`

# SUPERVISED NEURAL NETWORK CONTROL OF REAL-TIME TWO WHEEL INVERTED PENDULUM

**Hanan Nabil**
Teaching Assistant
Minia University, Faculty of Engineering
hananazowz@mu.edu.eg

## 1.Abstract

This research paper investigates an intelligent control technique stabilizing a real-time model ofthe two-wheel inverted pendulum. The TWIP model is a highly non-linear, open-loop, and unstable system which makes control a challenge. Initially,a state-feedback controller that uses the dynamical system states and control signals to construct the precise control decision is used to stabilize the system. Later, Supervised Feed-Forward Neural Networks (SFFNN) based on back propagation Levenberg-Marquardt optimization algorithm are trained by using real-time measurements of system states and motors control signals from state-feedback controller stabilization. SFFNN control the two-wheel inverted pendulum better than a state-feedback controller.

***Keywords*—**TWIP, State-feedback controller, Supervised Feed-forward neural network, Levenberg-Marquardt Optimization Algorithm

## 2. Introduction

The research on inverted pendulum has acquired momentum over the last decade in a lot of global control laboratories [1-4]. The inverted pendulum is regarded as a famous and exclusive benchmark for many researchers in both fields of control and robotics [5, 6]. Researchers employ inverted pendulum for testing control theories and algorithms such as (PID controller, neural networks, fuzzy control, and genetic algorithms)[7]. As for TWIP, it is considered to be an under-actuated mechanical system that is open-loop unstable with highly nonlinear dynamics. This property is considered a challenging control problem that has made researchers interested in solving this problem over recent years [8, 9]. It has become essential to investigate the possibilities of implementing a control system to keep the system in equilibrium. The basics of the control system are the two-wheel inverted pendulum balancing at certain set point by applying a suitable control signal (PWM_L and PWM_R) to the model actuators (left and right dc motors). The TWIP mobile robot is an extremely nonlinear, open-loop and unstable system. This indicates that standard linear techniques are unable to model the nonlinear dynamics of the system. When the model is simulated it falls over quickly. The characteristics of the TWIP model make control more interesting and challenging. The TWIP model is also known as a self-balancing mobile robot and this self-balancing bot is a developed version of a self-balancing platform, which makes use of data from accelerometer and gyroscope. It includes the essential signal processing part which uses Kalman filter and compensatory filter for estimating the correct angle out of noise or disturbance (a sudden hit). The system model used in this study consists of two geared DC motors with encoders, data acquisition card (DAQ (Arduino UNO)), and balancing shield holds the inertia measurement unit (IMU) which contains system sensors (accelerometer and gyroscope). Additionally, this shield contains a motor driver (L298N) and two wheels. To build a self-balancing robot, it is necessary to get to the bottom of the inverted pendulum problem or get to the bottom of an inverted pendulum on a cart. When the robot begins to tilt in one direction, the wheels should go in the

`

inclined or tilt direction with a speed relative to angle and acceleration of tilt to correct the inclination angle. When the divergence from the equilibrium point is small, the move should be slow or gentle and vice versa. The two-wheel inverted pendulum derived from the inverted pendulum model is developed or designed as a platform to investigate a Kalman filter used for sensor fusion[10]. A real-world application based on inverted pendulum system is the Segway vehicle that is designed to carry a human. Recently the commercial Segway vehicle was used to transport a robonaut in some NASA projects.

The purpose of this work is to investigate a neural network controller which determines and calculates the precise control action needed to stabilize the system and learn from experience how to regulate the wheels' position so that the inclination angle remains stable within a predetermined value[11, 12]. Research on neural-network based control systems has had great attention over the past several years. This is simply because of the neural networks approximation's ability for nonlinear function[13]. The artificial neural networks consist of several nonlinear elements and this gives them an advantage over linear techniques in modeling nonlinear systems. ANN is trained by adaptive learning, the network learns how to do tasks and perform functions for training purposes that are based on the data attached. The knowledge learned throughout the training process is stored in the synaptic weights. SFFNN is the used type of neural network in this study, used to stabilize the TWIP model. Training SFFNN requires an existing controller; training the neural network to emulate an existing controller requires a vector of inputs and outputs from the controller. The supervised control allows the neural network to be trained to imitate a robust controller like a state-feedback controller[14]. The robust controller operates perfectly if the process operates within a certain point or set point. The neuro-controller operates in a similar manner to the robust controller but can also adapt if any unexpected disturbance arises in the system. The developed neural networks are a two-layerFFNN with four neurons in input layer which are system states, variable number of nodes or neurons in the hidden layer and two neurons in the output layer which represent PWM_L and PWM_R of two geared DC motors. This is used to study the system and the back propagation based on Levenberg-Marquardt optimization algorithm is used to train and update the neural network weights. The hyperbolic tangent activation function was used in the hidden layer and in the output layer; it was the pure-line function. Figure (1) shows the over all TWIP system structure.

A lot ofresearchers present the controlling two-wheel inverted pendulum using artificial neural network controller. J. S. Noh, G. H. Lee, and S. Jung for instance, presented a position control of mobile inverted pendulum system by utilizing the radial basis function (RBF) and successfully controlled the pendulum angle and postion[15].S. Jung and S. S. Kim introduced and implemented control experiment using neural network control of a wheel-driven mobile inverted pendulum based on encoders and a gyro sensor[13].As for C. Yang, Z. Li, R. Cui, and B. Xu they suggested studying the wheel inverted pendulum as an underactuated system controlled by using neural network[16].C.-C. Tsai, H.-C. Huang, and S.-C. Lin studied controlling a self-balancing two-wheeled known as a scooter based on the adaptive neural network[17].Z. Li and C. Yang presented a study about controlling real-time application derived from wheeled inverted pendulum systems[18].C. Yang, Z. Li and J. Li also introduced the Trajectory planning and optimized adaptive control applied to a kind of wheeled vehicle models based on inverted pendulum [19]. Jung and S. Su Kim implementedan intelligent controller based onthe neural network with a field programmable gate array (FPGA) and a digital signal processing (DSP) board to get to the bottom of control problems ofthe nonlinear system, they successfully controlled pendulum angle and position[20].

This paper is arranged as follows: (1) An introduction about the two-wheel inverted pendulum system and the previous work in controlling TWIP using neural networks. (2) The mathematical equations describing the TWIP system which were developed based on Newtonian mechanics is presented in section "TWIP mathematical description." (3) A description of the controller type used for balancing the TWIP system before training neural network controllers is presented in the section "The state-feedback controller of the TWIP system." (4) The neural network optimization algorithms used to train the neural networks is presented in section "Neural network optimization algorithms." (5) Finally, the neural networks controllers applied to the model and the obtained results are presented in section "Results."

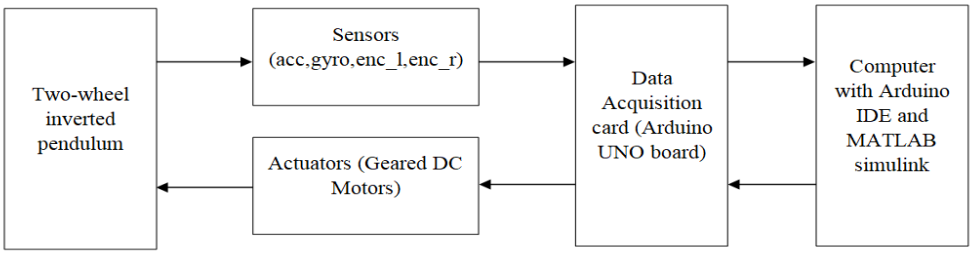


**Fig. 1** The complete diagram of two-wheel inverted pendulum mobile robot system

## 3. TWIP Mathematical Description

The two-wheel inverted pendulum can adapt the direction by altering the wheel velocity driven by DC motors. The control objective is to stabilize the model's equilibrium. An accelerometer and a gyroscope are used to estimate the correct angle of the model through a Kalman filter. Motor encoders are employed to count wheel rotations. The Newtonian method is used to explore the equations of motion of the TWIP model[21].Table(I) contains the numerical values of TWIP parameters. These parameters were previously estimated by us because the used system is a commercial type known as Balan Bot and there was a lack of information about the system. The Pattern search-Latin hypercube-an active set method is used to estimate these parameters. The nonlinear equations describing the TWIP systems are,

$$\ddot{x} = \frac{-2K_m K_e}{Rr^2\beta}\dot{x} - \frac{2K_m}{Rr\beta}V_a + \frac{M_p \ell \dot{\theta}^2}{\beta}\sin\theta - \frac{M_p \ell \cos\theta}{\beta}\ddot{\theta} \quad (1)$$

$$\ddot{\theta} = \frac{2K_m K_e}{Rr\alpha}\dot{x} - \frac{2K_m}{R\alpha}V_a - \frac{M_p g \ell \sin\theta}{\alpha} - \frac{M_p \ell \cos\theta}{\alpha}\ddot{x} \quad (2)$$

Where

$$\beta = 2M_w + \frac{2I_w}{r^2} + M_p$$

$$\alpha = I_p + M_p \ell^2$$

TABLE I
TWIP PARAMETERS

| Symbol | Quantity | Estimated values | Unit |
|---|---|---|---|
| $\mathbf{M_p}$ | Pendulum mass | 0.84719 | Kg |
| $\mathbf{M_w}$ | Wheel mass | 0.068563 | Kg |
| $\mathbf{I_w}$ | Wheel inertia | 0.99999 | N/m/sec |
| $\mathbf{I_p}$ | Body inertia | 0.035229 | $kg.m^2$ |
| $\boldsymbol{\ell}$ | COG distance | 0.059023 | m |
| **r** | Wheel radius | 028938 | m |
| **R** | Motor resistance | 4.3132 | Ohm |
| **g** | Acceleration constant | 9.81 | $m/s^2$ |
| $\mathbf{K_m}$ | Torque constant | 0.6073 | N.m/Amp |
| $\mathbf{K_e}$ | Voltage constant | 0.6073 | V/rad/sec |

## 4.The State-Feedback Controller of the TWIP System

State-feedback control was chosen in this work as a principal control technique since concurrent control of several degrees of freedom can only be ensured if they are all taken into consideration at the same time as shown in the figure below[22]. The controller inputs are the real-time readings of accelerometer, gyroscope, left motor encoder and right motor encoder (system states). These values are used to produce the controller output signal in forms of pulse width modulation that will be applied to geared dc motors (M1 and M2) through Arduino pins (pin 5 and pin 6).Readings of left and right encoders are used to estimate the position and velocity of the model through a written MATLAB function. The figure below illustrates the control flow of TWIP mobile robot, where x(t) are system states ( $\theta$, $\dot{\theta}$, x, $\dot{x}$) ,$T_{M1}$, $T_{M2}$are motors torques and PWM_L, PWM_R are the control signal applied on system actuators(motors) .

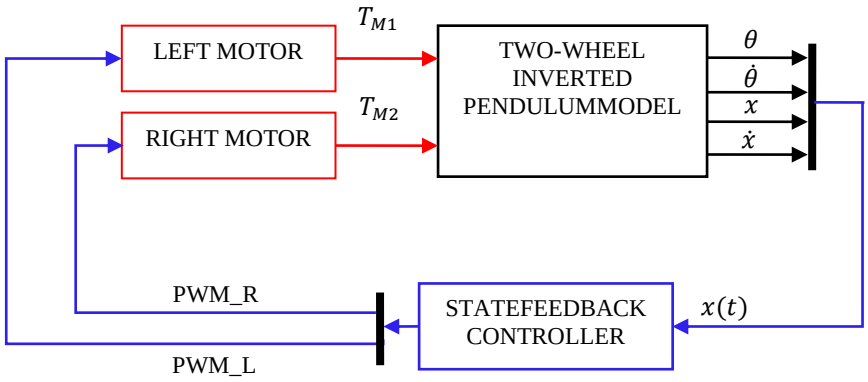


**Fig.2** Block diagram of the state-feedback control of the TWIP system

## 5. Neural Network Optimization Algorithms

Neural network architecture can be formed using two or more combined neurons to develop a multi-layer network[23]. Fig (3) represents an example of a multilayer architecture for a neural network[24]. The architecture of a neural networks consist of three layers, i.e., input layer which accepts system states or sensors real-time measurements, a hidden layer which is the intermediate layer of the network and output layer which is the model outputs or motors inputs. First of all, the input layer nodes are passive because they do not modify the data. The second layer is the hidden layer which processes and handles the data among the input and output layers of the network to enhance a behavioral representation of the problem. Finally, the output layer presents the desired outputs of a trained system. Different nodes are shown at the end of each layer in a neural network. These nodes emulate or imitate biological neurons by processing input data. The relationship among the nodes is manipulated by weights related to the nodes' outputs. This denotes that each node corresponds to a summation value of all inputs that feed a certain node. Several transfer functions can be involved to manipulate the association between the inputs and output of each node such as hyperbolic tangent, pure line, Sigmoid, and so on. In addition, there are biases linked to the nodes that activate them.

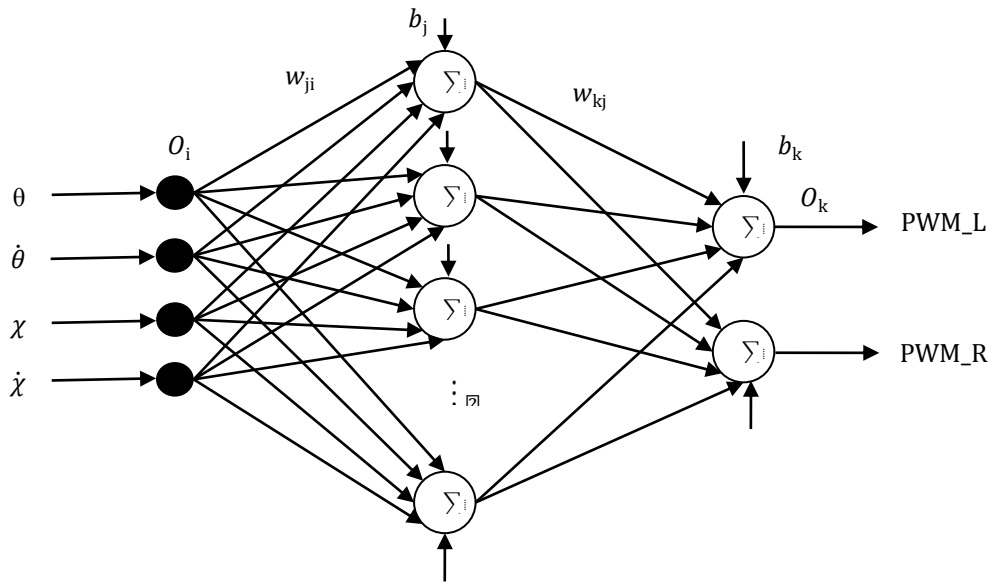


**Fig 3.** A three-layer feed forward neural network

The equations describing interconnection among network layers at each stage as follows:

$$O_k = f(net_k) \quad (3)$$

$$net_k = \left( \sum_j w_{kj} O_j + b_k \right) \quad (4)$$

$$O_i = f(net_j) \quad (5)$$

$$net_j = \left( \sum_i w_{ji} O_i + b_j \right) \quad (6)$$

Where

$w_{ji}$is the weights matrix from the input layer to hidden layer.

$w_{jk}$is the weights matrix from the hidden layer to output layer.

$b_j$ , $b_k$are the hidden layer and the output layer biases, respectively.

f(Net) The activation functions in both the hidden and output layer (linear function for output layer, hyperbolic tangent for the hidden layer). Activation functions calculate a layer's output from its net input.

Many optimization algorithms have already been developed for training neural-networks[25]. The steepest descent optimization algorithm, also common as the error back propagation (EBP) algorithm isolated the dark clouds on the field of artificial neural networks and could be considered as one of the most significant penetrations for training artificial neural networks [26]. The EBP algorithm is still commonly used, however; it is also defined as an inefficient algorithm due to the slow convergence associated with it. There are two main reasons for the time-consuming convergence. The first reason is that its step sizes should be sufficient to the gradients. Logically, small or tiny step sizes should be chosen where the gradient is sharp so as not to diverge from the needed minima (due to oscillation). So, at a constant step size, it should be chosen small. Then, in the gentle place of the gradient, the process of training would be very slow. The second reason is the dissimilar curvature of the error surface in all directions such as the Rosen brock function. The Gauss-Newton optimization algorithm enhances the steepest descent method of slow convergence. The Gauss-Newton algorithm can locate suitable step sizes for each direction and converges them quickly especially if the function of error has a quadratic surface that can converge quickly in the first iteration. But this improvement only occurs when the quadratic approximation error function is realistic. Otherwise, the Gauss-Newton algorithm would typically be unlike or divergent[27].

The Levenberg–Marquardt optimization algorithm (LM)integrates both the steepest descent method and the Gauss-Newton

`

algorithm [27, 28]. Opportunely, it inherits the Gauss-Newton algorithm speed advantage and the stability of the steepest descent method. It's more robust or strong than the Gauss-Newton algorithm, because of its good convergence in many cases, although the error surface is more complex than the quadratic situation. The LM algorithm is slower than the Gauss-Newton algorithm (in the convergent situation) and faster than the steepest descent method.

The essential idea of the Levenberg–Marquardt algorithm is that it carries out a combined training process about the area with complex curvature. The Levenberg–Marquardt algorithm swaps to the steepest descent algorithm until the local curvature is appropriate to make a quadratic approximation. Then it almost becomes the Gauss-Newton algorithm, which can significantly accelerate the convergence.
To derivate the LM algorithm, the subsequent four training algorithms will be presented;

1 back-propagation algorithm
2 Newton's method
3 Gauss-Newton's algorithm
4 Levenberg-Marquardt algorithm.

The mean square error (MSE) used to estimate the error value in training process and network outputs as follows:

$$E(x,w) = \frac{1}{2}\sum_{p=1}^{P}\sum_{m=1}^{M} e_{p,m}^2 \quad (7)$$

$$e_{p,m} = d_{p,m} - O_{p,m} \quad (8)$$

Where

x The vector of the ANN inputs
w The ANN weights matrix
m The output number of the ANN
$e_{p,m}$ The training process error
$d_{p,m}$ The desired output
$O_{p,m}$ The network approximation (actual output)

**A.** Steepest Descent Algorithm (EBP)

The EBP is an optimization algorithm from the first-order. It obtains the minima in error space by using the first-order derivative of the total error function. Normally, gradient g is described as the first-order derivative of total error function:

$$g = \frac{\partial E(x,w)}{\partial w} = \left[\frac{\partial E}{\partial w_1}\frac{\partial E}{\partial w_2}\dots\frac{\partial E}{\partial w_N}\right]^T \quad (9)$$

The steepest descent algorithm update rule as follows:

$$w_{k+1} = w_k - \alpha g_k \quad (10)$$

Where,
αThe learning constant (step size)
N the weights number
k the number of iterations.
The process of training of the steepest descent algorithm is asymptotic convergence. Near the solution, all the gradient vector elements would be very small and the weights change slowly.

**B.** Newton's algorithm

In Newton's method [29-31], it is supposed that all the gradient components, $g_i$ (i = 1, 2… N) Are functions of weights, these weights are linearly independent:

$$g_i = F_i(w_1, w_2, \dots, w_i) \quad (11)$$

Where Nis the weights number and $F_i$is a nonlinear relationship between weights and associated gradient components. Thus, to spread out each $g_i$in (11) the Taylor series is used and the first-order approximation is taken:

$$g_i \approx g_{i,0} + \frac{\partial g_i}{\partial w_1}\Delta w_1 + \frac{\partial g_i}{\partial w_2}\Delta w_2 + \dots + \frac{\partial g_i}{\partial w_i}\Delta w_i \quad (12)$$

By combining the definition of gradient vector g in (11), it could be determined that

$$\frac{\partial g_i}{\partial w_j} = \frac{\partial(\frac{\partial E}{\partial w_j})}{\partial w_j} = \frac{\partial^2 E}{\partial w_i\,\partial w_j} \quad (13)$$

By inserting (13) to (12):

$$g_i \approx g_{i,0} + \frac{\partial^2 E}{\partial w_i\partial w_1}\Delta w_1 + \frac{\partial^2 E}{\partial w_i\partial w_2}\Delta w_2 + \dots + \frac{\partial^2 E}{\partial w_i^2}\Delta w_i \quad (14)$$

So as to obtain the minima of total error function E, the gradient vector components set to zero.

$$0 \approx g_{i,0} + \frac{\partial^2 E}{\partial w_i\partial w_1}\Delta w_1 + \frac{\partial^2 E}{\partial w_i\partial w_2}\Delta w_2 + \dots + \frac{\partial^2 E}{\partial w_i^2}\Delta w_i \quad (15)$$

By combining (11) with (15) up

$$-\frac{\partial \mathrm{E}}{\partial \mathrm{w_i}} = -\mathrm{g_{i,0}} \approx \frac{\partial^2 \mathrm{E}}{\partial \mathrm{w_i}\,\partial \mathrm{w_1}}\Delta \mathrm{w_1} + \frac{\partial^2 \mathrm{E}}{\partial \mathrm{w_i}\,\partial \mathrm{w_2}}\Delta \mathrm{w_2} + \cdots + \frac{\partial^2 \mathrm{E}}{\partial \mathrm{w_i^2}}\Delta \mathrm{w_i} \quad (16)$$

From the last equation, it is clear that there are N parameters for N equations. This denotes that all $\Delta \mathrm{w_i}$ can be estimated throughout the learning process and the weights will be updated periodically. Equation (16) can be written as a matrix as follows:

$$\begin{bmatrix} -g_1 \\ \vdots \\ -g_i \end{bmatrix} = \begin{bmatrix} -\frac{\partial E}{\partial w_1} \\ \vdots \\ -\frac{\partial E}{\partial w_i} \end{bmatrix} = \begin{bmatrix} \frac{\partial^2 E}{\partial w_1^2} & \cdots & \frac{\partial^2 E}{\partial w_1 \partial w_i} \\ \vdots & \vdots & \vdots \\ \frac{\partial^2 E}{\partial w_i \partial w_1} & \cdots & \frac{\partial^2 E}{\partial w_i^2} \end{bmatrix} \times \begin{bmatrix} \Delta w_1 \\ \vdots \\ \Delta w_i \end{bmatrix} \quad (17)$$

Where H is Hessian matrix:

$$H = \begin{bmatrix} \frac{\partial^2 E}{\partial w_1^2} & \cdots & \frac{\partial^2 E}{\partial w_1 \partial w_i} \\ \vdots & \vdots & \vdots \\ \frac{\partial^2 E}{\partial w_i \partial w_1} & \cdots & \frac{\partial^2 E}{\partial w_i^2} \end{bmatrix} \quad (18)$$

By combining (9) and (18) with (17)

$$-\mathrm{g} = \mathrm{H}\Delta \mathrm{w} \quad (19)$$

So

$$\Delta \mathrm{w} = -\mathrm{H}^{-1}\mathrm{g} \quad (20)$$

Consequently, Newton's method updates the rule for weights as follows :

$$\mathrm{w_{k+1}} = \mathrm{w_k} - \mathrm{H_k}^{-1}\mathrm{g_k} \quad (21)$$

Where, H refers to a Hessian matrix that presents the second-order derivatives associated with total error function and supplies the appropriate evaluation on the change of gradient descent. It's noticeable that the inverted Hessian matrix provides compatible step size.

### *C. Gauss-Newton Algorithm*

In Gauss-Newton algorithm, Jacobian matrix J is presented to reduce the difficulty in the calculation process of the second-order derivatives associated with total error function with Newton's method which complicates calculating of Hessian matrix for weight updating.

$$J = \begin{bmatrix} \frac{\partial \mathrm{e_{P,1}}}{\partial \mathrm{w_1}} & \cdots & \frac{\partial \mathrm{e_{P,1}}}{\partial \mathrm{w_i}} \\ \vdots & \vdots & \vdots \\ \frac{\partial \mathrm{e_{P,M}}}{\partial \mathrm{w_1}} & \cdots & \frac{\partial \mathrm{e_{P,M}}}{\partial \mathrm{w_i}} \end{bmatrix} \quad (22)$$

By combining (7) and (9), elements of gradient descent vector can be estimated as follows:

$$\mathrm{g_i} = \frac{\partial \mathrm{E}}{\partial \mathrm{w_i}} = \frac{\partial (\frac{1}{2}\sum_{\mathrm{p=1}}^{\mathrm{P}}\sum_{\mathrm{m=1}}^{\mathrm{M}} \mathrm{e_{p,m}^2})}{\partial \mathrm{w_i}} = \sum_{\mathrm{p=1}}^{\mathrm{P}}\sum_{\mathrm{m=1}}^{\mathrm{M}} \left(\frac{\partial \mathrm{e_{p,m}}}{\partial \mathrm{w_i}}\mathrm{e_{p,m}}\right) \quad (23)$$

The relationship between gradient descent (g) and Jacobian matrix (J) can be obtained by combining (23) and (22) as follows:

$$\mathrm{g} = Je \quad (24)$$

Where the error (e) (a vector of network errors) has the following form;

$$\mathrm{e} = \begin{bmatrix} \mathrm{e_{1,1}} \\ \vdots \\ \mathrm{e_{1,M}} \\ \vdots \\ \mathrm{e_{P,1}} \\ \vdots \\ \mathrm{e_{P,M}} \end{bmatrix} \quad (25)$$

Inserting (7) into (18), the elements of Hessian matrix, i.e., ith row and jth column can be observed as

$$\mathrm{h_{i,j}} = \frac{\partial^2 \mathrm{E}}{\partial \mathrm{w_i}\,\partial \mathrm{w_j}} = \frac{\partial^2 (\frac{1}{2}\sum_{\mathrm{p=1}}^{\mathrm{P}}\sum_{\mathrm{m=1}}^{\mathrm{M}} \mathrm{e_{p,m}^2})}{\partial \mathrm{w_i}\,\partial \mathrm{w_j}} = \sum_{\mathrm{p=1}}^{\mathrm{P}}\sum_{\mathrm{m=1}}^{\mathrm{M}} \left(\frac{\partial \mathrm{e_{p,m}}}{\partial \mathrm{w_i}}\frac{\partial \mathrm{e_{p,m}}}{\partial \mathrm{w_j}}\right) + \mathrm{S_{i,j}} \quad (26)$$

Where $\mathrm{S_{i,j}}$as follows:

$$\mathrm{S_{i,j}} = \sum_{\mathrm{p=1}}^{\mathrm{P}}\sum_{\mathrm{m=1}}^{\mathrm{M}} \frac{\partial^2 \mathrm{e_{p,m}}}{\partial \mathrm{w_i}\,\partial \mathrm{w_j}}\partial \mathrm{e_{p,m}} \quad (27)$$

From Newton's method, it is supposed that the $S_{i,j}$is a very small value near to zero[23]. Consequently, Jacobian matrix (J) and Hessian matrix (H) are linked with the following equation:

$$H = J^T J \quad (28)$$

By using (21), (24) and (28), the weights updating rule of the Gauss-Newton algorithm can be illustrated as in below:

$$w_{k+1} = w_k - (J_k^T J_k)^{-1} J_k e_k \quad (29)$$

It is clear that the advantage of the Gauss-Newton algorithm, which isn't available through the standard Newton's method in weight updating, is that the previous does not demand the computation of second-order derivatives of the total error function by presenting Jacobian matrix (J). But, the Gauss-Newton algorithm still suffers from the same convergent problem as the Newton algorithm for complex error space optimization. Mathematically, the problem lies in the inability to get the inverse of the matrix$(J_k^T J_k)$.

**D.** The Levenberg–Marquardt optimization algorithm

The Levenberg–Marquardt optimization algorithm was separately developed by Kenneth Levenberg and Donald Marquardtand presented a numerical solution for minimizing a nonlinear function problem[32, 33]. It is used to update weight and bias values of feed forward neural net due to its fast and stable convergence. In the scope of neural-networks, this algorithm is suitable to train small- and medium-sized problems. It is used in Matlab through a trainlm function which is usually the best Back propagation algorithm in the toolbox and is kindly recommended as the best choice supervised algorithm yet still requiring a larger memory than other algorithms. The Levenberg-Marquardt algorithm was investigated to arrive at the second-order training speed without computing the Hessian matrix .The Levenberg-Marquardt algorithm employs the Hessian matrix approximation in the following Newton-like update:

$$H = J^T J + \mu I \quad (30)$$

$$X_{k+1} = X_k - [J^T J + \mu I]^{-1} J^T e \quad (31)$$

The Levenberg–Marquardt algorithm update rule is described in the below equation

$$w_{k+1} = w_k - (J_k^T J_k + \mu I)^{-1} J_k e_k \quad (32)$$

When the scalar value of combination coefficientμ is zero, this is Newton's method, using the calculated Hessian matrix. When μ is large, this becomes a gradient descent with a small learning rate (step size). Newton's method near an error minimum is faster and more precise, so the aim is to move towards Newton's method as quickly as possible. Thus, μ is reduced after each reduction in error function and rises only when a tentative step would increase the error or performance function. In this way, the performance function will continuously be reduced at all algorithm iterations.

## 6. Results

**A.** State-feedback Controller

A state-feedback controller is designed in this work to stabilize the real-time model of TWIP. The state-feedback controller stabilizes the model by putting or moving the unstable closed-loop poles to a stability region. Figure (5) shows State feed back controller Simulink model is developed for the real-timework. The state-feedback controller provides remarkable close loop response of the two-wheel inverted pendulum and it shows that the pendulum angle is stable which is shown in Fig.7.

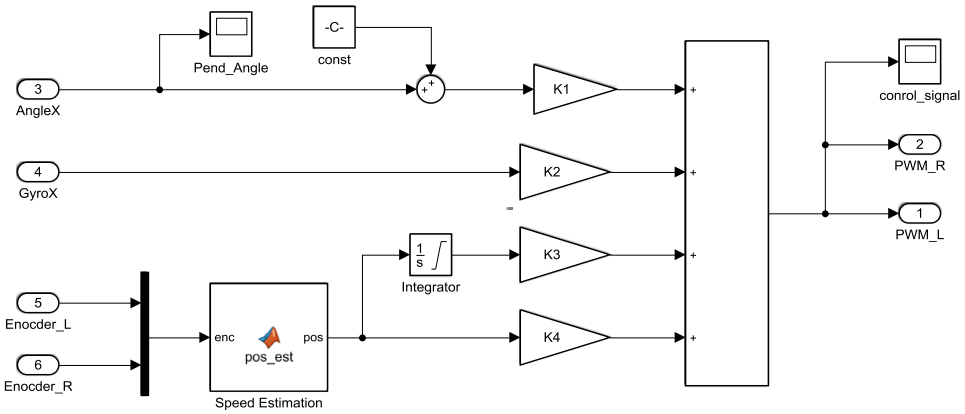


**Fig.5** TWIP Simulink model of control using the state-feedback controller

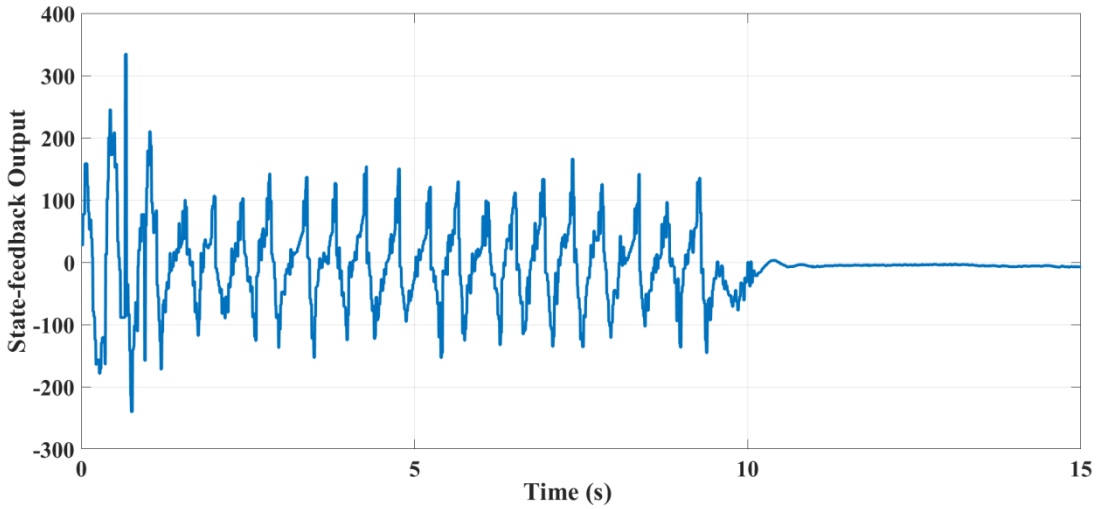


**Fig 6.** The control signal of TWIP model provided by the state-feedback controller

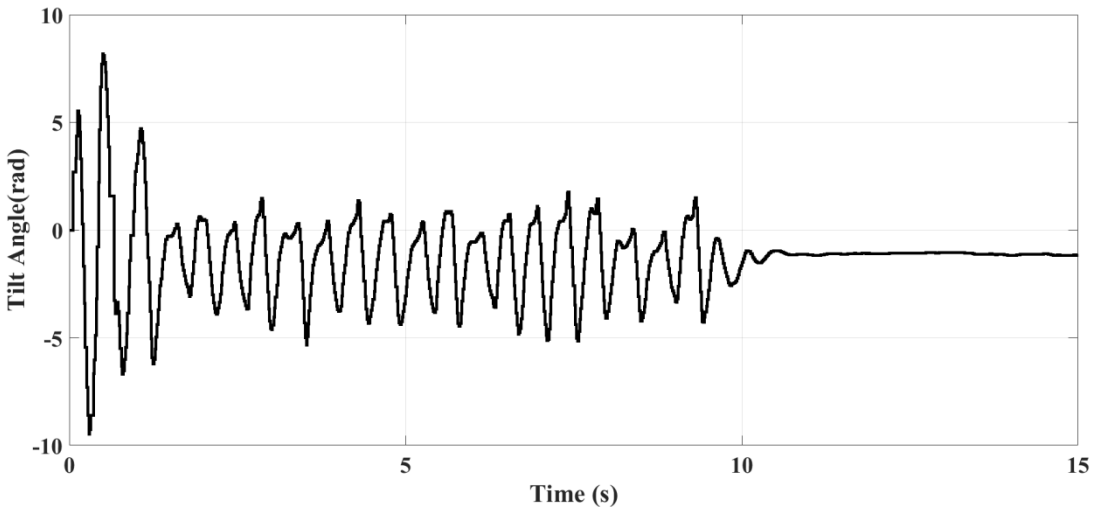


**Fig. 7** Two-wheel inverted Pendulum angle controlled by the state-feedback controller

## B. Neural Networks Controller

Neural networks controllers were developed to stabilize the real-time model of two-wheel inverted pendulum system. The feed-forward networks were able to precisely stabilize the model. Different architectures of FFNN are tested using MATLAB-Simulink as shown in Fig.8. Most of them are able to stabilize the system but are different in their performance according to the size of the hidden layer. The best network to stabilize the model was

the 4-10-2 FFNN as observed in real time work which takes small settling time and low over shoot to stabilize TWIP model. Figure (9) illustrates the responses of each tested network, indicating that the best performance belongs to the 4-10-2 network. This network gives better performance than each of the 4 tested networks as well as state-feedback as shown in Fig.9. Its hidden layer has ten neurons with the hyperbolic tangent activation function and pure line function in the output layer. It denotes that the biases number in the hidden layer is ten, and the weights number among the input layer and the hidden layer is 40. Because there are two neurons in the output layer, it denotes the weights number between the hidden layer and the output layer as 20 and we have two biases in the output layer. The weights and biases matrices of this network are written below. The mean square error of training versus epoch numbers is shown in Fig.11. The goal is represented by a dashed line which equals 1e-06 as set in the nntool GUI window. In Fig.12, the best result shows when the actual output is similar to the desired output and that is represented by a dashed line. The solid line represents the best fit linear regression between network approximations (current outputs) and desired outputs. The 'R' value specifies the nature relationship between the actual outputs or network approximations and targets. When 'R' = 1, this proves that there is a perfect linear relationship between actual outputs and desired outputs or targets. If 'R' is approximately zero, there is no linear relationship between actual outputs and desired outputs. From Fig.12 it's obvious that R=1 which means that there is an exact linear relation between the network output and the state-feedback output (target).Table (2)displays the parameters which are common to all tested neural networks and Table (3)displays the output parameters of the best tested network.

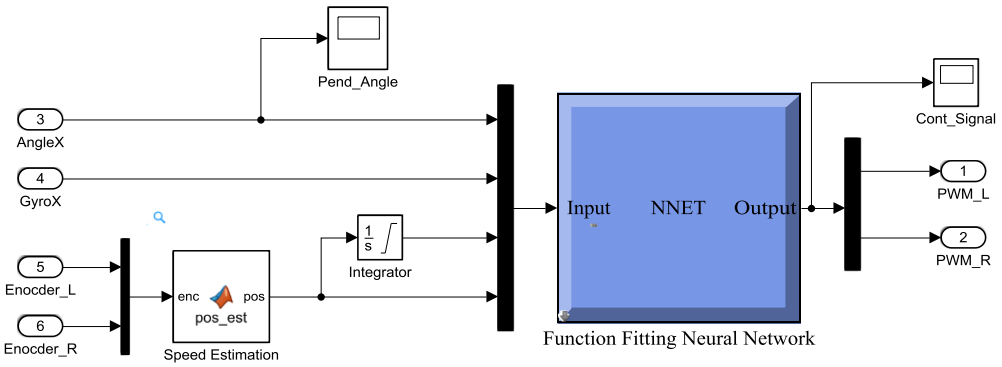


**Fig. 8** Simulink model of real-time control of TWIP using feedforward neural network controller

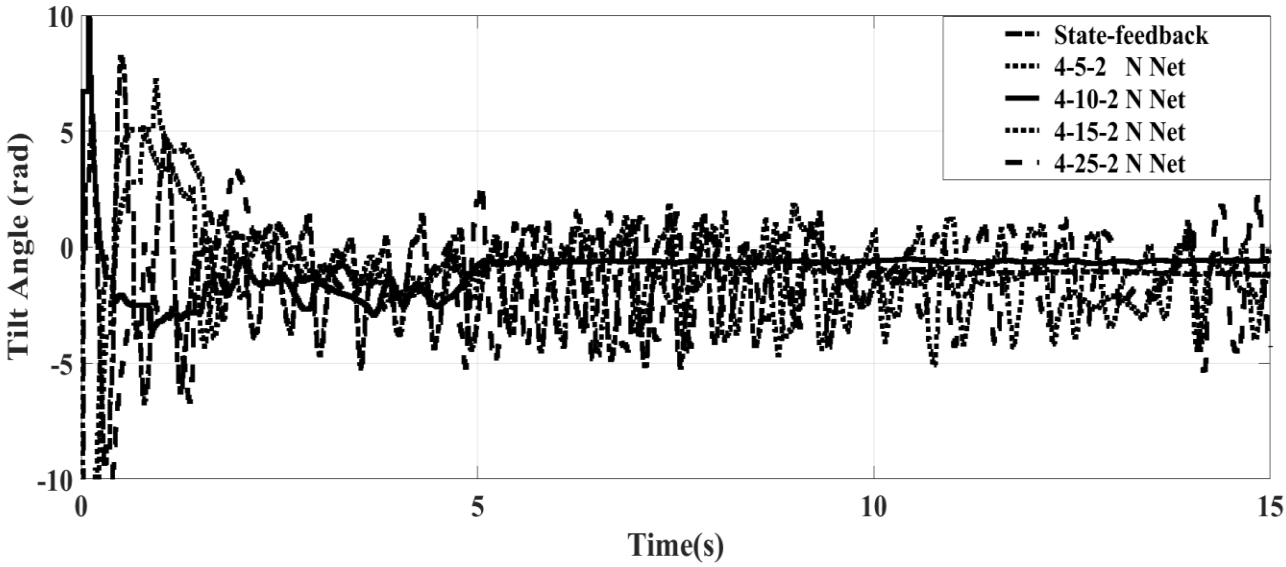


**Fig.9** Comparison between the state-feedback controller and feed-forward neural networks angle response

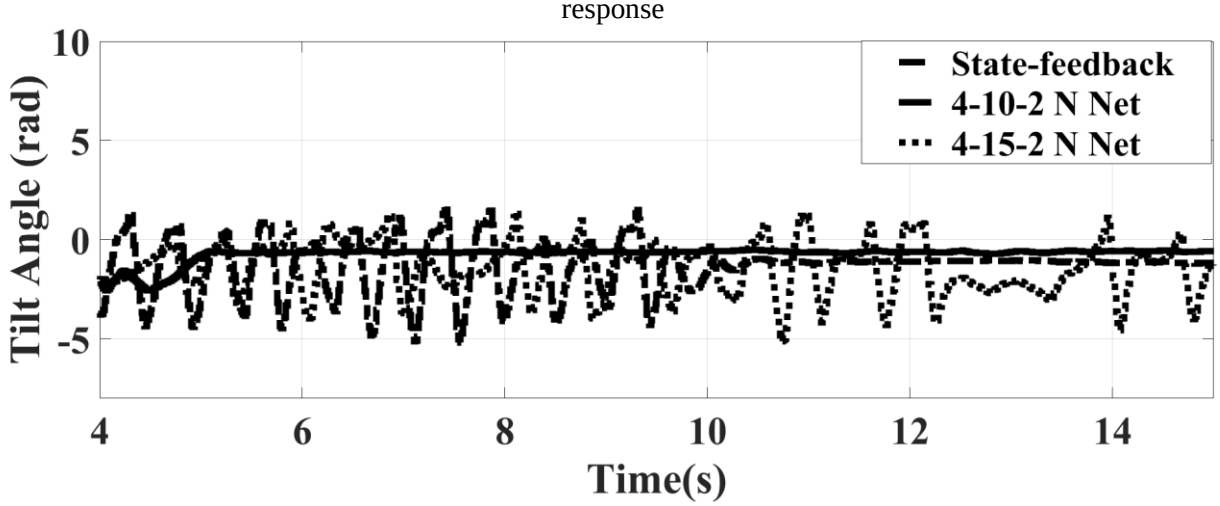


**Fig.10** State-feedback, 4-10-2 NN and 4-15-2 NN controller response

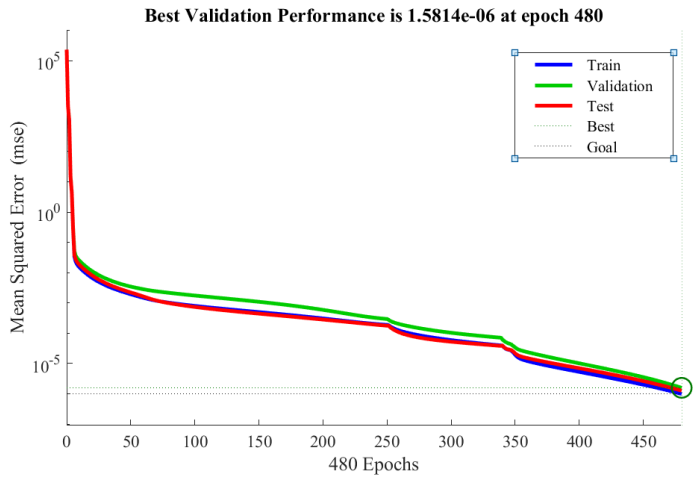


**Fig.11** Training performance for 4-10-2 feed-forward NN of two-wheel inverted pendulum angle control

`

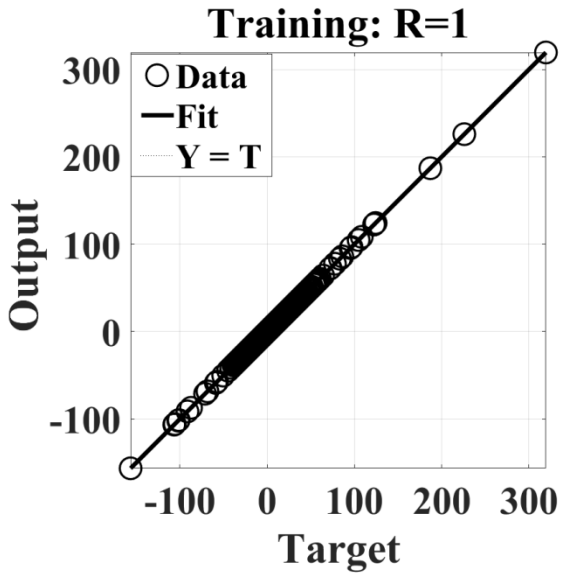


**Fig.12** Regression plot for 10-4-2 feed forward NN of two-wheel inverted pendulum angle control

**C.** The weights and biases values of the 4-10-2 feedforward neural network

$$w_{ij} = \begin{bmatrix} 2.2101 & 2.4250 & -2.8033 & 11.0636 \\ 4.9485 & 5.7542 & -0.3351 & 4.2578 \\ -1.1903 & -0.2418 & 2.0944 & -2.0278 \\ 2.1744 & 2.3994 & -0.1363 & 1.6311 \\ 2.4530 & -0.4101 & 1.9150 & -6.0918 \\ -0.2403 & -0.3194 & -0.0053 & -0.2679 \\ -1.1152 & -1.4280 & -0.0076 & -1.1531 \\ -0.3980 & 0.8751 & -1.4866 & -7.0246 \\ 2.3629 & 1.9870 & 0.1583 & 2.8165 \\ -0.8172 & -1.8545 & -3.2340 & -5.8106 \end{bmatrix} \quad b_{ij} = \begin{bmatrix} -1.7504 \\ -2.1837 \\ -0.4975 \\ 1.3461 \\ -2.5777 \\ -2.5777 \\ 0.1480 \\ 3.1997 \\ 0.1270 \\ 5.1741 \end{bmatrix},$$

$$w_{jk} = \begin{bmatrix} 3.6053e-06 & -0.0018392 & -4.0651e-06 & -0.0047693 & -1.9173e-05 & -3.9774 & 0.074904 & 1.4564e-05 & 8.0258e-06 & -7.3855e-05 \\ 3.6053e-06 & -0.0018392 & -4.0651e-06 & -0.0047693 & -1.9173e-05 & -3.9774 & 0.074904 & 1.4564e-05 & 8.0258e-06 & -7.3855e-05 \end{bmatrix},$$

$$b_{jk} = \begin{bmatrix} -0.20269 & -0.2027 \end{bmatrix}$$

TABLE 2
SHARED PARAMETERS BETWEEN NETWORKS

| Parameter | value |
|---|---|
| Number of input samples | 12000 |
| Sampling time | 0.005 (s) |
| Number of training data samples | 8400 (70%) |
| Number of validation data samples | 1800(30%) |
| Number of test data samples | 1800 (30%) |
| Goal | 0.000001 |
| Learning rate | 0.01 |

`

TABLE 3
TESTED FEED-FORWARD NEURAL NETWORKS TRAINING PARAMETERS

| FF-Neural Network Architecture | Num-of Epochs | Training Time (s) | MSE of training data | MSE of Validation data | Regression (R) | Gradient | Momentum (Mu) |
|---|---|---|---|---|---|---|---|
| 4-5-2 | 685 | 11 | 9.97e-07 | 1.3365e-06 | 9.9999e-01 | 0.00409 | 0.001 |
| 4-10-2 | 480 | 13 | 9.93e-07 | 1.5814e-06 | 9.9999e-01 | 0.368 | 0.0001 |
| 4-15-2 | 46 | 2 | 9.87e-07 | 6.7093e-07 | 9.9999e-01 | 0.257 | 0.001 |
| 4-25-2 | 804 | 123 | 9.99e-07 | 1.7976e-06 | 9.9999e-01 | 0.00401 | 0.0001 |

## 7. Conclusion

In this paper, supervised feed-forward neural networks are developed to real-time control of the TWIP angle. A state-feedback controller was developed previously to stabilize the two-wheel inverted pendulum. Input-Output datasets were collected by using Arduino UNO board and Matlab-Simulink. The data was then used later for training neural networks. When the training process ended, the neural network was exported to MATLAB-Simulink and the network was placed in the feedback loop instead of the existing state-feedback controller as shown in Fig.8. The SFFNN trained with a different number of neurons in hidden layer stabilizing the TWIP model. Therefore, the results indicate that feed-forward networks were able to stabilize the TWIP model. The network response depends on hidden layer size (neuron numbers). All the SFFNN have four neurons in input layer (system states) and two neurons in the output layer (DC motors control signals) but different in hidden layer number of neurons. Figure (9) shows that the feed forward networks with 10 neurons in hidden layer give better performance among the tested networks and successfully stabilize the system better than a state-feedback controller. The SFFNN controller successfully stabilizes the TWIP model without being knowledgeable about any system dynamics.

## REFERENCES


[1] Ponce, "A REVIEW OF INTELLIGENT CONTROL SYSTEMS APPLIED TO THE INVERTED-PENDULUM PROBLEM," *American Journal of Engineering and Applied Sciences American Journal of Engineering and Applied Sciences,* vol. 7, pp. 194-240, 2014.

[2] J. Goncalves, N. Gago, C. Arantes, F. Soares, J. S. Esteves, P. Garrido, *et al.*, "A realization of the inverted pendulum and cart," *Lect. Notes Eng. Comput. Sci. Lecture Notes in Engineering and Computer Science,* vol. 2217, pp. 434-439, 2015.

[3] C.-H. Huang, W.-J. Wang, and C.-H. Chiu, "Design and implementation of fuzzy control on a two-wheel inverted pendulum," *IEEE Transactions on Industrial Electronics,* vol. 58, pp. 2988-3001, 2011.

[4] F. Grasser, A. D'arrigo, S. Colombi, and A. C. Rufer, "JOE: a mobile, inverted pendulum," *IEEE Transactions on industrial electronics,* vol. 49, pp. 107-114, 2002.

[5] O. Boubaker, "The inverted pendulum: history and survey of open and current problems in control theory and robotics," *TheInverted Pendulum in Control Theory and Robotics: From Theory to New Innovations,* p. 1, 2017.

[6] R. P. M. Chan, K. A. Stol, and C. R. Halkyard, "Review of modelling and control of two-wheeled robots," *Annual Reviews in Control,* vol. 37, pp. 89-103, 2013.

[7] A. A. Bature, S. Buyamin, M. N. Ahmad, M. Muhammad, and A. A. Muhammad, "Identification and model predictive position control of Two Wheeled Inverted Pendulum

mobile robot," *J. Teknol*, vol. 73, pp. 153-156, 2015.

[8] S. Nawawi, M. Ahmad, and J. Osman, "Real-time control of a two-wheeled inverted pendulum mobile robot," *World Academy of Science, Engineering and Technology*, vol. 39, pp. 214-220, 2008.

[9] J. Gonçalves, N. Gago, C. Arantes, F. Soares, G. Lopes, J. S. Esteves, *et al.*, "A Realization of the Inverted Pendulum and Cart," in *Proceedings of the World Congress on Engineering*, 2015.

[10] H.-J. Lee and S. Jung, "Gyro sensor drift compensation by Kalman filter to control a mobile inverted pendulum robot system," in *Industrial Technology, 2009. ICIT 2009. IEEE International Conference on*, 2009, pp. 1-6.

[11] V. Mladenov, "Application of neural networks for control of inverted pendulum," *WSEAS Transactions on Circuits and Systems*, vol. 10, pp. 49-58, 2011.

[12] T. Callinan, "Artificial Neural Network identification and control of the inverted pendulum," *MEng Project Reports, School of Electronic Engineering Dublin City University*, 2003.

[13] S. Jung and S. S. Kim, "Control experiment of a wheel-driven mobile inverted pendulum using neural network," *IEEE Transactions on Control Systems Technology*, vol. 16, pp. 297-303, 2008.

[14] K. Pathak, J. Franch, and S. K. Agrawal, "Velocity and position control of a wheeled inverted pendulum by partial feedback linearization," *IEEE Transactions on robotics*, vol. 21, pp. 505-513, 2005.

[15] J. S. Noh, G. H. Lee, and S. Jung, "Position control of a mobile inverted pendulum system using radial basis function network," in *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, 2008, pp. 370-376.

[16] C. Yang, Z. Li, R. Cui, and B. Xu, "Neural network-based motion control of an underactuated wheeled inverted pendulum model," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, pp. 2004-2016, 2014.

[17] C.-C. Tsai, H.-C. Huang, and S.-C. Lin, "Adaptive neural network control of a self-balancing two-wheeled scooter," *IEEE Transactions on Industrial Electronics*, vol. 57, pp. 1420-1428, 2010.

[18] Z. Li and C. Yang, "Neural-adaptive output feedback control of a class of transportation vehicles based on wheeled inverted pendulum models," *IEEE Transactions on Control Systems Technology*, vol. 20, pp. 1583-1591, 2012.

[19] C. Yang, Z. Li, and J. Li, "Trajectory planning and optimized adaptive control for a class of wheeled inverted pendulum vehicle models," *IEEE Transactions on Cybernetics*, vol. 43, pp. 24-36, 2013.

[20] S. Jung and S. su Kim, "Hardware implementation of a real-time neural network controller with a DSP and an FPGA for nonlinear systems," *IEEE Transactions on Industrial Electronics*, vol. 54, pp. 265-271, 2007.

[21] R. C. Ooi, "Balancing a two-wheeled autonomous robot," *University of Western Australia*, vol. 3, 2003.

[22] S. Jadlovska and J. Sarnovsky, "A complex overview of modeling and control of the rotary single inverted pendulum system," *Advances in Electrical and Electronic Engineering*, vol. 11, p. 73, 2013.

[23] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the Marquardt algorithm," *IEEE transactions on Neural Networks*, vol. 5, pp. 989-993, 1994.

`

[24] B. C. Aissa and C. Fatima, "Neural Networks Trained with Levenberg-Marquardt-Iterated Extended Kalman Filter for Mobile Robot Trajectory Tracking," *Journal of Engineering Science & Technology Review*, vol. 10, 2017.

[25] R. Battiti, "First-and second-order methods for learning: between steepest descent and Newton's method," *Neural computation*, vol. 4, pp. 141-166, 1992.

[26] B. M. Wilamowski, "Neural network architectures and learning algorithms," *IEEE Industrial Electronics Magazine*, vol. 3, 2009.

[27] H. Yu and M. Bogdan, "Levenberg-Marquardt training," in *Electrical Engineering Handbook intelligent systems*, ed:
, 2011, pp. 1-16

[28] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the society for Industrial and Applied Mathematics*, vol. 11, pp. 431-441, 1963.

[29] J. J. Callahan, *Advanced calculus: a geometric view*: Springer Science & Business Media, 2010.

[30] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization," in *Advances in neural information processing systems*, 2014, pp. 2933-2941.

[31] S. Wright and J. Nocedal, "Numerical optimization," *Springer Science*, vol. 35, p. 7, 1999.

[32] A. A. Suratgar, M. B. Tavakoli, and A. Hoseinabadi, "Modified Levenberg-Marquardt method for neural networks training," *World Acad Sci Eng Technol*, vol. 6, pp. 46-48, 2005.

[33] B. M. Wilamowski and H. Yu, "Improved computation for Levenberg–Marquardt training," *IEEE transactions on neural networks*, vol. 21, pp. 930-937, 2010.

# التحكم المقود للشبكات العصبية فيالنظام الحقيقي للبندول المعكوس ذي العجلتين

**الملخص العربي:**

يعد البندول المعكوس ذي العجلتين من اهم الامثلة واكثرها انتشارا علي الانظمة الغير خطية ،مفتوحة الحلقة وغير المستقرة للغاية مما يجعل البحث عن تقنية تحكم ذكية تعمل علي تثبيته في وضع افقي مستقر تحديا يجذب اهتمام الباحثين في مجال التحكم الذكي. بداية ، يتم استخدام وحدة تحكم التغذية المرتدة او الرجعية التي تستخدم حالات النظام الديناميكي وإشارات التحكم في المحركات لبناء قرار التحكم الدقيق لتثبيت النظام افقيا ثم، يتم تدريب الشبكات العصبية الإطعامية الموجهة إلى الأمام اعتمادا علي خوارزمية التحسين لفينبرغ– ماركوارت للانتشار الخلفي باستخدام قياسات في الوقت الحقيقي لحالات النظام وإشارات التحكم في المحركات و اوضحت النتايج العملية ان الشبكات العصبية كانت قادرة علي تحقيق وانجاز مهمة استقرار البندول المعكوس ذي العجلتين بكفاءة.