command unit is not working properly, i.e. the situation in which several control commands have unwanted values regarding tracking the particular trajectory. If the wheel command in every control iteration is not as expected (calculated), the mobile robot could make a significant error or even completely mistrack the desired trajectory. To prevent this from happening, in this study we investigated NN behaviour in the domain of failure prediction in the case of tracking two types of trajectories.

An example of good/bad control commands for tracking of *M-shaped* trajectory is given in Table 6 (1 unit corresponds to the wheel motion of 1/12 mm). It is noticeable that each control value must be as closer to the desired one in order to successfully track this kind of trajectory. Obviously, in the case of employed incorrect wheel commands (third column in Table 6), the tracking problem is not successfully solved. Second studied trajectory is more complicated *labyrinth-type* trajectory, with good and bad wheel signals given in Table 6. In this case, the tracking error is even more evident: The tracked trajectory is completely different unless every control value matches the desired information. As in the previous experiment, failed data are incorporated in random manner (number of failures and index locations) in the training set.

An implemented NN here is the same as in the obstacle detection experiment: BR algorithm with 10–8–4–2 architecture, and with sigmoid and linear activation functions. Input to the network is scaled to the right and left wheel commands, while the output represents successful or incorrect failure prediction. We tested each trajectory several times, and the results of one test is presented in Figure 8 (the red line denotes the robot orientation in every control iteration). The experiments confirmed the usefulness

Table 6. Example of correct and failed wheel commands.

	Correct wheel commands	Failed (incorrect) wheel commands
M-shaped trajectory	$Left\_wheel = \begin{bmatrix} -150 & 200 & 150 \\ 200 & -150 & 200 \\ 150 & 200 & -150 \end{bmatrix}$	$Left\_wheel = \begin{bmatrix} X & 200 & 150 \\ 200 & -150 & 200 \\ 150 & X & -150 \end{bmatrix}$
	$Right\_wheel = \begin{bmatrix} 150 & 200 & -150 \\ 200 & 150 & 200 \\ -150 & 200 & 150 \end{bmatrix}$	$Right\_wheel = \begin{bmatrix} 150 & X & -150 \\ X & 150 & 200 \\ X & 200 & 150 \end{bmatrix}$
Labyrinth-type trajectory	$Left\_wheel = \begin{bmatrix} 200 & -124 & 200 \\ -124 & 150 & -124 \\ 150 & -124 & 100 \\ -124 & 100 & -124 \\ 20 & 20 & 10 \end{bmatrix}$	$Left\_wheel = \begin{bmatrix} 200 & X & 200 \\ -124 & 150 & -124 \\ X & -124 & 100 \\ X & X & -124 \\ 20 & 20 & 10 \end{bmatrix}$
	$Right\_wheel = \begin{bmatrix} 200 & 124 & 200 \\ 124 & 150 & 124 \\ 150 & 124 & 100 \\ 124 & 100 & 124 \\ 20 & 20 & 10 \end{bmatrix}$	$Right\_wheel = \begin{bmatrix} 200 & X & 200 \\ 124 & 150 & 124 \\ 150 & 124 & X \\ 124 & 100 & 124 \\ X & 20 & 10 \end{bmatrix}$

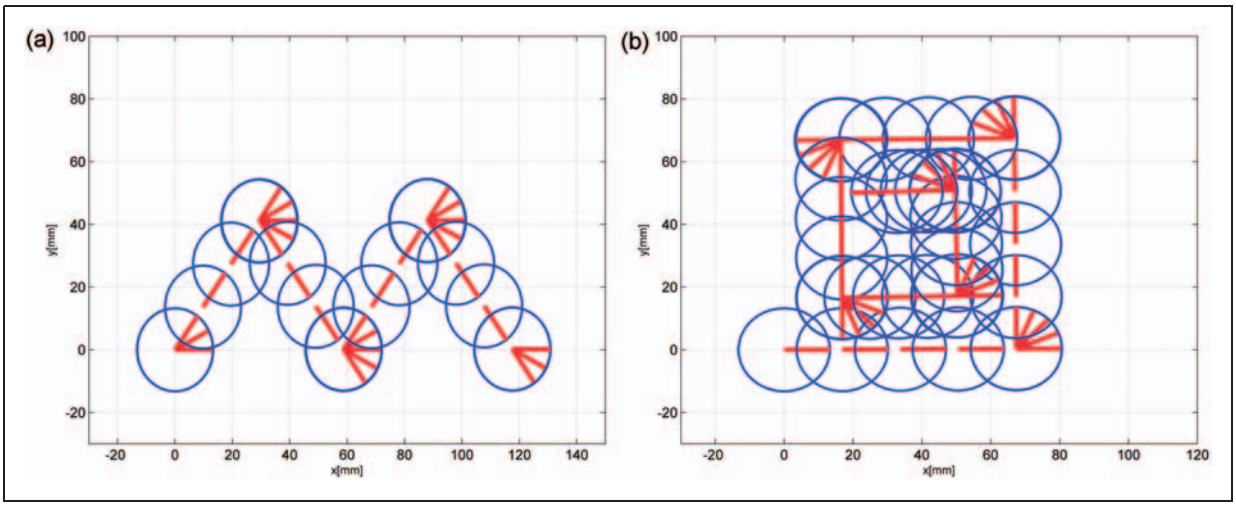

Figure 8. Results of trajectory tracking experiments: (a) *M-shaped* trajectory, (b) *Labyrinth-type* trajectory.

of the proposed approach: In more than 99% of the cases, the network successfully predicted the failure prediction. In addition, in order to successfully track chosen trajectory, failed values are replaced with the desired information. The result shows that the mobile robot is able to track each trajectory and that robot poses do not significantly differ from the wanted ones in every time instant.

Finally, Figure 9 denotes robot poses in characteristic iterations during tracking of *labyrinth-type* trajectory (see also Figure 8(b)). Starting from an arbitrarily pose in indoor environment, Figure 9 shows robot poses at the beginning and at the end of each straight-line motion (for example, Figure 9(a) and (b), and also Figure 9(d) and (e)), and the pose during the rotation (for example, Figure 9(c) and (f)). It is obvious that, using the NN-based prediction method, mobile robot successfully tracks the complex trajectory. The minimal error in the final robot pose evidences the usefulness and the robustness of the proposed approach.

Conclusions

In this paper, a novel approach for robot failure prediction based on the NNs is presented. A successful

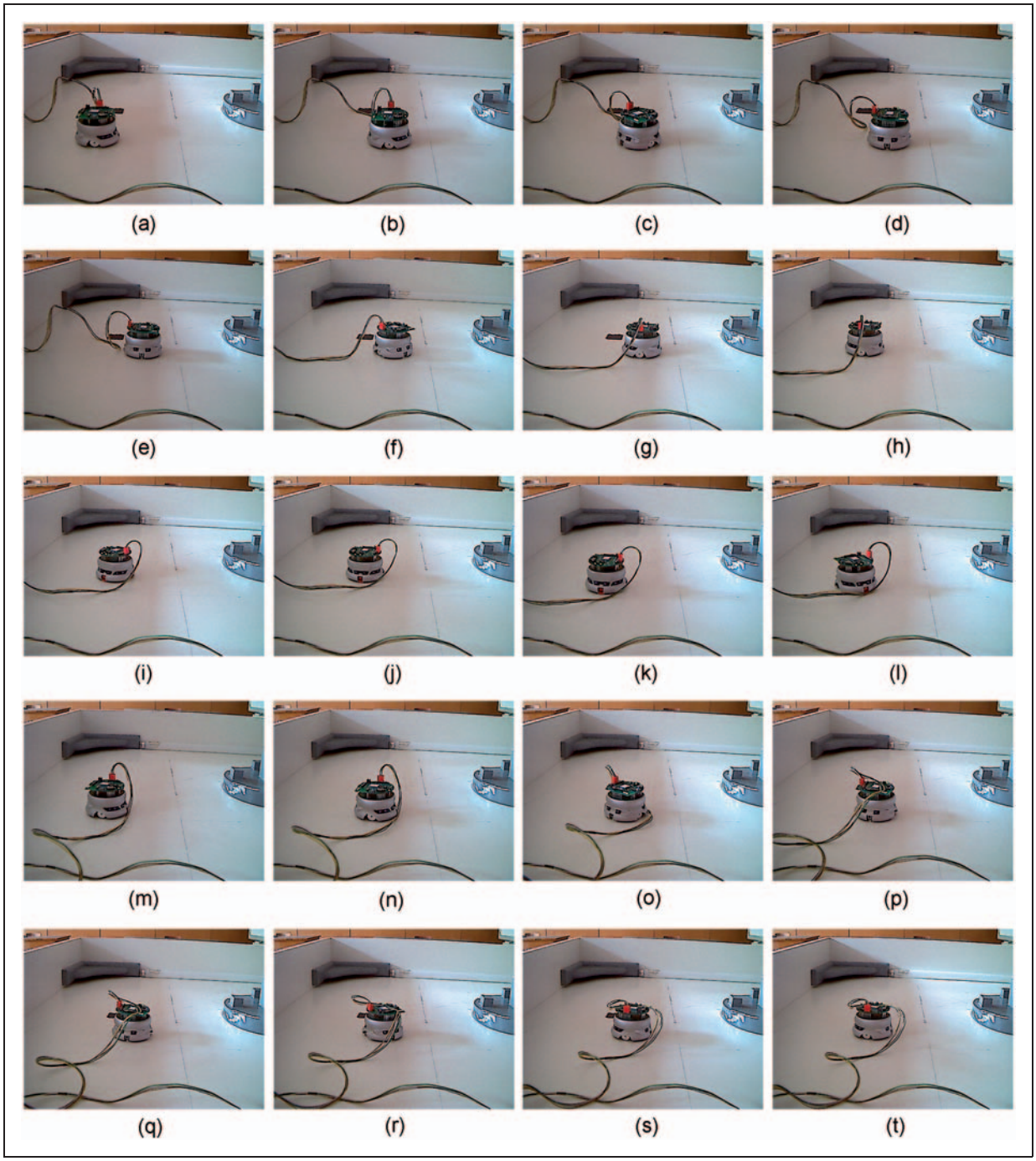

Figure 9. Mobile robot tracking the labyrinth-type trajectory: (a)–(t) Robot poses in characteristic control iterations.

mapping from the execution forces and torques to the four possible cases that correspond to the particular input (normal, collision, obstruction or front collision) is developed using NNs. The training dataset consists of a real robot data that are recorded immediately after the system failure during the execution of the specific task. In order to fully show the robustness of the proposed approach, several real-world experiments for obstacle detection and trajectory tracking are conducted on a *KheperaII* mobile robot.

Two software tools are used for the prediction purposes in this study – Matlab and BPnet. In total, 6 algorithms and 24 neural architectures are tested in Matlab environment so as to find optimal solution. Each multilayer feedforward NN with different learning algorithm and architecture that consists of one, two, three or four hidden layers is evaluated several times and the best results are presented and explained in details. In addition to the Matlab testing, specially designed software titled BPnet which employs most common feedback method for error minimization is used. In this software, the same NN architectures are trained using gradient descent BP algorithm. The experimental results show that the NN outperforms the Naïve Bayes, Decision Trees and Support Vector Machine based algorithms[14] employed for the

prediction of robot execution failures. The best prediction rate of 95.4545% proves the effectiveness of the NN engagement for this problem.

Finally, the experiments conducted in indoor environment present usefulness of the proposed approach. Six infrared sensors mounted on the mobile robot are used to obtain information of the obstacle located left and right from the platform. Randomly generated failed sensor data are integrated into the training set so as to test the NN performance in this task. The result shows that in over 96% of all tested cases NN recognized failed value, meaning that the obstacle location is successfully determined after the failed information is replaced with the expected one. Likewise, the tracking of the *M-shaped* and *labyrinth-type* trajectories showed as a fairly easy task for the developed prediction method. In more than 99% of the cases, the network predicted the wheel command failure, which is next replaced with the desired value in order to successfully track chosen trajectory. The experiments show that a mobile robot can track desired trajectories with a minimal error in every control iteration, which evidences the robustness and the applicability of the proposed approach.

Funding

This work is partially supported by the Serbian Government – the Ministry of Education, Science and Technological Development – through the project TR35004 (2011–2014).

Acknowledgement

The authors gratefully acknowledge the significant effort of anonymous reviewers whose suggestions substantially improved an earlier version of this paper.

References

1. Dixon W, Walker I, Dawson D, et al. Fault detection for robot manipulators with parametric uncertainty: A prediction-error-based approach. *IEEE Trans Robot Autom* 2000; 16(6): 689–699.
2. Visinsky M, Cavallaro J and Walker I. Robotic fault detection and fault tolerance: A survey. *Reliab Eng Syst Safety* 1994; 46(2): 139–158.
3. Sreevijayan D, Tosunoglu S and Tesar D. Architectures for fault tolerant mechanical systems. In: *Proceedings of the IEEE mediterranean electrotechnical conference*, Antalya, Turkey, 12–14 April 1994, pp.1029–1033.
4. Vemuri A and Polycarpou MM. A dynamic fault tolerance framework for remote robots. *IEEE Trans Robot Autom* 1995; 11(4): 477–490.
5. Shin JH and Lee JJ. Fault detection and robust fault recovery control for robot manipulators with actuator failures. In: *Proceedings of the IEEE international conference on robotics and automation*, Detroit, USA, 10–15 May 1999, pp.861–866.
6. Brambilla D, Capisani LM, Ferrara A, et al. Fault detection for robot manipulators via second-order sliding modes. *IEEE Trans Ind Electron* 2008; 55(11): 3954–3963.
7. Halder B and Sarkar N. Robust fault detection of a robotic manipulator. *Int J Robot Res* 2007; 26(3): 273–285.
8. Muradore R and Fiorini P. A PLS-based statistical approach for fault detection and isolation of robotic manipulators. *IEEE Trans Ind Electron* 2012; 59(8): 3167–3175.
9. Dixon W, Walker I and Dawson D. Fault detection for wheeled mobile robots with parametric uncertainty. In: *Proceedings of the IEEE/ASME international conference on advanced intelligent mechatronics,* Como, Italy, 8–11 July 2001, pp.1245–1250.
10. Roumeliotis SI, Sukhatme GS and Bekey GA. Sensor fault detection and identification in a mobile robot. In: *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems*, Victoria, Canada, 13–17 October 1998, pp.1383–1388.
11. Hashimoto M, Kawashima H, Nakagami T, et al. Sensor fault detection and identification in dead-reckoning system of mobile robot: Interacting multiple model approach. In: *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems*, Maui, USA, 29 October–3 November 2001, pp.1321–1326.
12. Ming Y, Danwei W and Qijun C. Prediction of multiple failures for a mobile robot steering system. In: *Proceedings of the IEEE international symposium on industrial electronics*, Hangzhou, China, 28–31 May 2012, pp.1240–1245.
13. Twala B. Robot execution failure prediction using incomplete data. In: *Proceedings of the IEEE international conference on robotics and biomimetics*, Guilin, China, 19–23 December 2009, pp.1518–1523.
14. Koohi T, Mirzaie E and Tadaion G. Failure prediction using robot execution data. In: *Proceedings of the 5th symposium on advances in science and technology*, Mashhad, Iran, 12–17 May 2011.
15. Miljković Z and Aleksendrić D. *Artificial neural networks – solved examples with theoretical background (in Serbian)*. Belgrade: University of Belgrade – Faculty of Mechanical Engineering, 2009.
16. Lolasa S and Olatunbosun OA. Prediction of vehicle reliability performance using artificial neural networks. *Expert Syst Appl* 2008; 34(4): 2360–2369.
17. Miljković Z, Gerasimović M, Stanojević LJ, et al. Using artificial neural networks to predict professional movements of graduates. *Croat J Educ* 2011; 13(3): 117–141.
18. Miljković Z, Vuković N, Mitić M, et al. New hybrid vision-based control approach for automated guided vehicles. *Int J Adv Manuf Technol* 2013; 66(1–4): 231–249.
19. Parhi DR and Singh MK. Real-time navigational control of mobile robots using an artificial neural network. *Proc IMechE, Part C: J Mechanical Engineering Science* 2009; 223(7): 1713–1725.
20. Miljković Z, Mitić M, Lazarević M, et al. Neural network reinforcement learning for visual control of robot manipulators. *Expert Syst Appl* 2013; 40(5): 1721–1736.
21. Al-Assadi HMAA, Mat Isa AA, Hasan AT, et al. Development of a real-time-position prediction algorithm for under-actuated robot manipulator by using of artificial neural network. *Proc IMechE, Part C: J Mechanical Engineering Science* 2011; 225(8): 1991–1998.

22. Althoefer K, Lara B, Zweiri YH, et al. Automated failure classification for assembly with self-tapping threaded fastenings using artificial neural networks. *Proc IMechE, Part C: J Mechanical Engineering Science* 2008; 222(6): 1081–1095.
23. Bevilacqua M, Braglia M, Frosolini M, et al. Failure rate prediction with artificial neural networks. *J Qual Maint Eng* 2005; 11(3): 279–294.
24. Van M, Kang HJ and Ro YS. A robust fault detection and isolation scheme for robot manipulators based on neural networks. *Lecture Notes Comput Sci* 2011; 6838: 25–32.
25. Miljković Z. *Systems of artificial neural networks in production technologies (in Serbian)*. Belgrade: University of Belgrade – Faculty of Mechanical Engineering, 2003.
26. Rumelhart DE, Hinton GE and Williams RJ. Learning internal representations by error propagation in parallel distributed processing. In: DE Rumelhart and JL McClelland (eds) *Parallel distributed processing*. Cambridge, MA, USA: MIT Press, 1986, pp.318–362.
27. Blake CL and Merz CJ. UCI Repository of Machine Learning databases, http://www.ics.uci.edu/~mlearn/MLRepository.html (1987, accessed 11 November 2012).
28. Schalkoff RJ. *Artificial neural networks*. New York: McGraw-Hill, 1997.
29. Adriaans P and Zantinge D. *Data mining*. New York: Addison-Wesley, 1997.
30. Wong BK, Bonovich TA and Selvi Y. Neural network applications in business: A review and analysis of the literature (1988–95). *Decis Support Syst* 1997; 19(4): 301–320.
31. Vuković N and Miljković Z. A growing and pruning sequential learning algorithm of hyper basis function neural network for function approximation. *Neural Networks* 2013; 46C: 210–226.
32. Liao SH and Wen CH. Artificial neural networks classification and clustering of methodologies and applications – literature analysis form 1995 to 2005. *Expert Syst Appl* 2007; 32(1): 1–11.
33. Miljković Z and Babić B. Empirical control strategy for learning industrial robot. *FME Trans* 2007; 35(1): 1–8.
34. Visual Basic 5.0. Enterprise edition. *Microsoft Corporation*, 1997.
35. Kube CR. A minimal infrared obstacle detection scheme. Robot Practitioner: *J Robot Builders* 1996; 2(2): 15–20.
36. Siegwart R, Nourbakhsh I and Scaramuzza D. *Introduction to autonomous mobile robots*. 2nd ed. Cambridge, MA, USA: MIT Press, 2011.