Dominik ŁUCZAK, Adrian WÓJCIK

Politechnika Poznańska, Instytut Automatyki i Inżynierii Informatycznej

doi:10.15199/48.2016.05.19

DSP implementation of state observers for electrical drive with elastic coupling

Streszczenie. *W pracy przedstawiono wyniki implementacji trzech dyskretnych metod estymacji wektora stanu układu dynamicznego – obserwatora Luenbergera, filtru Kalmana i estmatora neuronowego. Implementacji dokonano w języku C++, w środowisku VisualDSP 5.0, przy założeniu wykorzystania algorytmów do obserwacji stanu części mechanicznej napędu elektrycznego, charakteryzującego się sprężystym połaczeniem z maszyną roboczą. Wykorzystano technikę programowania obiektowego.* ***Wyniki implementacji trzech dyskretnych metod estymacji wektora stanu układu dynamicznego – obserwatora Luenbergera, filtru Kalmana i estmatora neuronowego***

Abstract*. Article presents the results of implementation of three discrete methods of estimation of the state vector of the dynamic system - Luenberger observer, Kalman filter and neural network estimator. Implementation was done in C ++, in an VisualDSP 5.0 environment, assuming the use of algorithms to observe the state of the mechanical part of the electric drive, characterized by a elastic connection with the working machine. Autors had used object-oriented programming technique.*

Słowa kluczowe: obserwator stanu, procesor sygnałowy, implementacja w C++, napęd elektryczny, połączenie sprężyste.
Keywords: state observers, DSP, implementation in C++, electric drive, elastic coupling.

Introduction

Requirements for modern electrical drives - especially dynamic characteristics - often necessitate the need for control based on whole state of the system, not only directly measured values. This implies the need to estimate the values that cannot be measured.

Popular and interesting algorithms for performing this task are Luenberger state observer [1], [2], Kalman filter [3], [4], and neural-network-based [5] estimator. Applying these methods in control system requires implementation that let both simple initializations as well as easy obtains a result of each iteration. The specific elements of control and measurement system require application of proper software [6]. Principle of separation of estimation and control suggest the possibility of encapsulation of these methods. Therefore it is justifiable to use object-oriented techniques.

Implementation was based on the assumption that the methods are used to estimate state vector of mechanical part of electrical drive. The block diagram of the system is shown in Fig.1. Mechanical part of drive is two-mass system with elasticity.

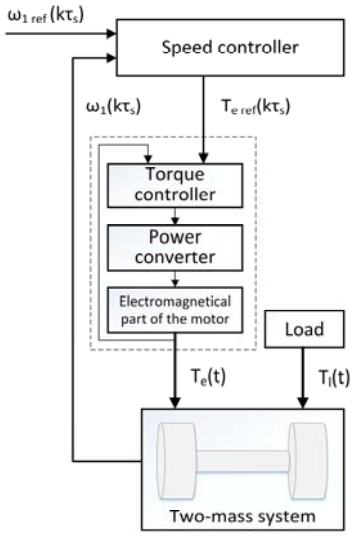

Fig.1. Drive structure

The electric motor works in the cascade structure. Inner loop consists of electromagnetic torque (active current) controller, power converter and electromagnetic part of the motor. Outer loop consists of speed controller, closed torque control loop and mechanical part of the motor. Speed control is performed by the discrete PI controller. Algorithm is executed by digital signal processor DSP.

Regardless of the method and mathematical model of the system, the state observation is based on the measurement of angular velocity of the shaft on the motor side and knowledge of reference motor torque.

The article describes implementation of observers in C++ with usage of object-oriented programing techniques. Presented approach is one module of software framework for electrical drives, which is under development by research team.

Mathematical model of drive system

Mathematical model of the system is composed of three elements: model of the mechanical part, model of closed torque control loop and speed controller. In most articles two-mass mechanical system is considered [7]–[10].

Two-mass system with elasticity is physical model of the mechanical part of drive. Estimating the state vector requires state-space model. Angular velocity of the shaft on the motor side ω_1, on the load side ω_2 and torsion T_s, were chosen as state variables (1).

$$\mathbf{x}(t) = \begin{bmatrix} \omega_1 \\ \omega_2 \\ T_s \end{bmatrix} \tag{1}$$

Inputs of the system (2) are electromagnetic motor torque T_e and load torque T_l. Output of the system (3) is angular velocity on the motor side ω_1. This one is only state variable which can be measured directly. It is also necessary for speed control.

$$\mathbf{u}(t) = \begin{bmatrix} T_e \\ T_l \end{bmatrix} \tag{2}$$

$$\mathbf{y}(t) = \omega_1 \tag{3}$$

Based on [11] state-space matrix can be written as (4).

(4) $$\mathbf{A}=\begin{bmatrix}0 & 0 & -\frac{1}{J_1}\\ 0 & 0 & \frac{1}{J_2}\\ k & -k & 0\end{bmatrix}\quad \mathbf{B}=\begin{bmatrix}\frac{1}{J_1} & 0\\ 0 & -\frac{1}{J_2}\\ 0 & 0\end{bmatrix}$$
$$\mathbf{C}=\begin{bmatrix}1 & 0 & 0\end{bmatrix}\quad \mathbf{D}=\begin{bmatrix}0 & 0\end{bmatrix}$$

The parameters of the model present in the system matrix $\mathbf{A}$ and input matrix $\mathbf{B}$ are: J_1, J_2 - moment of inertia respectively on the motor side and on the load side (kg·m^2) and k - modulus of elasticity (kg·m^2·s^{-2}).

Luenberger observer and Kalman filter synthesis requires discrete-time state-space model with sample time τ_s equal to algorithms iteration time. Based on [12] Tustin transform was applied as discretization method, according to formulas (5).

(5)
$$\mathbf{A}_d=(\mathbf{I}-\tfrac{1}{2}\tau_s\mathbf{A})^{-1}(\mathbf{I}+\tfrac{1}{2}\tau_s\mathbf{A})$$
$$\mathbf{B}_d=\tau_s(\mathbf{I}-\tfrac{1}{2}\tau_s\mathbf{A})^{-1}\mathbf{B}$$
$$\mathbf{C}_d=\mathbf{C}(\mathbf{I}-\tfrac{1}{2}\tau_s\mathbf{A})^{-1}$$
$$\mathbf{D}_d=\tfrac{1}{2}\tau_s\mathbf{C}(\mathbf{I}-\tfrac{1}{2}\tau_s\mathbf{A})^{-1}\mathbf{B}+\mathbf{D}$$

Obtained discrete-time model must be modified due to fact that load torque cannot be directly measured. Input matrix $\mathbf{B}_d$ is therefore reduced to first column.

Assuming optimization of motor torque controller, closed torque control loop can be modeled as first-order inertial system with time constant τ_T, time delay τ_{Td} and gain equal to torque constant k_T. In this case, black box model is sufficient. Model of inner loop is described by continuous-time transfer function (6).

(6) $$G(s)=\frac{k_T}{s\tau_T+1}e^{-s\cdot\tau_{Td}}$$

Input of torque control loop is discrete-time signal $T_{e\,ref}(k\tau_s)$. Zero-order hold interpolation was assumed.

Discrete parallel PI controller was applied as speed controller (7). Based on [13], controller was tuned according to formulas (8).

(7) $$u(k\tau_s)=K_P e(k\tau_s)+K_I\sum_{n=0}^{k}e(n\tau_s)$$

(8) $$K_P=2\sqrt{\frac{\tau_2}{\tau_c}}\qquad K_I=\frac{\tau_1}{\tau_2\tau_c}$$

Time constants of inertia of the motor shaft τ_1 and τ_2 and time constant of the motor shaft stiffness τ_c are expressed by formulas (9). Where ω_N is nominal motor angular velocity and T_{eN} is nominal motor torque. Controller output is limited to $\pm1.5\cdot T_{eN}$. Clamping was applied as anti-windup method.

(9) $$\tau_1=\frac{\omega_N}{T_{eN}}\cdot J_1\quad \tau_2=\frac{\omega_N}{T_{eN}}\cdot J_2\quad \tau_c=\left(\frac{\omega_N}{T_{eN}}\right)^{-1}\cdot\frac{1}{k}$$

State observers algorithms

Discrete full-order Luenberger observer and Kalman filter are methods of estimating the state vector based on discrete-time state-space model of the system. Neural estimator is based on neural networks trained off-line with measurement or simulation data. In all cases, the most compact and clear notation - and implementation - is matrix analysis. Application of object-oriented techniques allowed to create class template *matrix*. Class provides an interface for hardware implementation of basic operation of linear algebra executed on DSP - addition, multiplication and inversion. Subclass template *vector* is special case *matrix* - a column matrix. As a result, observation algorithms were implemented in C++, in a way resembling MATLAB scripting language.

Each method has been implemented as C++ class. All of them inherit from the base class *state_observer*, containing the current state vector *x_act* and virtual member function *update()*. Function *update()* performs single step of algorithm.

Discrete full-order Luenberger state observer is method applicable under the following conditions: system is observable, discrete-time state-space model is well known (the precise identification) and noise level in system is insignificant. However, initial state of system is unknown. Block diagram of algorithm as prediction observer is shown in Fig. 2. Due to the assumption made on the system, both input u and output y are scalar values.

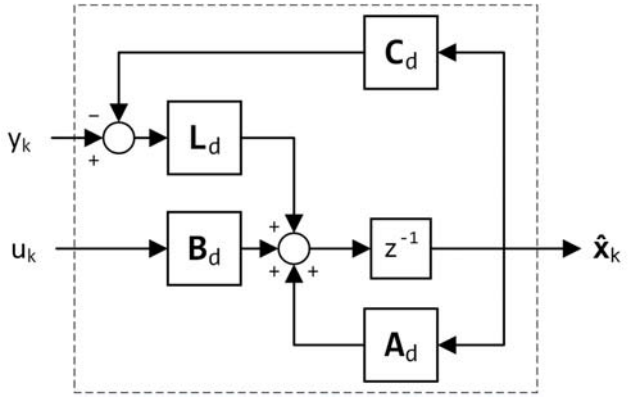

Fig.2. Discrete full-order Luenberger state observer as prediction observer

Equivalent description, which is also a basis for the implementation, shown in the following pseudocode [14], [15]:

1) update at time k :

(10) $$\mathbf{x}(k)=\mathbf{x}(k+1).$$

2) prediction at time k to $k+1$:

(11) $$\mathbf{x}(k+1)=\mathbf{A}_d\mathbf{x}(k)+\mathbf{B}_d u(k)+\mathbf{L}_d\left(y(k)-\mathbf{C}_d\mathbf{x}(k)\right).$$

Listing 1. Body of the *upadate()* function, *luenberger_observer* class (observer gain $\mathbf{L}_d$ designated as K)

```
vector<double>
luenberger_observer::update(double u, double y)
{
        // update
        x_act = x_next;

        // prediction
        x_next = Ad*x_act + Bd*u + K*(y - Cd*x_act);

        // current (actual) state
        return x_act;
}
```

Listing 2. An examples of definion of pointer to *luenberger_observer* class object and call member function *update(*

```
// array of state observers
state_observer** observer = new state_observer*[3];

// class contructor
observer[0] = new luenberger_observer(Ad, Bd, Cd, Ld, x_init);

// result vector
vector<double> x_e(3, 0.0);

// Luenbeger observer update
x_e = observer[0]->update(input, output);
```

Code in C++, shown in Listing 1.-2., presents body of the member function *update()* in *luenberger_observer* class and examples of definition of pointer to class object and call function *udpate()*.

An example of definition of pointer to class object contains call class constructor. Observer gain $\mathbf{L}_d$ is constructor argument. Therefore this vector needs to be specified and define outside of the class - implementation does not provide methods of determination of the gain.

Kalman filter is extension of discrete state observer concept. Problem of determination of the filter gain is solved. Therefore *kalman_filter* class is derived class of *luenberger_observer*. Filter gain $\mathbf{K}$ is optimal in sense of minimize the variance of the estimation error. Standard form of Kalman filter is current estimator (correction takes place at update step, not prediction step). In Fig. 3. block diagram of Kalman filter is shown - calculations related to the determination of gain have been omitted. $\mathbf{K}_k$ is filter gain computed at time *k*.

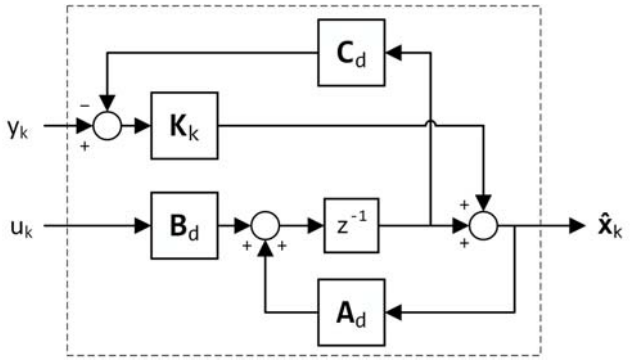

Fig.3. Kalman filter as observer

Complete description, including formulas for filter gain $\mathbf{K}$ and estimation error covariance matrix $\mathbf{P}$, which is a basis for the implementation, was shown in the following pseudocode [16], [17]:

1) computing gain at time k :

$$\mathbf{K}(k)=\left(\mathbf{P}(k+1)\mathbf{C}_d^T\right)\left(\mathbf{C}_d\mathbf{P}(k+1)\mathbf{C}_d^T+\mathbf{R}_d\right)^{-1}. \quad (12)$$

2) update and correction at time k :

$$\mathbf{x}(k)=\mathbf{x}(k+1)+\mathbf{K}\left(y(k)-\mathbf{C}_d\mathbf{x}(k+1)\right), \quad (13)$$

$$\mathbf{P}(k)=\left(\mathbf{I}-\mathbf{K}(k)\mathbf{C}_d\right)\mathbf{P}(k+1).$$

3) prediction at time k to $k+1$:

$$\mathbf{x}(k+1)=\mathbf{A}_d\mathbf{x}(k)+\mathbf{B}_d u(k), \quad (14)$$

$$\mathbf{P}(k+1)=\mathbf{A}_d\mathbf{P}(k)\mathbf{A}_d^T+\mathbf{Q}_d.$$

Code in C++, shown in Listing 3.-4., presents body of the member function *update()* in *kalman_filter* class and examples of definition of pointer to class object and call function *udpate()*.

Listing 3. Body of the *upadate()* function, *kalman_filter* class

```
vector<double>
kalman_filter::update(double u, double y)
{
        // gain computation
        K = (P * Cd.tr()) / (Cd * P * Cd.tr() + Rd);

        // update and correction
        x_act += K*(y - Cd*x_act);
        P -= K*Cd*P;

        // prediction
        x_next = Ad*x_act + Bd*u;
        P = Ad*P*Ad.tr() + Qd;

        // current state
        return x_act;
}
```

Listing 4. An examples of definion of pointer to *kalman_filter* class object and call member function *update()*

```
// array of state observers
state_observer** observer = new state_observer*[3];

// class contructor
observer[1] = new kalman_filter(Ad, Bd, Cd, Qd, Rd, x_init, P_init);

// result vector
vector<double> x_e(3, 0.0);

// Kalman filter update
x_e = observer[1]->update(input, output);
```

Kalman_filter class constructor has more arguments than its base class - although the solution to the problem of determination of gain is built into the algorithm, it is necessary to provide information about noises occurring in the system. These information are stored in form of system noise covariance matrix $\mathbf{Q}_d$ and measurement noise covariance matrix $\mathbf{R}_d$. Both matrices refer to a discrete-time noise model, with a sampling time equal to algorithm iteration time. It is also necessary to determine the initial estimation error covariance matrix $\mathbf{P}(0)$.

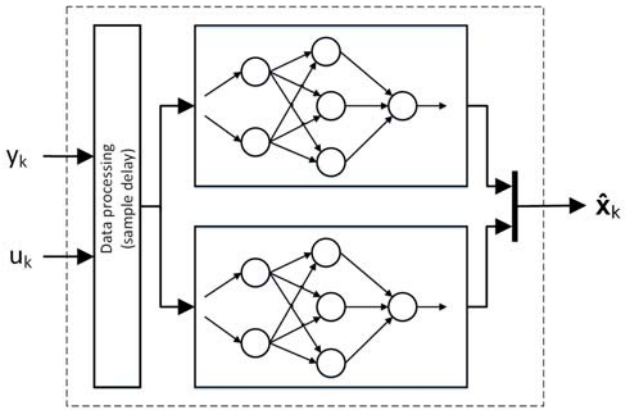

Fig.4. Neural-network-based state estimator

Neural estimator is implementation of set of neural networks trained off-line. Each network estimates one of

unknown state variables. For the assumed system "result of estimation" is two elements vector - not, as in previous cases - three elements vector. This is equivalent to reduced-order state observer. Class implementation is based on 2D array of *matrix* class objects. First array index determines network number (state variable), second determines network layer number. Every network contains equal number of layers, use the same sets of activation function (sigmoid for hidden layer, pure linear for output layer) and takes identical input vectors. Number of neurons in each layers can be different between networks. Fig. 4. shows simplified block diagram of neural estimator for two-mass system.

The implementation of multilayer, one-way, sigmoidal neural network was based on [18]. Code in C++, shown in Listing 5.-6., presents body of the member function *update()* in *neural_estimator* class and examples of definition of pointer to class object and call function *udpate()*.

Listing 5. Body of the *upadate()* function, *neural_estimator* class

```
vector<double>
neural_estimator::update(double u, double y)
{
        // data processing
        vector<double> in = data_processing(u, y);

        // result vector
        vector<double> out;

        for(unsigned i = 0; i < net_number; i++)
        {
                out = in;

                // neural programs
                for(unsigned j = 0; j < layer_number; j++)
                {
                    out = W[i][j]*out + b[i][j];
                    out = activation_fcn(j, out);
                }

                x_act(i) = out(0);
        }

        // current state
        return x_act;
}
```

Listing 6. An examples of definion of pointer to *neural_estimator* class object and call member function *update()*

```
// array of state observers
state_observer** observer = new state_observer*[3];

// class contructor
observer[2] = new neural_estimator(net_n, layer_n, delay, W, b);

// result vector
vector<double> x_e(2, 0.0);

// neural estimator update
x_e = observer[2]->update(input, output);
```

Functions *data_processing()* and *activation_fcn()* are private member functions. First one forms input vector out of current and previous samples of input u and output y. Number of previous samples used by observer is determined by class constructor third argument - *delay*. Second function is layer activation function, supporting *vector* class objects. The last two arguments of class constructor - *W* and *b* shown in an example - are 2D arrays of *matrix* class objects containing respectively network weights and biases. Sizes of those arrays are determinate by class constructor arguments *net_n* and *layer_n*. Implementation do not provide training algorithms - weights and biases are not modified during estimator iterations.

With a common base class for all estimation method, presented implementation has a significant advantage. For each class call *update()* function is identical. On Fig. 5. class inheritance diagram is shown. Diagram was created by Doxygen documentation generator.

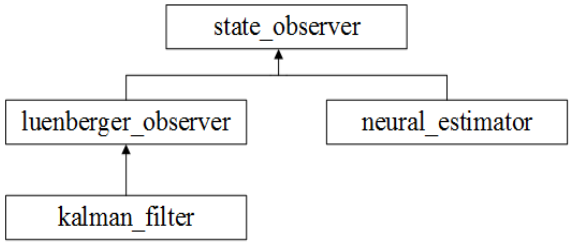

Fig.5. Inheritance diagram for state observers algorithms classes

Simulation results

The electrical drive, shown in Fig. 1., was simulated in MATLAB/Simulink environment. Comparative implementation of state observers algorithms was done in MATLAB script language. In Fig. 6. plots of input signals are shown - reference angular velocity $\omega_{1\,ref}$ and load torque T_l.

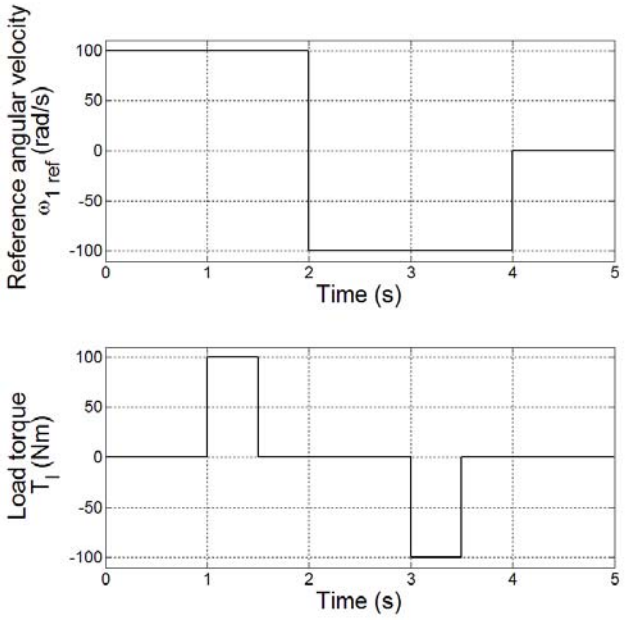

Fig.6. Control system input signals: reference angular velocity (discrete-time) and load torque (continuous-time)

In Fig.7. plots of the responses of the control system are shown - reference motor torque $T_{e\,ref}$ and actual angular velocity on the motor side ω_1. Velocity oscillations are caused by nature of analyzed system which was described in detail in previous work [11], [13]. Those signals are input data for estimation algorithms. Sample time τ_s is equal to 100 μs. Simulation step size is equal to 1/100 of this value - 1 μs.

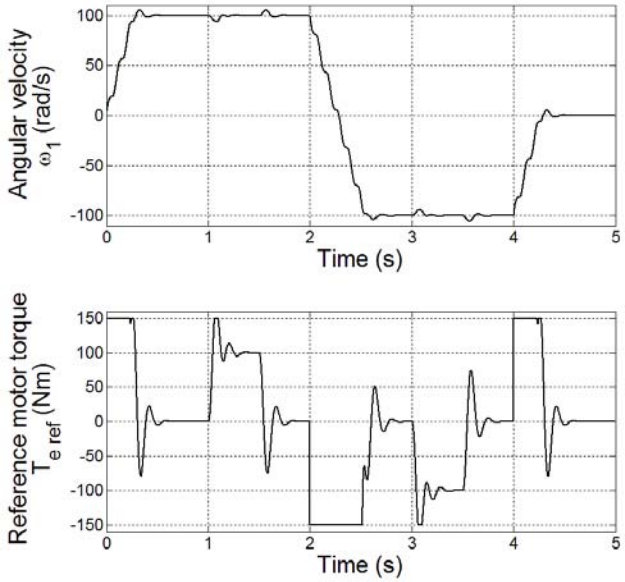

Fig. 7. Response of the control system: angular velocity on the motor side (discrete-time) and reference motor torque (discrete-time)

Plots shown in Fig. 8. to 10. are results of simulation of electrical drive in MATLAB/Simulink environment compared with results of discrete estimation algorithms implemented in C++, in VisualDSP++ 5.0 environment. Program was executed on digital signal processor ADSP-21369, member of the SHARC family.

For simulation purposes, discrete Luenberger observer gain was determined. Dynamics of the estimation error is equivalent to a dynamics of the Butterworth filter with time constant equal to 10 ms. In Fig. 8 results of simulation and estimation of unknown state variables are shown.

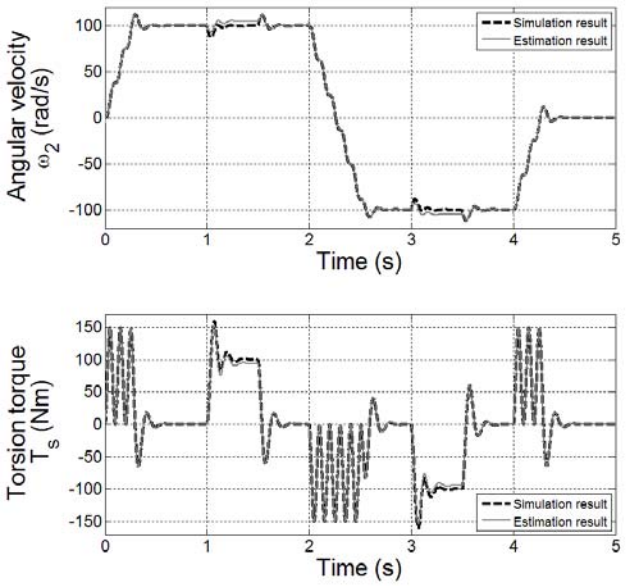

Fig.8. Estimation results - discrete full-order Luenberger state observer

Kalman filter works in system with additive white Gaussian noise. For simulation purposes, to measured angular velocity ω_1 was added white noise of standard deviation equal to 1 rad/s. In Fig. 9 results of simulation and estimation of unknown state variables are shown. The choice of matrices $\mathbf{Q}_d$ and $\mathbf{R}_d$ was experimental.

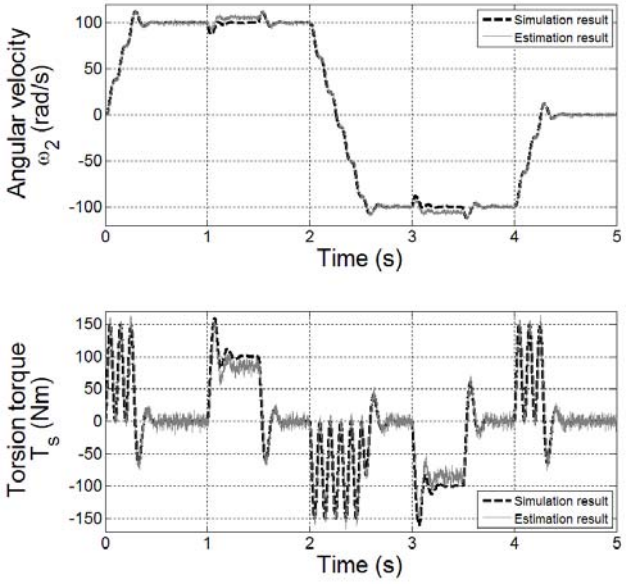

Fig.9. Estimation results - Kalman filter

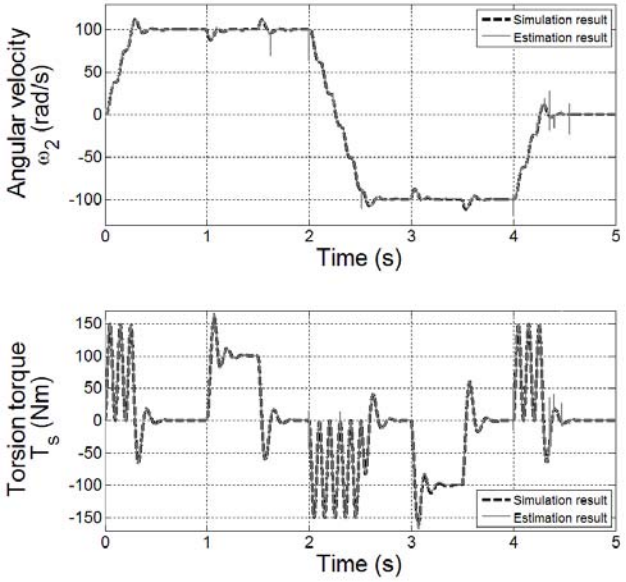

Fig.10. Estimation results - neural estimator

Both Luenberger observer and Kalman filter underestimate angular velocity ω_2 and overestimate torsion T_s during load. This is due to the fact that mathematical model of the system do not contain information on load torque. This effect is consistent with the assumptions of algorithms. Solution to this problem could be extension of observer structure shown in [19] or [20]. Objective-oriented technique will facilitate such modifications.

Based on [18], for simulation purposes, neural networks of neural estimator had following structure: 7 neurons of first hidden layer (input layer), 8 neurons of second hidden layer and 1 neuron of output layer. Input vector consist of current and 3 previous samples of reference motor torque $T_{e\,ref}$ and angular velocity ω_1. Networks training was based on

simulation data - CHIRP signal as reference motor torque and responses of the control system to starting-up, braking and reversing. In Fig. 10 results of simulation and estimation of unknown state variables are shown.

Conclusions

The article presents objective-oriented implementation of three state observers methods: Luenberger observer, Kalman filter and neural estimator. Implementation was dedicated to digital signal processor. Results of estimation are in all cases consistent with assumptions and are identical with result of comparative implementation done in MATLAB/Simulink environment. Luenberger observer is characterized by desired dynamics of estimation error. Kalman filter properly performs estimation of state vector under measurement noise. Neural estimator generalize training data and estimate state variable properly also during load.

Encapsulation of estimation methods was a success - all necessary observers parameters and sub-algorithm are classes members. Object-oriented techniques allow to use both convenient syntax and hardware implementation of operations on floating-point number arrays.

Further development of systematized object-oriented library in C++ for electrical drive is planned. Main goal is to enable easy collaboration between DSP and MATLAB/Simulink environment, primarily in terms of neural-network-based estimator.

Acknowledgment

The presented results of research were carried out under the theme No. 04/45/DSPB/0135, which was funded by the Polish Ministry of Science and Higher Education.

***Authors**: dr inż. Dominik Łuczak, Politechnika Poznańska, Instytut Automatyki i Inżynierii Informatycznej, ul. Piotrowo 3a, 60-965 Poznań , E-mail: dominik.luczak@put.poznan.pl,*

inż. Adrian Wójcik, Politechnika Poznańska, E-mail: adrian.wojcik@student.put.poznan.pl

REFERENCES

[1] Szabat K., Tran-Van T., Kaminski M. , A Modified Fuzzy Luenberger Observer for a Two-Mass Drive System, *IEEE Trans. Ind. Inform.*, 11 (2015), n.2, 531–539

[2] Kamiński M., Adaptacyjny-neuronowy obserwator Luenbergera zastosowany w estymacji zmiennych stanu układu dwumasowego, *Przegląd Elektrotechniczny*, 90 (2014), nr 6

[3] Drozdz K., Janiszewski D., Szabat K., Application of fuzzy Kalman filter in adaptive control structure of two-mass system, 16th International Conference and Exposition in Power Electronics and Motion Control (PEMC), 2014 575–578

[4] Dróżdż K., Szabat K., Adaptacyjne sterowanie układu dwumasowego z wykorzystaniem rozmytego filtru Kalmana," *Przegląd Elektrotechniczny*, 90 (2014), nr 6,

[5] Kamiński M., Estymacja zmiennych stanu układu dwumasowego za pomocą modeli neuronowych, *Pr. Nauk. Inst. Masz. Napędów Pomiarów Elektr. Politech. Wroc. Stud. Mater.*, 69 (2013), n.33, 222–238

[6] Luczak D., DSP implementation of electric drive control system, 8th International Symposium on Communication Systems, Networks Digital Signal Processing (CSNDSP), 2012, 1 –3

[7] Muszynski R., Deskur J., "Ds," *I*amping of Torsional Vibrations in High-Dynamic Industrial Drive, *IEEE Trans. Ind. Electron.*, 57 (2010), n.2, 544–552

[8] Villwock S., Pacas M., Application of the Welch-Method for the Identification of Two- and Three-Mass-Systems, *IEEE Trans. Ind. Electron.*, 55 (2008), n.1, 457–466

[9] Deskur J., Pajchrowski T., Zawirski K., Speed controller for a drive with complex mechanical structure and variable parameters, 16th International Power Electronics and Motion Control Conference and Exposition (PEMC 2014),762–767

[10] Saarakkala S.E., Hinkkanen M., Identification of two-mass mechanical systems using torque excitation: Design and experimental evaluation, 2014 International Power Electronics Conference (IPEC-Hiroshima 2014 - ECCE-ASIA), 2489–2496

[11] Luczak D., Mathematical model of multi-mass electric drive system with flexible connection, in *Methods and Models, 19th* International Conference On Automation and Robotics (MMAR 2014), 590–595

[12] Bruschetta M., Picci G., Saccon A., A variational integrators approach to second order modeling and identification of linear mechanical systems, *Automatica*, 50 (2014), n.3, 727–736

[13] Szabat K., Orlowska-Kowalska T., Vibration Suppression in a Two-Mass Drive System Using PI Speed Controller and Additional Feedbacks mdash;Comparative Study, *IEEE Trans. Ind. Electron.*, 54 (2007), n.2, 1193–1206

[14] Luenberger D.G., An introduction to observers, *IEEE Trans. Autom. Control*, 16 (1971), n.6, 596-602

[15] Radisavljevic-Gajic V., Linear observers design and implementation, Conference of the American Society for Engineering Education (ASEE Zone 1), 2014 Zone 1, 1–6

[16] Welch G., Bishop G., An Introduction to the Kalman Filter," 2006. [Online] Available: https://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf

[17] Cazan I., Kalman filters, 2011. [Online]. Available: http://www.colby.edu/math/program/honorsprojects/2011-Cazan-Honors.pdf

[18] Bose B.K., Neural Network Applications in Power Electronics and Motor Drives, *IEEE Trans. Ind. Electron.*, 54 (2007), n.1, 14–33

[19] Serkies P.J., Szabat K., Adaptacyjna struktura sterowania z predykcyjnym regulatorem prędkości dla układu napędowego z połączeniem sprężystym, *Pr. Nauk. Inst. Masz. Napędów Pomiarów Elektr. Politech. Wroc. Stud. Mater.*,(2011), n.31, 320–330

[20] Szabat K., Model obserwatora zmiennych stanu dla układu z nieliniowym wałem mechanicznym, *Pr. Nauk. Inst. Masz. Napędów Pomiarów Elektr. Politech. Wroc. Stud. Mater.*, vol. 63 (2009), n.29, 355–368