Dominik ŁUCZAK*
Adrian WÓJCICK*

OBJECT-ORIENTED DSP IMPLEMENTATION
OF NEURAL STATE ESTIMATOR
FOR ELECTRICAL DRIVE WITH ELASTIC COUPLING

The study presents results and procedure of object-oriented and test-driven implementation of neural-network-based state estimator. The presented algorithm has been developed for estimation of the state variables of the mechanical part of electric drive with elastic coupling. Estimated state variables – load speed and shaft stiffness torque – can be used in speed control process for reducing mechanical vibrations of working machine. The basic objective was to create a simple, extensible and readable program code, performing the task of state estimation of the considered system. The target platform is a DSP (Digital Signal Processor) from SHARC (Super Harvard architecture Single-Chip Computer) family, which allows for hardware acceleration of matrix operations. The IDE (Integrated Development Environment) available for the selected platform made it possible to write program in C++. The usage of UML (Unified Modelling Language) in the development of control software was discussed.

KEYWORDS: state observers, DSP, object-oriented programming technique C++, electric drive, elastic coupling, dual-mass system

1. MATHEMATICAL MODEL OF CONTROL SYSTEM

1.1. Model of electrical drive

The control system discussed in this paper is description of practical example of embedded control software development task. In this example plant is electrical drive working in cascade control structure (Fig. 1). Mechanical part of drive is characterized by elastic connection between motor shaft and working machine. In literature this type of mechanical systems is typically modeled by dual-mass system [6, 7, 3]. Dual–mass system consist of two rigid bodies with common axis of rotation. The relative angular displacement between bodies creates stiffness torque. On the basis of D’Alembert’s principle can be derived
mathematical model of system in state space form (1) [3, 1]. In many application
only readily available state variable motor speed $\omega_1$. 
\[
\begin{bmatrix}
\dot{\omega}_1(t) \\
\dot{\omega}_2(t) \\
\ddot{T}_l(t)
\end{bmatrix} = 
\begin{bmatrix}
\frac{b_1}{J_1} & \frac{b_1}{J_1} & -\frac{1}{J_1} \\
\frac{b_2}{J_2} & -\frac{b_2}{J_2} & \frac{1}{J_2} \\
k & -k & 0
\end{bmatrix}
\begin{bmatrix}
\omega_1(t) \\
\omega_2(t) \\
T_\ell(t)
\end{bmatrix} + 
\begin{bmatrix}
\frac{1}{J_1} & 0 & 0 \\
0 & -\frac{1}{J_2} & 0 \\
0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
T_c(t) \\
T_i(t)
\end{bmatrix}
\]
(1)

where: $\omega_1(t)$ [rad/s] – motor speed (measurable state variable), $\omega_2(t)$ [rad/s] –
load speed (unmeasurable state variable), $T_c(t)$ [Nm] – stiffness torque
(unmeasurable state variable), $T_\ell(t)$ [Nm] – motor electromechanical torque
(input), $T_i(t)$ [Nm] – load torque (input), $J_1$ [kg·m²] – moment of inertia on
motor side (parameter), $J_2$ [kg·m²] – moment of inertia on load side (parameter),
$b$ [Nm·s/rad] – coefficient of viscous friction (parameter), $k$ [Nm/rad] – stiffness
(parameter).

Electromagnetic part of the motor together with power inverter and torque
controller can be modeled as simple first-order inertial system with transport
delay and input zero–order hold interpolation (2) [1]
\[
G_T(s) = \frac{T_c(s)}{T_\ell \text{ref}(s)} = \frac{1}{\tau_T} \frac{1}{s} \frac{e^{-\tauTd}}{\tau_s + 1} 
\]
(2)

where: $\tau_T$ [s] – torque control loop time constant (parameter), $\tauTd$ [s] – torque
control loop transport delay (parameter), $\tau_s$ [s] – control system sample time
(parameter).

Fig. 1. Control system – electrical drive in cascade torque / speed control structure

Adopted model is sufficient only if torque control meets the following
requirements:
1. Active current control is optimal in IAE (Integral Absolute Error) or ISE
(Integral Square Sense) sense.
2. Active current is proportional to electromechanical torque. This statement is
equivalent with requirement of ideal compensation of parasitic load and any
nonlinear behavior.
With this representation in further work type of electrical motor and inverter structure can be left aside.

Speed measurer is digital device, therefore its input can be characterized with formula (3). Taking this detail into account allows for more accurate simulation test of embedded control software.

\[
\omega_i(k) = q \cdot \frac{1}{2} \text{sat} \left( \omega_i(t) \cdot \delta (t - k \tau_s), \omega_{\text{max}} \right) + \frac{1}{2}
\]

where: \(\omega_i(k) \text{ [rad/s]}\) – digital value of motor speed in time k (output), \(q \text{ [rad/s]}\) – speed quant (parameter), \(\omega_{\text{max}} \text{ [rad/s]}\) – torque control loop transport delay (parameter), \(\tau_s \text{ [s]}\) – control system sample time (parameter).

1.2. Speed control algorithm

In discussed system speed controller is based on digital signal processor (DSP) and performs two main tasks:
1. Load speed control by PI controller with additional proportional state feedback (Fig. 2).
2. Mechanical system state variable estimation used in control procedure (described in 1.3).

Tuning of discrete PI controller (in form (4)) and feedback gain is based on [6]. Due to the need to develop reliable software tests, in this paper two controller configurations are used. First one uses only stiffness torque feedback (5), second one only load speed (6).

\[
G_r(z) = K_p + K_i \cdot \frac{\tau_s}{z - 1}
\]

\[
k_1 = \frac{4 \left( \xi_{\text{ref}} \right)^2}{\tau_2} \tau_1 - 1 \\
k_2 = \frac{\tau_1 + \tau_2}{\tau_1 \left( 4 \left( \xi_{\text{ref}} \right)^2 + 1 \right)} - 1 \\
K_i^h = \frac{\tau_1}{\tau_2 \tau_c} \\
K_i^2 = \frac{\tau_1 \left( 1 + k_1 \right)}{\tau_2 \tau_c}
\]

where: \(k_1 [-]\) – stiffness torque \(T_s\) feedback gain (parameter), \(k_2 [-]\) – load speed \(\omega_i\) feedback gain (parameter), \(K_p [-]\) – speed controller proportional gain (parameter), \(K_i [1 / s]\) – speed controller integral gain (parameter), \(\xi_{\text{ref}} [-]\) – reference damping factor of closed-loop speed control system (parameter), \(\tau_i = \alpha_N J_i \text{ [s]}\) – motor mechanical time constant (parameter) (normalized momentum of inertia), \(\tau_s = \alpha_N J_2 \text{ [s]}\) – load mechanical time constant (parameter) (normalized momentum of inertia), \(\tau_c = 1/(\alpha_N k) \text{ [s]}\) – connection stiffness time constant (parameter) (normalized stiffness), \(\alpha_n = \omega_N T_{\text{eN}} \text{ [rad/s / Nm]}\) –
normalization coefficient (ratio of nominal motor speed and nominal motor torque) (parameter).

Fig. 2. Speed controller – PI controller with additional state feedback

Selected controller settings are intend to dampen mechanical vibrations on working machine side. Use of additional state feedback adds a degree of freedom and allows selection of closed-loop damping factor [6, 7]. Development of state estimation method is therefore a vital task, improving control quality.

1.3 State estimation algorithm

Software development procedure is shown on the basis of state estimation algorithm, used in example. In previous work three state estimation methods were implemented and tested: discrete full-order Luenberger state observer, Kalman filter and neural–network–based estimator [1]. In this paper neural estimator was chosen due to regular (and therefore scalable) structure of algorithm. Also it is well-suited to object–oriented implementation.

Neural estimator for dual-mass system was based on [4, 5]. Estimator inputs are: measured state variable (motor speed $\omega_1$) and reference value (motor torque $T_{e \text{ref}}$). Output signals are unmeasurable state variables: load speed $\omega_2$ and stiffness torque $T_s$ (Fig. 3). In general case number of output is algorithm parameter. Because of static connection between speed controller input and output via state estimator, state variables estimates are delayed by single sample time.

Estimator algorithm consists of three main elements:
1. Data preprocessing – procedure of creating input vector for artificial neural networks consisting of three steps:
   1.1. Normalization to $<-1, 1>$ range. Depends on measurement and control range, therefore speed normalization gain $n_\omega$ and torque normalization gain $n_T$ are parameters of the algorithm.
1.2. Low pass filtering. Optional step.

1.3. Combining current input samples with previous samples to a single input vector (7). The numbers of previous speed samples $d_\omega$ and torque samples $d_T$ are parameter of the algorithm.

\[
\mathbf{x}_0 = \left( n_\omega \cdot \left[ \omega_i(k) \cdots \omega_i(k-d_\omega) \right] \right) \sqcup \left( n_T \cdot \left[ T_e^{ref}(k) \cdots T_e^{ref}(k-d_T) \right] \right) \tag{7}
\]

2. Array of static, off-line trained, multilayer, sigmoidal feedforward neural networks. Every network perform estimation of single state variable so for dual-mass system, array consist of two separate networks with scalar outputs. All networks get common input vector prepared in preprocessing procedure. Therefore number of networks and their input sizes are defined by introduced parameters. Another – generalized – parameter is actual internal structure (hidden layer sizes) of networks. Obtaining neural networks via performing estimation task requires following initial actions:

2.1. Selection of network input size and internal structure. All networks have common input size and scalar output.

2.2. Generating / obtaining training data vectors – sets of inputs and corresponding outputs of estimator.

2.3. Performing networks training (optimization) and validation with selected method.

2.4. Performing networks estimation quality verification in closed control system.

Described actions allows to acquire ordered sets of constant matrices, which must be than applied to neural network algorithm (8) [8, 5]. There is no need to performs those actions directly in embedded system.

\[
\mathbf{x}_{i+1}(k) = f_i \left( W_{i+1} \cdot \mathbf{x}_i(k) + b_{i+1} \right) \land i \in \{0, \ldots , L\} \tag{8}
\]

where: $\mathbf{x}_i(k)$ – output vector of $i$–th network layer and input vector of $(i-1)$–th network layer in time $k$ ($\mathbf{x}_0$ is network input, $\mathbf{x}_L$ is network output), $W_i$ – coefficient matrix of $i$–th network layer, result of offline training
procedure, $\mathbf{b}_i$ – bias vector of i–th network layer, $f_i$ – activation function i–th network layer, $L$ – number of layers in network (both hidden and output).


2. NEURAL STATE ESTIMATOR DEVELOPMENT PROCEDURE

2.1. Object-oriented technique and test driven development

Object-oriented technique is programming paradigm based on a concept of grouping code together with state the code modifies. Today it is dominant technique in overall software development – according to the software engineering market analysis today's most popular programming language is Java – language strongly focused on object–oriented programming. The next places in the ranking occupy the languages C and C ++ [9]. Both of the latter are popular in embedded software. Manufactures of popular microcontrollers and DSPs, such as PIC32, STM32 or SHARC, increasingly provides C++ compilers for their platforms. C++ in contrast to C, allows the use of class–based object–oriented programming. Therefore, you should expect the growing role of object–oriented approach in the design of control systems [2].

Test driven development is type of software development process based on cycles of two tasks. First one is implementation of specific automatic test derived directly from software requirements. Second one is implementation of actual code. When implemented code pass test cycle is over and development process should proceed to next requirement [2]

Description of used programing and design technique indicates, that first step of neural state estimator is gathering software specification. Requirements results from simulation model of electrical drive. The use of object–oriented approach make it easy to use Unified Modeling Language (UML), which is semiformal way to represent models of software structure and behavior. Since version 2.0, published in 2005, standard pays greater importance to embedded system [2]This modeling language is theoretically independent from programing language, but selection class–based language with mechanisms such as inheritance, aggregation, templates, operators overloading etc., make it simple and direct.

2.2. From specification to unity tests

The proposed development procedure starts with definition of software requirements specification i.e. set of documented requirements to satisfied by implemented software. Table 1 shows three general requirements that results in significant consequences and determines the direction of tests and implementation.
Table 1. Neural state estimator requirements specification

<table>
<thead>
<tr>
<th>Objective</th>
<th>Input (required information)</th>
<th>Output (produced information)</th>
</tr>
</thead>
</table>
| 1. | 1. Implementation of neural state estimator:  
1.1. Object–based implementation  
1.3. Suitable for dual–mass system state estimator | Target platform  
Detailed algorithm description  
Performance time requirements  
Reliable neural network code | NeuralEstimator class tests results |
| 2. | 2. Implementation of feedforward neural network  
2.1. Object–based implementation  
2.2. Modular, resizable structure (aggregation)  
2.3. Simple and readable code of neural procedure | Detailed algorithm description  
Reliable matrix / vector operation code | Network class tests results  
NetworkLayer class tests results |
| 3. | 3. Implementation matrix operation library  
3.1. Object–based implementation  
3.2. Multi–type support (template)  
3.3. Simple and readable code, recalling mathematical notation  
3.4. Use of hardware matrix operations | Detailed numerical accuracy description  
Detailed list of operators | Mat template class test results  
Vec template class test results |

First objective is most general one. As target platform was selected ADSP–21369 from SHARC family. Its inputs requires also algorithm description (provided in section 1), performance test requirement (max. performance time for well–trained network shall be least than ¼ of sample time) and reliable neural network code, to build upon. Second objective results from input of previous one. Third objective results from inputs of second one. Specification of matrix operation assumed accuracy to 6 decimal place for double-precision matrices. Actual code writing start with matrix class template.

Definition of software specification allows to determine relation between algorithm modules and layers. Those relation can be expressed in UML structural class diagram (Fig. 4). For neural state estimation use case diagram, describing behavior of software, is very simple and results directly from control system model.

Outputs of every objective are results of automated unity test of module. Therefore next step in development procedure is implementation of unity testes for Mat and Vec template classes. Example of operator unity test is presented in listing 1. Automated unity test in C++ are (typically short and simple) peace of
codes containing call of tested procedure, reading from reference source or computing in alternative way correct result and comparing them with form of assert function. There are many tools for unity testing in form of separate frameworks (CppUnit, CUTE, UnitTest++) or as a part of it (BOOST). In presented examples no additional libraries are used – testing is of course possible in plain C++ language with standard libraries.

If all methods of class templates Mat and Vec are tested for double type (only for this type accuracy was specified), development process should move to implementation of neural network tests (Listing 2 – 3).

Last step in test implantation process is set of unity test for complete NeuralEstimator class (Listing 4). Only first objective has time requirements, therefore only in neural estimator tests contains performants time check. After code pass all those test discussed is complete but it is strongly advised to test integration with other modules of software (i.e. PI controller).
According to test driven approach, tests should precedes implementation. Listing 5 presents sample code of matrix multiplication operator of Mat class template for double type. Procedure is based on target platform matrix operation library [10]. Listings 6 – 7 presents compression of neural procedure implemented with and without Mat and Vec class templates.

Code in listing 6. is far more readable – syntax reminds natural mathematical notation. But underlying operations based on local objects – procedures of allocation and deallocation of memory is additional overhead for static procedure like offline trained neural network. Functional approach will be faster but is less readable and modifiable. This leads to important insights concerning software quality – produced code should be evaluated only in criteria of
fulfilling or not specification requirements. Only object–oriented implementations meets our criteria.

```cpp
template<> inline
Mat<
double>
Mat<double>::operator*(const
double& rhs)
{
    /* Local variable – result buffer */
    unsigned
    int
    new_rows = rows;
    unsigned
    int
    new_cols = rhs.cols;
    Mat<long
double>
    result(new_rows, new_cols);
    /* matrix.h library double matrix multiplication */
    matmmlt(result.mat, mat, rhs.mat, new_rows, cols, new_cols);
    return result;
}
```

Listing 5. Mat multiplication operator definition

```cpp
... return f(w * x + b) ...
matmmlt(wx, w, x, w_rows, w_cols, x_size)
matmadd(y, wx, b, x_size, 1);
return f(y);
...
```

Listing 6. Neural procedure – object–oriented approach

Listing 7. Neural procedure – functional approach

### 3. SIMULATION RESULTS

#### 3.1. Simulation environment and reference implementation

After whole development procedure, neural state estimator is ready to be verified by compression with reference implementation. Verification is done by carrying out two simulation with identical parameters and inputs in two software environments. First one is simulation tool for ADSP–21369 processor provided with CrossCore Embedded Studio. Second one is MATLAB with Simulink and Neural Network Toolbox. MATLAB is also source of reference results for all conducted unity tests. In Tables 2 – 3 summarize all significant parameters of simulations.

Both neural networks had identical structure {6 – 8 – 1}, which mean 6 inputs, 8 neurons in hidden layer and 1 neuron in output layer / 1 output. Simulation sample time was chosen as \( \tau_s = 500 \mu s \).

#### 3.2. Simulation results compression

In Fig. 5 presents response of neural estimator executed on DSP simulator, relative and absolute error of response in relation to MATLAB reference implementation. For all samples value difference is less than 0.1% of estimated
value. Maximal performance time was 101 µs which meets requirements (consistent with the conclusion of the tests).

Table 2. Parameters of speed control system

<table>
<thead>
<tr>
<th>( n_N )</th>
<th>( T_{eN} )</th>
<th>( J_1 )</th>
<th>( J_2 )</th>
<th>( k )</th>
</tr>
</thead>
<tbody>
<tr>
<td>[rpm]</td>
<td>[Nm]</td>
<td>[kg m(^2)]</td>
<td>[kg m(^2)]</td>
<td>[Nm / rad]</td>
</tr>
<tr>
<td>1500.0</td>
<td>3.18</td>
<td>0.0041</td>
<td>0.0041</td>
<td>7.7939</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( b )</th>
<th>( \tau_T )</th>
<th>( \tau_{id} )</th>
<th>( K_p )</th>
<th>( K_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Nm \cdot s / rad]</td>
<td>[ms]</td>
<td>[ms]</td>
<td></td>
<td>[1 / s]</td>
</tr>
<tr>
<td>0.0</td>
<td>3.0</td>
<td>2.0</td>
<td>25.0 (( k_1 ))</td>
<td>384.6 (( k_1 ))</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>20.4 (( k_2 ))</td>
<td>256.4 (( k_2 ))</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( k_1 )</th>
<th>( k_2 )</th>
<th>( \omega_{ref} )</th>
<th>( q )</th>
<th>( n_{max} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>[-]</td>
<td>[-]</td>
<td>[-]</td>
<td>[rad/s]</td>
<td>[rpm]</td>
</tr>
<tr>
<td>1.0</td>
<td>-0.3333</td>
<td>0.7</td>
<td>1.256</td>
<td>3000.0</td>
</tr>
</tbody>
</table>

Table 3. Parameters of neural state estimator

<table>
<thead>
<tr>
<th>( N_s )</th>
<th>( n_{so} )</th>
<th>( n_T )</th>
<th>( d_{so} )</th>
<th>( d_T )</th>
</tr>
</thead>
<tbody>
<tr>
<td>[-]</td>
<td>[1 / rpm]</td>
<td>[1 / Nm]</td>
<td>[-]</td>
<td>[-]</td>
</tr>
<tr>
<td>2</td>
<td>1 / 3000</td>
<td>1 / 10</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Fig. 5. Load speed estimation – implementation compression results

4. CONCLUSION

Because of detailed unity test of underling procedures of neural estimator, obtained results were correct after first try. There were virtually no possibility of undetected bug. It can be concluded that presented embedded software development procedure can be adapted and implemented in control system.
Furthermore, procedure is consistent with software life cycle presented in IEC 61508–3 software safety integrity level standard. Proposed approaches result in reliable and therefore safe control software. Presented software is part of object–oriented embedded control framework developed by authors, designed primarily for electrical drive control systems.

LITERATURA


(Received: 24. 02. 2017, revised: 28. 02. 2017)