CZECH TECHNICAL UNIVERSITY IN PRAGUE

FACULTY OF MECHANICAL ENGINEERING

Department of Instrumentation and Control Engineering

Master's Thesis

Solution for Adaptive Control of Coupled Motor System with Flexible Joint

Author: ILIA ALI-OGLY

Academic Year: 2015/2016

Supervisor: doc. Ing. Ivo Bukovsky, Ph.D.

Czech Technical University in Prague
Dept. of Instrumentation and Control Engineering

Faculty of Mechanical Engineering
Academic year: 2015/2016

DIPLOMA THESIS SPECIFICATION

Name of student: Ilia ALI OGLY

Field of study: Instrumentation and Control Engineering

Title:
Solution for Adaptive Control of Coupled Motor System with Flexible Joint

Diploma thesis guidelines:

1) Review recently published adaptive algorithms for identification and control of single-input single-output systems.
2) Investigate and propose a low-cost HW and SW platform for the coupled motor system with flexible joint in lab 111.
3) Implement your HW and SW solution for real time data measurement and visualisation. Measure proces data (input and output)
4) Investigate linear and nonlinear adaptive models for adaptive plant identification (LNU, QNU, GD)
   a. with step-by-step gradient learning algorithm for real-time identification,
   b. with batch Levenberg-Marquardt learning algorithm for offline plant identification.
5) Investigate linear and nonlinear adaptive models for adaptive plant control (LNU, QNU, GD, LM)
6) Implement the complete adaptive control solution to the laboratory system and investigate the real-world aspects and their influence to the method (sampling, noise,…).

Extent of graphical work: max. 50 %

Extent of cover letter: min. 50 pages + appendix

List of literature:

[1] Ivo Bukovský, Matouš Sláma, "Platforma pro návrh adaptivního regulátoru, implementaci a testování algoritmů". Workshop, ARTEP 2014, Slovensko, 2014.

[2] P. Benes and I. Bukovsky, "On the Intrinsic Relation between Linear Dynamical Systems and Higher Order Neural Units," in *Intelligent Systems in Cybernetics and Automation Theory*, R. Silhavy, R. Senkerik, Z. K. Oplatkova, Z. Prokopova, and P. Silhavy, Eds. Springer International Publishing, 2016. (Accepted Paper)

[3] Peter Mark Benes, Miroslav Erben, Martin Vesely, Ondrej Liska, and Ivo Bukovsky, "HONU and Supervised Learning Algorithms in Adaptive Feedback Control," in *Applied Artificial Higher Order Neural Networks for Control and Recognition*, IGI Global, 2016. (in print)

[4] M.M. Gupta, Jin L., and N. Homma: *Static and Dynamic Neural Networks: From Fundamentals to Advanced Theory*, IEEE Press and Wiley-Interscience, published by John Wiley & Sons, Inc., 2003, ISBN 0-471-21948-7

[5] Bukovsky, I., Homma, N., Smetana, L., Rodriguez, R., Mironovova M., Vrana S.,: "Quadratic Neural Unit is a Good Compromise between Linear Models and Neural Networks for Industrial Applications", *ICCI 2010 The 9th IEEE International Conference on Cognitive Informatics*, Tsinghua University, Beijing, China, July 7-9, 2010.

[6] Zhuldyz Assylova: "*Comparison of Neural Network Models for Approximation of Pneumatic Muscle Actuator*", bachelor's thesis, CTU in Prague, FME, Department of Instrumentation and Control Engineering, Division of Automatic Control and Engineering Informatics, defended in 2013.

[7] more shall be found as a review task 1)

Diploma thesis supervisor: Doc. Ing. Ivo Bukovský, Ph.D.

Submission date: 12. 4. 2016

Delivery date: 15. 6. 2016

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta strojní
-71-

v.z. [illegible]

[illegible]

Head of Dept: doc. Ing. Jan Chyský, CSc. *Dean:* prof. Ing. Michael Valášek, DrSc.

Prague, 5. 4. 2016

Annotation List

Authors Name:	Ilia Ali-Ogly
Name of Master's Thesis :	Solution for Adaptive Control of Coupled Motor System with Flexible Joint
Year:	2016
Field of study:	Instrumentation and Control Engineering
Department:	Department of Instrumentation and Control Engineering
Supervisor:	Doc. Ing. Ivo Bukovský, Ph.D.

Bibliographical data:	Number of pages	90
	Number of figures	34
	Number of tables	7
	Number of attachments	1

Keywords: Neural Networks, system, state-space, flexible joint, analysis, identification, control, adaptive algorithm, model, implementation, Python, LabJack, Simulink, non-linearity, application, real-time, visualization, PID, neuro-controller, training, tuning, solution.

Statement

I declare that I have worked out this thesis independently assuming that the results of the thesis can also be used at the discretion of the supervisor of the thesis as its co-author. I also agree with the potential publication of the results of the thesis or its substantial part, provided I will be listed as the co-author.

In Prague:______________________ Signature:_______________________

Acknowledgments

I would like to express great thanks to my coordinator and supervisor **Doc. Ing. Ivo Bukovský, Ph.D.** , whose knowledge and guidance has helped me by continuous support and professional advices to get through the hard process while preparing this thesis. I would also like to thank **Ing. Beneš Petr, Ph.D.** for his valuable advice and expertise.

I also thank **prof. Ing. Vyhlídal Tomáš, Ph.D.** ,**prof. Ing. Uhlíř Ivan, DrSc.** , **Ing. Martin Novák, Ph.D.** and **Ing. Trnka Pavel** for the opportunity to work with them, their help and cooperation through my work and valuable suggestions.

Abstract

This master's thesis refers to analysis, design and implementation of adaptive neural algorithms, with low-cost hardware setup and open-source software, aimed to control input signal to handle non-linearity of coupled motor system with flexible joint. Three different neural network architectures and their static and dynamic versions were derived, programmed, implemented and tested via designed and programmed computational program developed for realization of whole experimental process starting from data acquisition ending with appliance of artificial neural networks for real process control by means of LabJack- measurement and automotive device.

Table of Contents

List of Figures

List of Tables

1 Introduction

Nowadays numerous industrial processes are requiring high quality automatic control. In accordance with process specification, if behavior of process is not within permissible limits, it could lead to serious consequences. Therefore, my objective in this thesis is to improve controlled process characteristics by implementing neural algorithms. I am beginning this thesis form description of process form physical and mathematical point of views following with virtual analogue of coupled motor system built in Simulink environment[1]. Further I am proceeding to description of various methods of process control with strong emphasis on artificial intelligence approach, which is based on adaptive algorithms for system identification and control of single-input single-output systems. Techniques of new methods are explained in conformity with review summary of recently published works in this field. As the next step I am introducing hardware and software platform for working with coupled motor system followed with brief description of main codes for interaction with this hardware. Then I am proposing solution with given HW and SW for real-time data measurement, analysis and control by means of my specially designed and programmed software application for these purposes, which is an extension of existing computational program. In the following chapters I am investigating linear and nonlinear adaptive models with step-by-step gradient descent learning algorithms for real-time and batch Levenberg-Marquardt learning algorithm for offline plant identification and control. Furthermore, I am experimentally testing and comparing capability of algorithms to efficient model approximation and control. At the end I am discussing results of implementation of the complete adaptive control solution to the laboratory system and influencing aspects to the applied methods. Then in the discussion section I am describing neural network control potential supported with figures and schematics.

2 System Description

The dynamic system Fig. 1 and Fig. 2 consists of tachometer and two direct-current electric motors, shafts of which are coupled with flexible joint FJ. First motor M performs a function of energy source or a leading motor. The second motor performs a function of a generator/load L-target object, which has to be controlled and measured by means of tachometer T, which is coupled to the generator and measuring rotations. The system is driven by a direct-current input U_M, the output voltage **U_T** is measured by tachometer, which is directly proportional to the speed of shaft connected to generator. Physico-machanical quantities of the system are angular velocity of the motor W_M and angular velocity of a tachometer W_L. Moreover, it is possible to load the shaft by connection of U_L source.

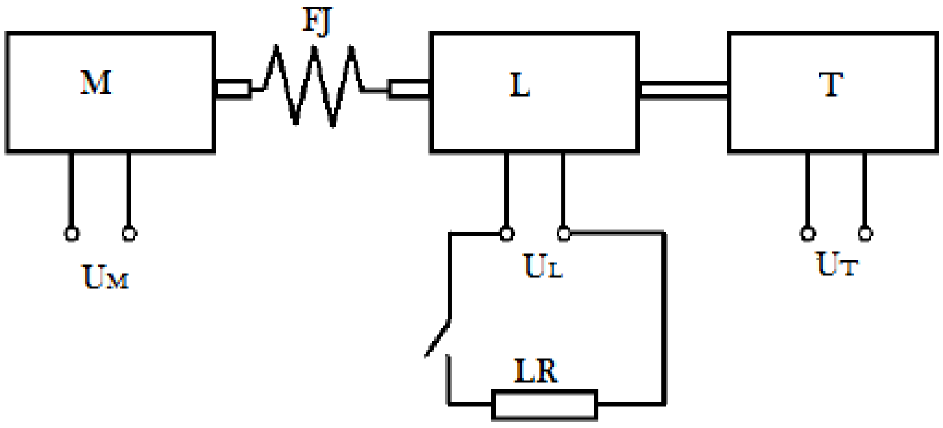


Fig. 1: Principal connection scheme of a system, adapted form[2] .

Voltage, which is controlling the speed of DC motor U_{IN} is in the form of standard unified signal 0-10 V Table 2. The system includes a source of a DC motor U_M with a range of 0-28,8V, which is controlled by a voltage of 0-10V (U_{IN}) at the input to system. The next component of a system is a 12V source, which is feeding the remaining control circuit Fig. 2 below.

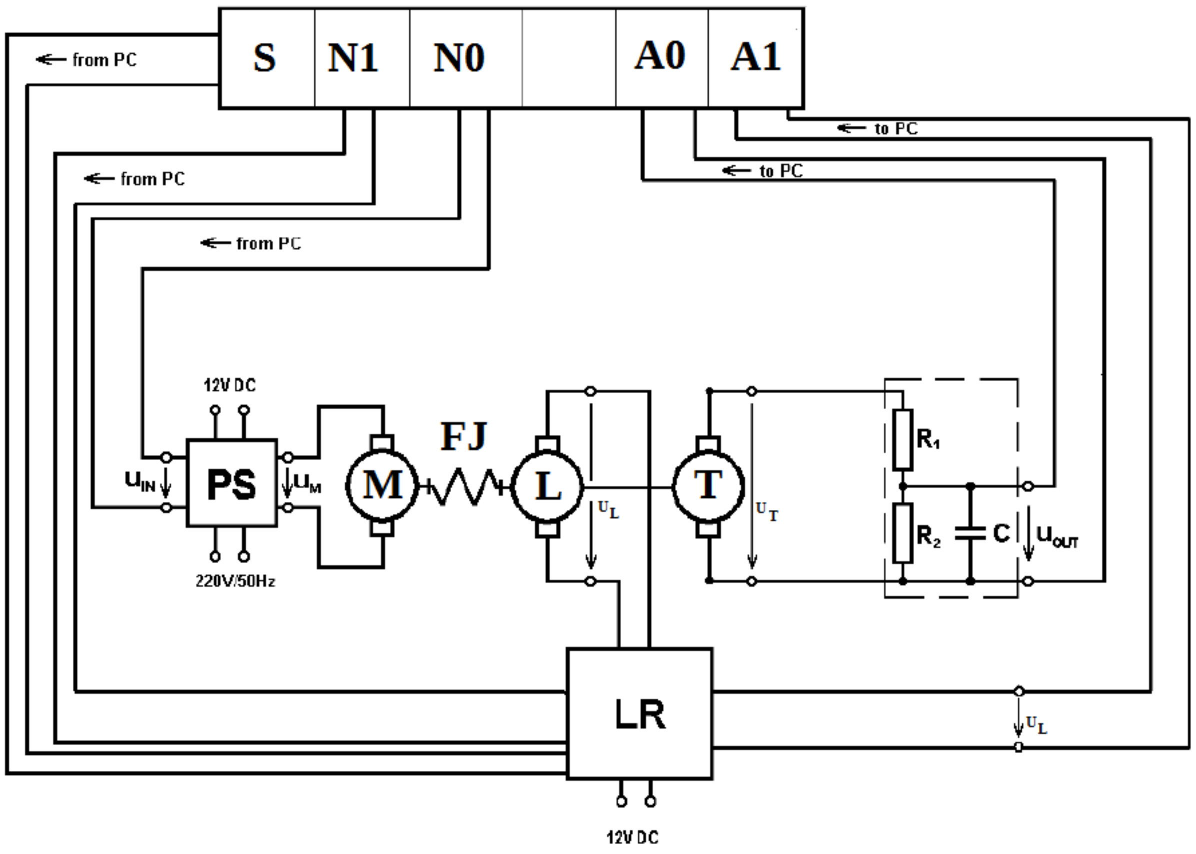


Fig. 2: Block diagram of connections, adapted and modified from [2].

M, L	DC motor with a permanent magnet P2TV553, 24V, 2000rpm, 80W
T	Tachometr K5A7-00, 20V/1000rpm
R1,R2	R1=656,3 kΩ R2=114,7 kΩ
LR	Electrical load
PS	Source of DC motor which is controlled by N0 port. The dynamics of the source could be described by following transfer function $$F_{PS}(s)=\frac{25}{(s+5)(s+5)}$$

Table 1: Block description, adapted and modified from [2]

Signal port	Signal type	Sinal limits	Signal description
N0	U_{IN}	0–10 V	Input to motor
N1	U_{LR}	0–10 V	Input to load
A0	$Uout$	0–10 V	Output voltage
A1	U_L	0–10 V	Voltage at the output of Load

Table 2: List of signals, adapted and modified from [2].

U_{IN} voltage from terminal $N0$ is controlling the speed of motor M and revolutions of M are transferred to generator by a flexible joint FJ. The speed of generator L is detected by an output voltage U_{out} which is determined by a voltage of the tachometer terminals, this voltage is reduced by resistance in the input measuring card. Therefore, real speed (revolutions per minute) of generator could be obtained through conversion of tachometer voltage conversion using following equations

$$U_{out} = U_T \frac{R_2}{R_1 + R_2}, \tag{2.1}$$

$$k_T = 1,2 = \frac{U_T}{rpm}. \tag{2.2}$$

For instance, if the output voltage U_{out} = 1 V, according to equations 2.1 and 2.2, U_L will be =6,72 V, which corresponds to a generator speed=5,6 rpm.

For more detailed research there is an option to separately increase the load of generator. See Table 3 below.

S1	S2	Function of electrical load
0	0	Generator is open circuit
1	0	Load of generator is 50% of its maximum
0	1	Load of generator is 100% of its maximum

Table 3: Logic of setting load of generator, adapted and modified from [2].

U_{IN} voltage from terminal $N0$ is coming to source of DC motor as shown in Fig. 2 in a range of 0-10V and then transferred to a motor voltage according to the following equation 2.3 as it was stated in system specification source [2]

$$U_m = U_{IN} 2.88. \tag{2.3}$$

The dependence of revolutions of motor to volt is as follows

$$W_M = \frac{2000}{24} \cdot \frac{28,8}{10} \cdot U_{IN}. \tag{2.4}$$

The dependence of voltage output to revolutions is as follows

$$W_L = \frac{1000}{20} \cdot \frac{114,7 + 656,3}{114,7} \cdot U_{out} \cdot \tag{2.5}$$

Parameter	**Description**	**Value**	**Unit**
Km	Motor constant	0,72	V·s-1
KT	Thachometer constant	20	V/rev.·min-1
KL	Generator constant	0,72	V·s-1
JM	Moment of inertia of a motor	2,8·10-2	kg·m2
JL	Moment of inertia of a tachometr	2,8·10-2	kg·m2
JT	Moment of inertia of a generator	2,8·10-5	kg·m2
BM	Friction coefficient of the motor	4,6·10-4	N·m.s
BL	Friction coeffcient of the generator	4,6·10-4	N·m.s
BT	Friction coefficient of the tachometr	2,3·10-4	N·m.s
Rm	Armature resisatnse of the motor	0,605	Ω
Rl	Armature resisatnse of the generator	0,605	Ω
Lm	Armature induction of the motor	1,6·10-3	H
Ll	Armature induction of the generator	1,6·10-3	H
R1	Resistor on the output	656,3	k·Ω
R2	Resistor on the output	114,7	k·Ω
LR	Load resistance of generator	6,6	Ω

Table 4: System parameters, adapted form [2]

3 Mathematical Model of The System

The model as described above, is analyzed for further use with two conventional PID control design methods in section 4.

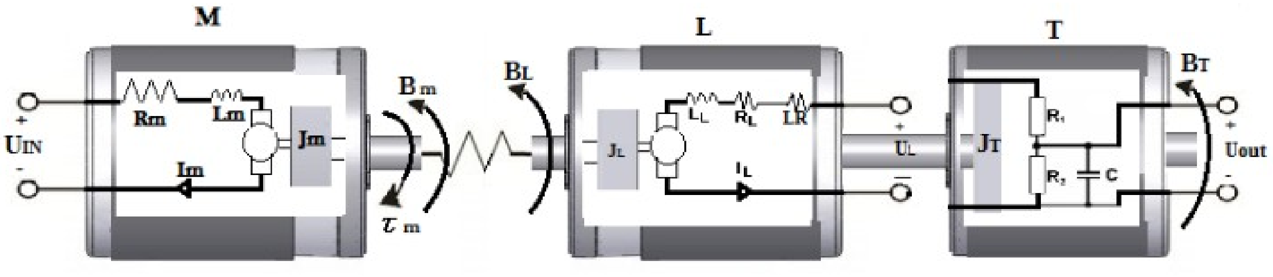


Fig. 3: Equivalent circuit of a system.

Figure above is a representation of a mechanical system with a single resonant mode. A rotary actuator with inertia J_m and internal friction B_m is connected to load with inertia J_L and friction B_L by a flexible joint and load is coupled with tachometer with inertia J_T and friction B_{T}. The motion of the actuator causes elastic deformation of the flexible connector of shafts. Flexible connector may be modeled as a linear torsional spring described by the damping factor C and the torsional stiffness factor St.

An understanding of electro-mechanical energy conversion, as exemplified by a motor, that is based upon acquaintance with several fundamental concepts from the field of mechanics and electrics: torque and electromagnetic induction respectively.

The description of motor equations, as it is explained in [3]–[5]starts with the relationship between electrical variables. These are known as the electrical equations

$$U_{IN} = L\frac{\partial I_m}{\partial t} + RI_m + K\frac{\partial \theta_m}{\partial t}\,, \tag{3.1}$$

where L is inductance, R is armature resistance, K is electromotive constant, I_m is armature current and U_{IN} is voltage supply.

Since the magnetic field in the motor is constant, the current produces a proportional torque τ_m

$$\tau_m = K_m I_m\,, \tag{3.2}$$

where K_M is the motor constant. Since there are no losses in power inside and outside of the motor, electromotive constant is equal to torsional motor constant. Therefore, in further work K_M will be used to denote both of these constants.

The dynamic of torque could be expressed as follows

$$\tau_m = J_m \frac{\partial^2 \theta_m}{\partial^2 t} + B_m \frac{\partial \theta_m}{\partial t} + D \frac{\partial \theta_m}{\partial t}. \tag{3.3}$$

Where B_m is a friction of the motor, and D is other torque proportional to the velocity. Since the motor is coupled to a load, the relation between the torques and revolutions at steady state (Practically they are not equal from the beginning but getting equal very rapid) will be described as follows

$$\tau_m - \tau_L = (J_m + J_L + J_T) \frac{\partial^2 \theta_m}{\partial t^2} + (B_m + B_L + B_T) \frac{\partial \theta_m}{\partial t} \, . \tag{3.4}$$

The motor is used as a component in a system; therefore, it is desired to describe it by the appropriate transfer function between the motor voltage and velocity. With further application of the Laplace transformation to the motor equation, we get

$$U_{IN}(s) = (Ls + R) I_m(s) + K_m \omega_m(s) \, , \tag{3.5}$$

$$I_m(s) = \frac{U_{IN}(s) - sK_m \theta_m(s)}{sL_m + R_m} \, . \tag{3.6}$$

After substituting equation 3.6 to equation 3.2 we obtain

$$\tau_m = K_m \frac{U_m(s) - sK_m \theta_m(s)}{sL_m + R_m} \, . \tag{3.7}$$

When the generator starts to rotate, it produces current I_L which is proportional to torque τ_L in the shaft:

$$\tau_L = K_L I_L \, . \tag{3.8}$$

By substituting equations of motor torque 3.7 and generator torque 3.8 to 3.4 and apply a Laplace transform, the new equation will take a form of

$$\mathbb{J} \theta(s) s^2 + \mathbb{B} \theta(s) s = K_m \frac{U_m(s) - sK_m \theta_m(s)}{sL_m + R_m} - K_L I_L \, , \tag{3.9}$$

where $\mathbb{J} = J_m + J_L + J_T$ and $\mathbb{B} = B_m + B_L + B_T$. And as a next step we can state that

$$s\theta(s) = \omega_L(s) \, . \tag{3.10}$$

As a result, angular velocity of load/generator will be as follows:

$$\omega_L(s) = \frac{K_m U_m(s) - (sL_m + R_m) K_L I_L}{(s\mathbb{J} + \mathbb{B})(sL_m + R_m) + K_m^2} \, . \tag{3.11}$$

Taking into account the fact that due to properties of flexible joint FJ there is a short delay between W_m and W_L (When I_L is steel zero). Therefore, it could be stated that the Transfer function at the beginning is as it shown in 3.12

$$G_{start}(s) = \frac{K_m}{(s\mathbb{J} + \mathbb{B})(sL_m + R_m) + K_m^2} \, . \tag{3.12}$$

Considering principle of generators, U_L is not an input but the output of back electromotive force; therefore,

$$U_L = K_L \omega_L(s) = (R_L + LR)I_L(s) + sL_L I_L \, . \tag{3.13}$$

Rearranging equation 3.13 will gives us the following formulation

$$I_L(S) = \frac{K_L \omega_L(S)}{sL_L + (R_L + LR)} \, . \tag{3.14}$$

And as soon as the generator stars to generate I_L, revolutions of motor$M$ will start to decrease. Therefore, the transfer function of reduction becomes

$$G_{reduction}(s) = \frac{-(sL_m + R_m)}{(s\mathbb{J} + \mathbb{B})(sL_m + R_m) + K_m^2} \, . \tag{3.15}$$

However the assumption of identical velocities of all the parts in the motor-load system is not accurate for high-performance systems, since the one of mechanical parts of the system is elastic FJ, and deflect under applied torque, more over stiffness coefficient of this joint is nonlinear. Consequently the instantaneous velocities of various parts are different. This condition allows the system to store a large amount of mechanical energy, which results in noticeable angular vibrations (torsional resonance). For this reason, it is necessary to describe changes of physical parameters of FJ numerically.

3.1 Stiffening of The Elastic Joint

The described system becomes higher order due to the presence of flexible joint. Therefore, short investigation of the effects of strain-stiffening on the response of torsion. It could be pretended that all rubber fibers together forming a cylinder (accept as an axiom). The most important fact, in virtue of which we have oscillatory behavior of system, is that cylinder is composed of incompressible isotropic non-linearly elastic material. As a basement for investigation neo-Hookean model was taken, where it is stated that cylinder always tends to further elongate on twisting.

According to research papers [6] and [7], the mechanical properties of elastomeric materials are described in continuum mechanics in terms of strain-energy density function W

$$W = \frac{1}{2}\mu(\alpha(I-3)) \, , \tag{3.16}$$

where $\mu > 0$ is the constant shear modulus (0.0003 rubber) for infinitesimal deformation and $\alpha = 1$ for neo-Hookean strain energy, and I is a strain invariant which is described by equation 3.17

$$I = \gamma^2 + 2\gamma^{-1} + \gamma\tau^2 R^2, \tag{3.17}$$

where γ is an axial stretch, R is cylinder radius and τ denotes the twists per unit length $(\tau R = \theta)$. The resultant moment M transferred by rubber joint is formulated as follows

$$M = 4\pi\tau \int_0^R R^3 W \partial R\,. \tag{3.18}$$

For our system, we consider that there is no axial stretch ($\gamma = 1$) and we have pure torsion.

To conclude, we have nonlinear stiffening and therefore we obtain nonlinear moment transfer as shown in Fig. 4 below.

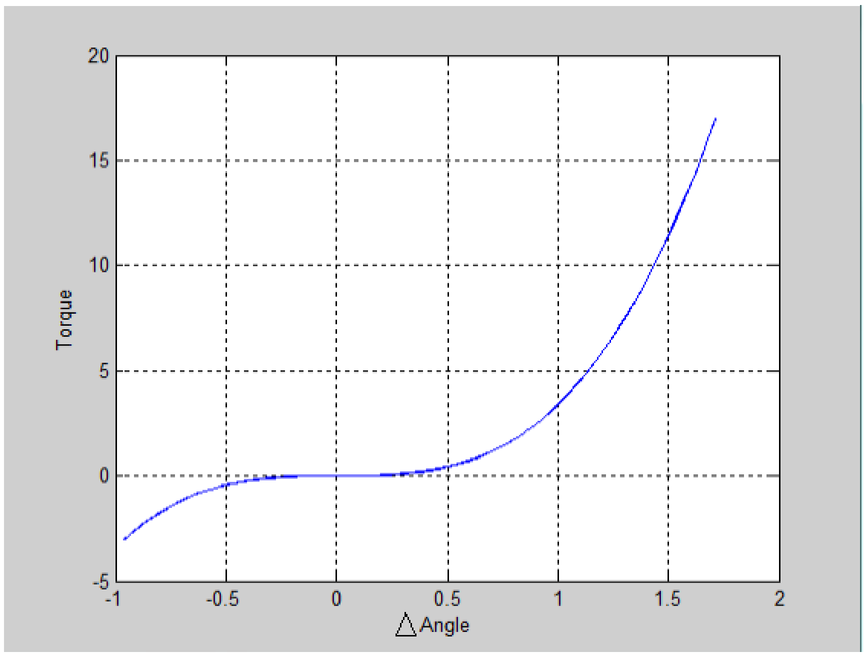


Fig. 4: Torque versus Angular difference between shafts-nonlinear dependence

With introduction of elastic parameter of FJ in terms of stiffness, the system will have difference in angle of shaft deflection. Therefore equation 3.4 becomes as follows

$$\tau_m - \tau_L = J_m \frac{\partial^2\theta_m}{\partial t^2} + B_m \frac{\partial\theta_m}{\partial t} + (J_L + J_T)\frac{\partial^2\theta_L}{\partial t^2} + (B_L + B_T)\frac{\partial\theta_L}{\partial t} \quad . \tag{3.19}$$

Due to the principle of torque following could be stated as

$$\tau_L + (J_L + J_T)\frac{\partial^2 \theta_L}{\partial t^2} + (B_L + B_T)\frac{\partial \theta_L}{\partial t} = St(\theta_m - \theta_L) + C(\frac{\partial \theta_m}{\partial t} - \frac{\partial \theta_L}{\partial t}) \quad , \qquad (3.20)$$

where term $St(\theta_m - \theta_L)$ is equivalent to M/θ from equation 3.18. Due to this nonlinear stiffening of FJ, system will have nonlinear behavior until both motors run with the same angular velocity.

3.2 State-Space Model

State space model [8] is represented in a form of

$$\begin{aligned} \frac{\partial x}{\partial t} &= Ax + Bu \\ y &= Cx + Du \end{aligned}, \tag{3.21}$$

where A, B, C and D are matrices of appropriate dimensions, x is the state vector, and u and y are the input and output vectors respectively.

State-space model below represents a system with all its inputs and outputs considering its physical and electrical parameters.

$$\begin{bmatrix} \theta_m{}' \\ \theta_m{}'' \\ \theta_L{}' \\ \theta_L{}'' \\ I_m{}' \\ I_L{}' \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ -\frac{St}{J_m} & -\frac{(B_m+C)}{J_m} & \frac{St}{J_m} & \frac{C}{J_m} & \frac{K_m}{J_m} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ \frac{St}{J_L+J_T} & \frac{C}{J_L+J_T} & -\frac{St}{J_L+J_T} & -\frac{(C+B_L+B_T)}{J_L+J_T} & 0 & -\frac{K_L}{J_L+J_T} \\ 0 & -\frac{K_m}{L_m} & 0 & 0 & -\frac{R_m}{L_m} & 0 \\ 0 & 0 & 0 & \frac{K_L}{L_L} & 0 & -\frac{(R_L+LR^*)}{L_L} \end{bmatrix} \begin{bmatrix} \theta_m \\ \theta_m{}' \\ \theta_L \\ \theta_L{}' \\ I_m \\ I_L \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \frac{2{,}88}{L_m} \\ 0 \end{bmatrix} U_{IN}$$

$$U_{out} = \begin{bmatrix} 0 & 0 & 0 & K_T \frac{R_2}{R_1+R_2} & 0 & 0 \end{bmatrix} \begin{bmatrix} \theta_m \\ \theta_m{}' \\ \theta_L \\ \theta_L{}' \\ I_m \\ I_L \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix} U_{IN} \tag{3.22}$$

As a remark following statements should be mentioned:

- LR* should be setted according to the load (e.g. LR/2 for half load)
- Either delete sixth row and sixth column in a state-space or set K_L, R_L, LR to 0 in case if no load is applied.

3.2.1 Output of State-Space Model with No Load

Sixth row and sixth column were deleted from state space for computation of system behavior with no load.

Transfer Function obtained after applying step to motor with no load in generator is as follows

$$G_{L_0} = \frac{1.474e06s + 4.17e - 13}{s^5 + 378.2s^4 + 1.194e04s^3 + 1.353e05s^2 + 2.067e06s - 3.71e - 08} \quad . \qquad (3.23)$$

The system has only a single pole equal to zero with all the other poles being negative real with or without imaginary part. Free responses are bounded but there exists bounded input signal which generate unbounded responses. Due to these facts we can state that this system is "marginally stable", as it was described in [9]

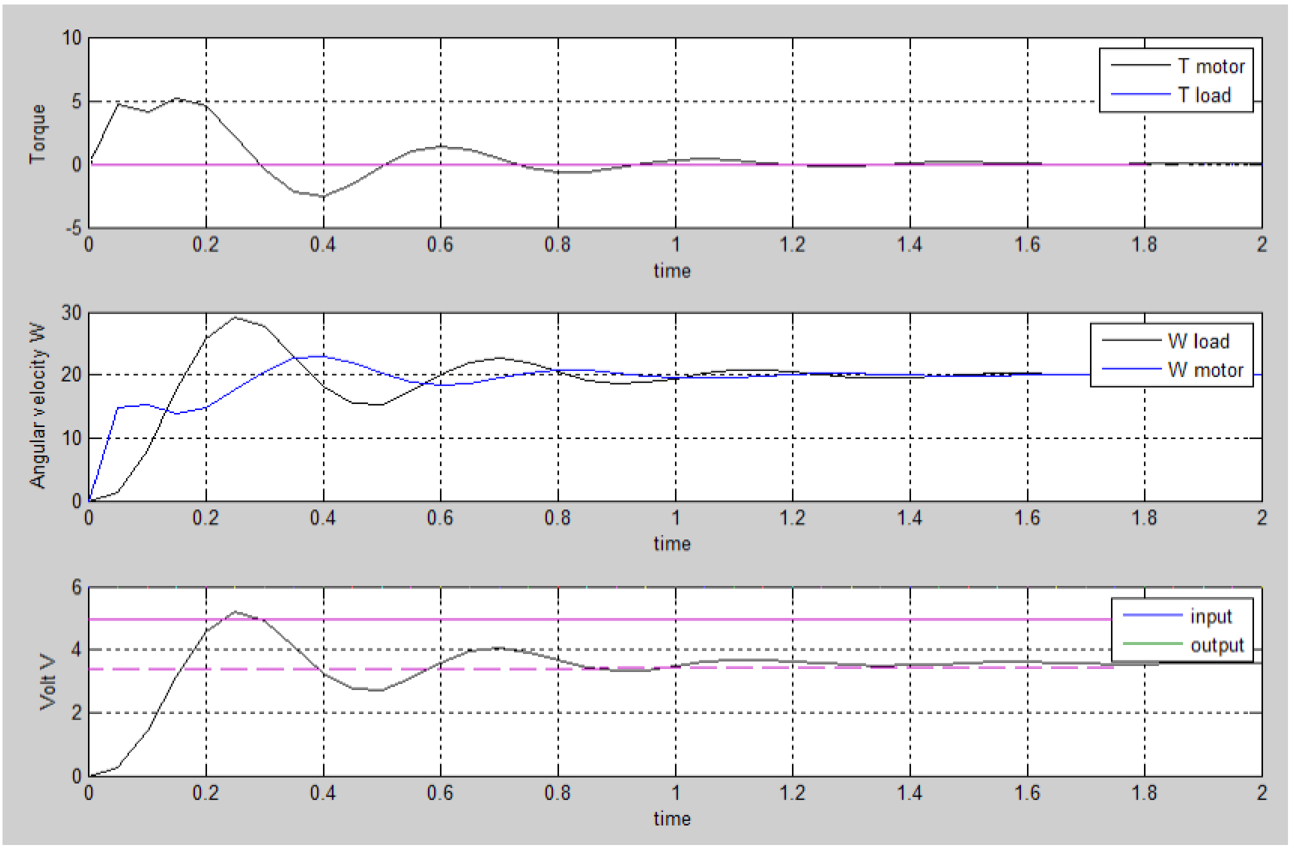


Fig. 5: State-space simulated result: generator is not loaded (T is generated torque and W is an angular velocity).

3.2.2 Output of State-Space Sodel with Half Load

Transfer function obtained after applying half load is as follows

$$G_{L_{1/2}} = \frac{1.474e06s^2 + 3.598e09s - 0.001636}{s^6 + 2819s^5 + 9.46e05s^4 + 3.366e07s^3 + 4.681e08s^2 + 5.825e09s - 0.00021}. \quad (3.24)$$

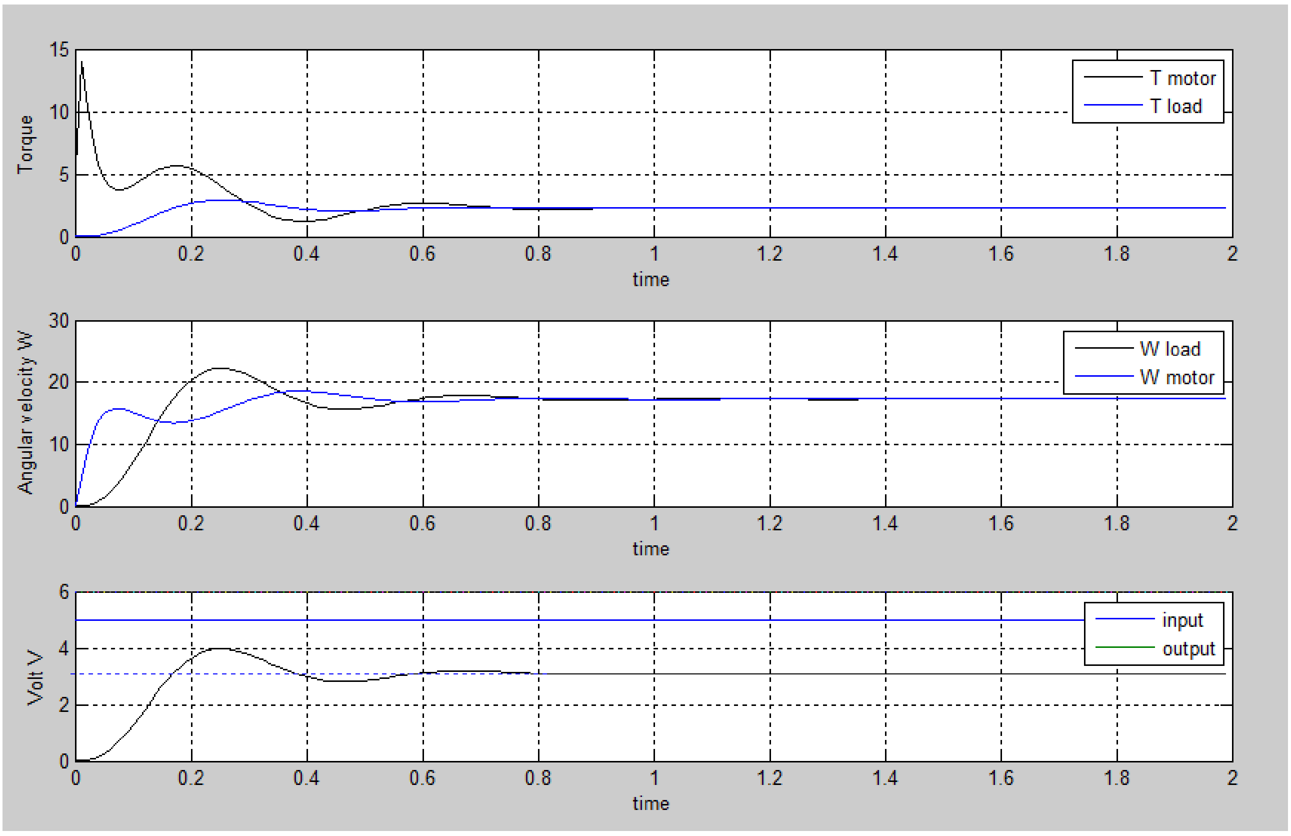


Fig. 6: State-space simulated result: generator is half loaded (T is generated torque and W is an angular velocity).

Comparing Fig. 5 and Fig. 6 above, it could be observed that loading of generator affected system behavior- dynamics became less oscillatory and decreased in steady sate gain of U_{out}

Due to the fact that the main goal of this thesis is to control oscillatory system. Therefore, in further work I will be dealing with unloaded generator case only.

Simulink Model

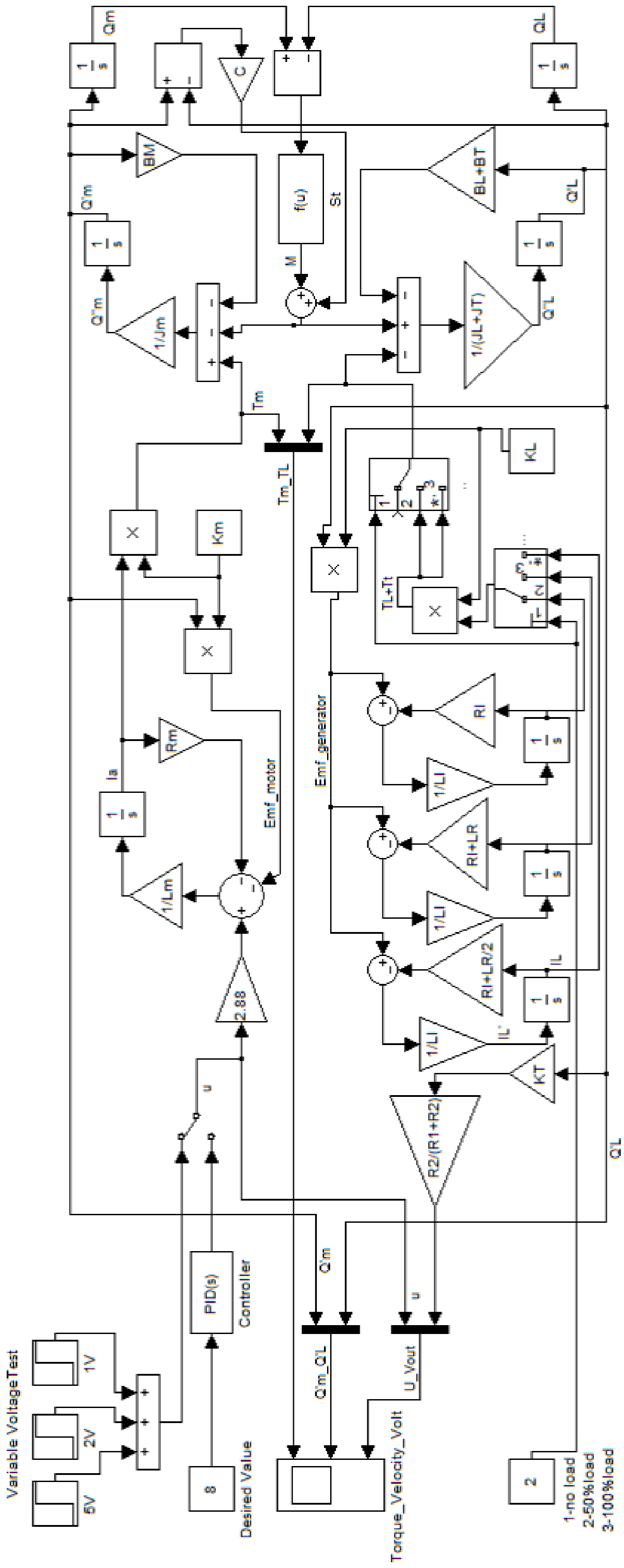


Fig. 7: Simulink model of the system

4 Conventional PID Control

In this section, fundamental conventional methods of control are reviewed while the real results on the coupled motor plant are shown in experimental section Error: Reference source not found.

4.1 Pole Placement

In this subsection, a PID controller is analyzed with theoretical method as in [10] is based on mathematical model derived in section 3.

General transfer function of PID is

$$G_c = K_p[1 + \frac{1}{T_i s} + T_d s] \ , \tag{4.1}$$

where K_p is the proportional gain, T_i is the integral time constant and T_d is the derivative time constant. In time domain , the controller output is as follows

$$U_{IN} = r_p e + r_i \int_0^t e \partial t + r_d e', \tag{4.2}$$

where $K_p = r_p, \dfrac{K_p}{T_i s} = r_i, K_p T_d = r_d$. In order to get rid of integral part take first derivative and it could be obtained as

$$U'_{IN} = r_p e' + r_i e + r_d e'' \ , \tag{4.3}$$

$$e = U_{set} - U_{out} \ ,$$

where U_{set} is a desired value or a desired steady state gain. Due to the fact that transfer function of unloaded system is as in equation 3.9, we obtain the following view of equation 4.3

$$\omega(s)s\frac{[\mathbb{J}\omega(s)s + \mathbb{B}\omega(s)][sL_m + R_m]}{K_m} = U'_{IN}, \tag{4.4}$$

$$= r_p(U_{set}(s)s - U_{out}(s)s) + r_i(U_{set}(s) - U_{out}(s)) + r_d(U_{set}(s)s^2 - U_{out}(s)s^2) \quad .$$

Desired polynomial characteristics for balance should be as follows

$$(s+p)^3 = p^3 + 3p^2 s + 3ps^2 + s^3. \tag{4.5}$$

Therefore, after manipulations with equation 4.4 and 4.5 the following balanced numerical approximation controller gains could be obtained

$$\begin{array}{r} r_p = \frac{K_m}{\mathbb{J}L_m} 3 \cdot p - (\frac{\mathbb{J}R_m + \mathbb{B}L_m)}{K_m} \\ r_d = \frac{K_m}{\mathbb{J}L_m} 3 \cdot p^2 - (\frac{\mathbb{B}R_m + K_m^2}{K_m}) \ , \\ r_i = \frac{K_m}{\mathbb{J}L_m} 3 \cdot p^3 \end{array} \tag{4.6}$$

where p is dimensionless parameter which has to be set according to desired system behavior.

4.2 Ziegler Nichols Closed Loop Tuning Method

In this subsection, a PID controller is analyzed with practical-tuning method as in [11] is based on mathematical model derived in section 3.

Another approach is to use a technique which was developed in the 1950's but which has stood the test of time and is still used today, which is known as the Ziegler Nichols closed loop tuning method.

The procedure is as follows:

1. Choose just proportional control. Integral and derivative part set to zero.

2. Increment the value of the proportional gain until sustained oscillations in the signal of a control system are reached. This will give us the critical value of controller gain K_c**.** It is important to pay attention on finding K_c without driving control signal to any saturation limit during oscillations. Otherwise we will find sustained oscillation for any large value of proportional gain and resulting K_c will be useless.

3. Measure the period of oscillation to obtain the critical time constant, T_c.

Once the values for K_c and T_c are obtained, the PID parameters can be calculated, according to the design specifications, from the following Table 5.

Control	**Kp**	**Ti**	**Td**
P only	**0.5 Kc**		
PI	**0.45 Kc**	**0.833 Tc**	
PID tight control	**0.6 Kc**	**0.5 Tc**	**0.125 Tc**
PID some overshoot	**0.33 Kc**	**0.5 Tc**	**0.33 Tc**
PID no overshoot	**0.2 Kc**	**0.3 Tc**	**0.5 Tc**

Table 5: Ziegler Nichols tuning coefficient table

4.3 Experimental Results With Current Solution

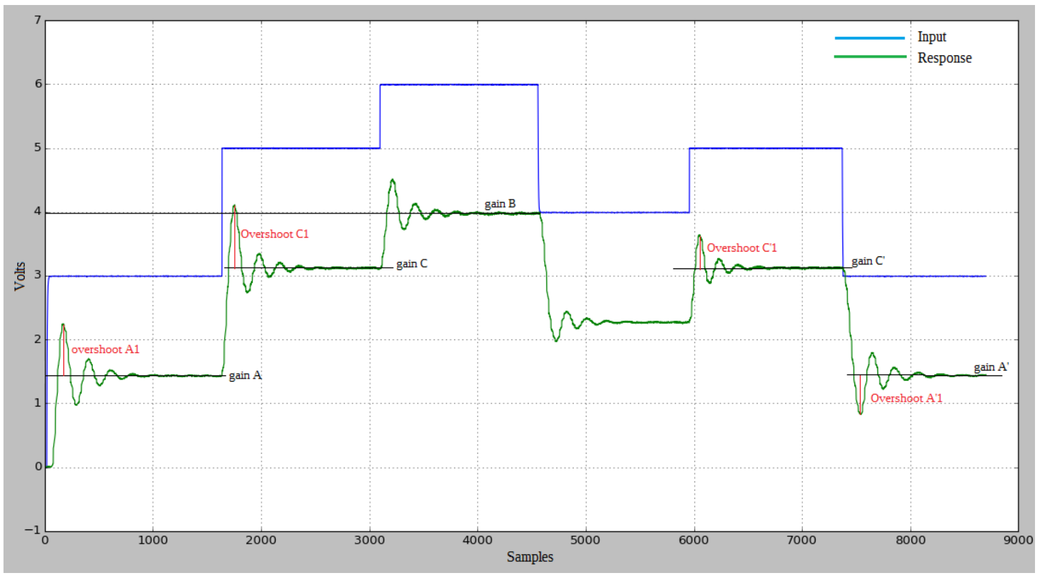


Fig. 8: Plant response to variable input step signals

Considering Fig. 8 above, it could be stated that is going to be tested where the operating point varies and the process dynamic properties are dependent on the initial conditions and operating point. For instance in the Fig. 8, it could be observed that the dynamics of A and A' as well as C and C' are highly dependent on initial conditions- either if step changed form higher point to lower or wise-verse. The second factor of non-linearity which could be observed by following gains A, B and C, is that system does not satisfy the superposition and homogeneity properties perfectly as it could be observed from static characteristics of the system in Fig. 9.

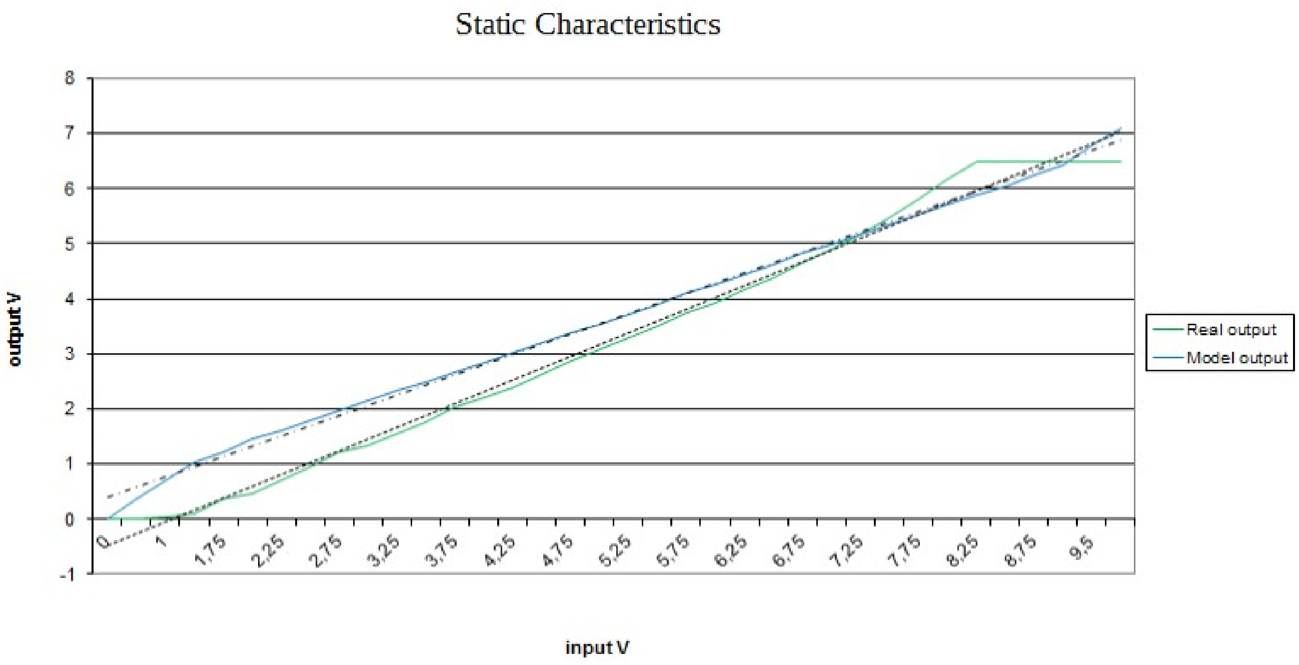


Fig. 9: Static characteristics of the plant.

Therefore, it could be considered to use gain scheduling for coarse tuning where PID parameters are adjusted as function of the operating point so that the control system dies not overshoot or oscillate over a set-point. But still the fact that the system is only having tendency to non-linearity which is mainly caused by the FJ properties, because system do not possess any non-linearity besides the joint as it was previously described in section 3.1, lead to idea that the process could be assumed as linear and control of the process could be realize with conventional PID.

Conventional Controller Implementation

Here follows result in the Fig. 10 of tuning PI controller with respect the Ziegler -Nichols method described in section 4. Where the critical gain was found as 0.8 and period of sustained oscillation as 2 seconds.

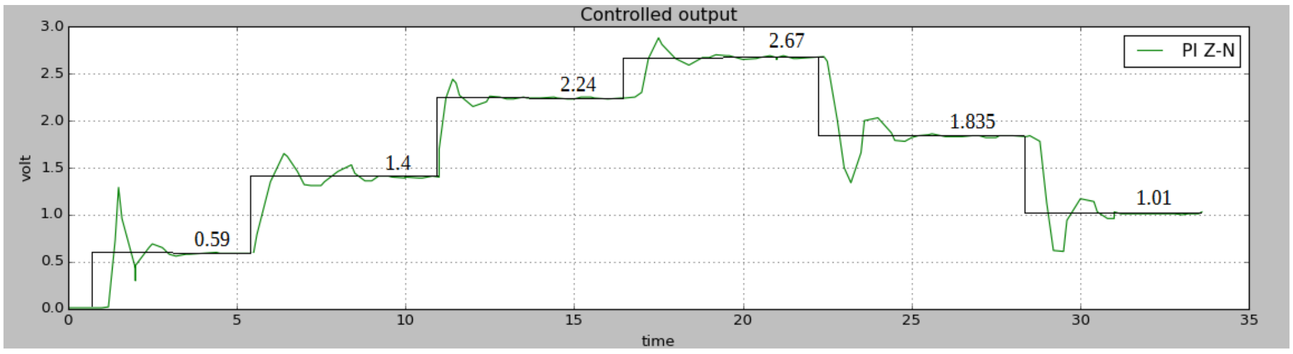


Fig. 10: Results of implementation PI to variable set-points controller with existing solution.

For the second approach of PID tuning I was using numerical approximation method described in section 22. As a model parameters I was using real values described in section 2. In the Fig. 11 we can see that in practice numerical approximation did not show the best result, what could be reasoned by shortage of some influential parameters in the mathematical model.

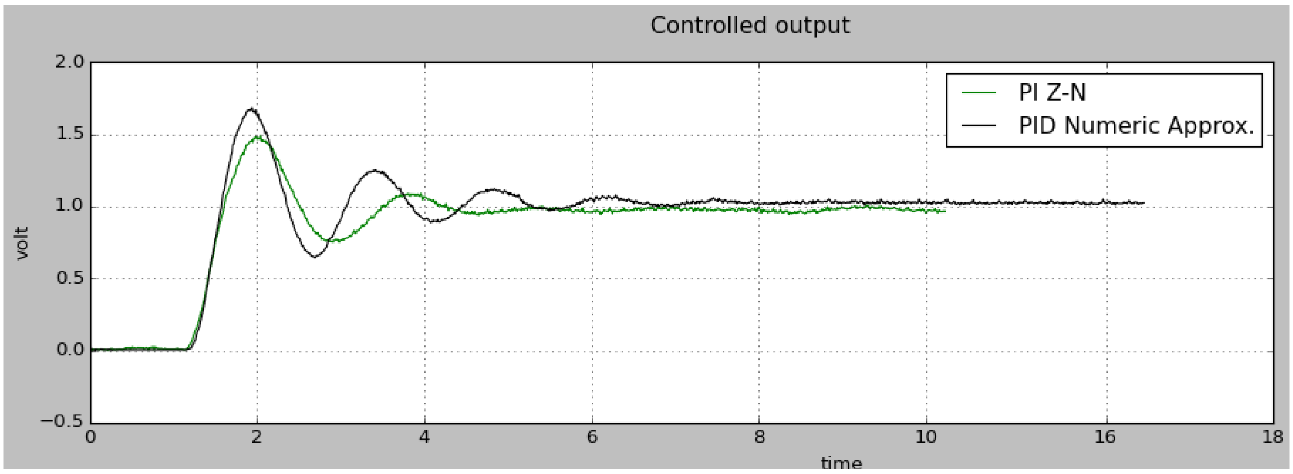


Fig. 11: Performance of three different types of controller over operating point.

This results are telling us that indeed it is convenient to use PI controller for this process. I strongly believe that results of controlled system with PI could be improved by increasing sampling time and trial-and-error procedure for finding more sufficient results. Due to the reason that it is not possible to increase sampling time with existing control solution which is limited by sampling frequency of measuring card CTRLV3. Therefore, objective of this thesis is to propose a new solution with the use of Neural Networks to control coupled motor system. For that reason, I will proceed with neural approach without going into details of possibility of more advanced tuning of PID controller.

5 Applied Identification and Control Algorithms

For solution of identification and control, I have adopted discrete-time adaptive approaches of Higher Order Neural Units (HONU) for identification as published [12]–[15] and for control as in [16].

In this section, I will use Higher Order Neural Units as a recurrent models for dynamic system identification and furthermore, as a controller. Implementing offline adaptively trained dynamic neural models for system identification, followed by static application of an extended neural unit state feedback controller, as a model with constant parameters, for control.

Algorithms that were implemented are polynomial function based HONUs. System will be identified and controlled via use of famous Gradient Descent (further just GD) and Levenberg-Marquardt (further just LM) algorithms. The applied learning rule for dynamic neural units is based on incremental and batch training techniques. The incremental adaptation is based on Real Time Recurrent Learning (further just RTRL) technique. The batch adaptation is based on modification of Bach Propagation Through Time (further BPTT) technique implemented as a combination of RTRL wuth the LM algorithms. Both learning techniques RTRL and BPTT can be used both independently and combined to adaptively identify or control a system.

5.1 Sample-by-Sample Learning Plant Identification

The essence of this algorithm is to learn the model of a plant and to tune a controller such as the neuron type controller. The GD is fundamental learning rule behind neural units such as Linear Neural Unit (further LNU) [12]–[15], [17] and Quadratic Neural Unit (further QNU) [12]–[15] and [17] . The LNU could be described by the following polynomial

$$y_n = w_0x_0 + w_1x_1 + w_2x_2 + w_3x_3 + ... + w_nx_n = \mathbb{W} \cdot \mathbb{X}, \tag{5.1}$$

where $\mathbb{W}$ is updatable vector of neural weights and $\mathbb{X}$ is a vector of inputs to a neuron. Similarly, the quadratic neural unit QNU can be expressed with the following polynomial

$$y_n = \sum_{i=0}^{n}\sum_{j=0}^{n} x_ix_jw_{i,j} = w_{0,0}x_0x_0 + w_{0,1}x_0x_1 + ...w_{n,n}x_nx_n = row\mathbb{X} \cdot col\mathbb{W} \quad , \tag{5.2}$$

where $row\mathbb{X}$ and $col\mathbb{W}$ are long-vectors representation of the input vector and the weight matrix of the QNU in general. Since we are dealing with dynamic neuron (with a feedback form output), we choose vector $\mathbb{X}$ so that to feed previous samples of outputs to inputs in from of

$$\mathbb{X} = \begin{bmatrix}1 & y(k) & y(k-1) & y(k-2) & u(k) & u(k-1) & u(k-2)\end{bmatrix}^T \quad , \tag{5.3}$$

where y is plant output, u is a plants input. Amount of $y(k)$s and $u(k)$s corresponds to components of input vector in terms of ny and nu respectively. These components were calculated using the model equation at different time instances of k. For instance, in such case size of input vector equals to 7, therefore updatable vector $\mathbb{W}$ must be the same size.

The main goal behind this algorithm is adaptation of neural weights. And this is a key factor for the learning process of the model. Adaptation of neural weight is achieved by modification of the fundamental GD formula. Modification of GD for the LNU is as follows

$$w_{i+1} = w_i - \frac{1}{2}\mu \cdot \frac{\partial e^2(k)}{\partial w_i}, \tag{5.4}$$

where μ is the learning rate of the weight adaptation and $e(k)$ represents current error between real and calculated output.

$$e(k) = y_r(k) - y_n(k) \tag{5.5}$$

Further modification of equation 5.4 will have the following form

$$\begin{gathered} w_{i+1} = w_i - \frac{1}{2}\mu \cdot 2e(k) \cdot \frac{\partial e(k)}{\partial w_i}, \\ w_{i+1} = w_i - \mu \cdot e(k)\left(\frac{\partial y_{r(k)}}{\partial w_i} - \frac{\partial y_{n(k)}}{\partial w_i}\right), \\ w_{i+1} = w_i + \mu \cdot e(k)\frac{\partial y_{n(k)}}{\partial w_i}, \end{gathered} \tag{5.6}$$

where the final term $\frac{\partial y_{n(k)}}{\partial w_i}$ corresponds to the partial derivatives of the neural inputs respective to each neural weight. Since y_r is independent of neural weights, its partial derivative is zero.

Modification of GD for QNU will work with the same procedure and will take a following from

$$col\mathbb{W}(k+1) = col\mathbb{W}(k) + \mu \cdot e(k) \cdot \frac{\partial y_{n(k+1)}}{\partial col\mathbb{W}(k)} \cdot \tag{5.7}$$

The principal schematics for the system identification should be considered as shown in the follows following Fig. 12

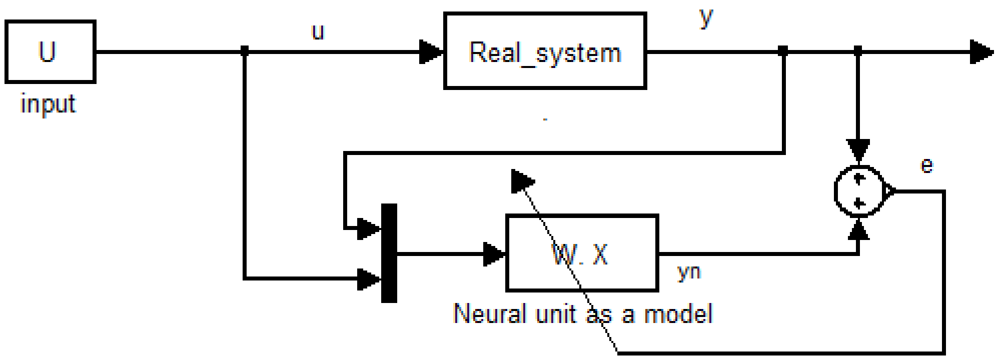


Fig. 12: Principal schematics for plant identification with a neural unit

5.2 Batch Learning Plant Identification

Levenberg-Marquardt algorithm as a back-propagation through time. The main aspect to distinguish from RTRL (GD) is that BPTT adaptation is not achieved by decremental step (sample by sample), but over a series of runs or Batches of the neural algorithm. The equation of neural weight update according to LM has a form of

$$w_{i+1} = w_i + (\mathbb{J} \cdot \mathbb{J}^T + \frac{1}{\mu}\mathbb{I})^{-1} \cdot \mathbb{J} \cdot \mathbf{e}\,, \tag{5.8}$$

where the term $\mathbb{I}$, is an identity matrix, $\mathbb{J}$ is the Jacobean matrix [18] of derivatives of the models polynomial equation. Thus a modification of equation 5.8 for a batch training of neural weight update for LNU and QNU will take form as shown in equation 5.9 and 5.10 respectively.

$$w_{i+1} = w_i + (\mathbb{X} \cdot \mathbb{X}^T + \frac{1}{\mu}\mathbb{I})^{-1} \cdot \mathbb{X} \cdot \mathbf{e}\,, \tag{5.9}$$

$$w_{i+1} = w_i + (col\mathbb{X} \cdot col\mathbb{X}^T + \frac{1}{\mu}\mathbb{J})^{-1} \cdot col\mathbb{X} \cdot \mathbf{e}\,. \tag{5.10}$$

5.3 Tuning Neuro-controller

Algorithm for Neuro-Controller possess the same concept as for plant identification. The difference is that Neuro-Controller is used to manipulate newly feed input into the neural model for control as shown in the Fig. 13 above.

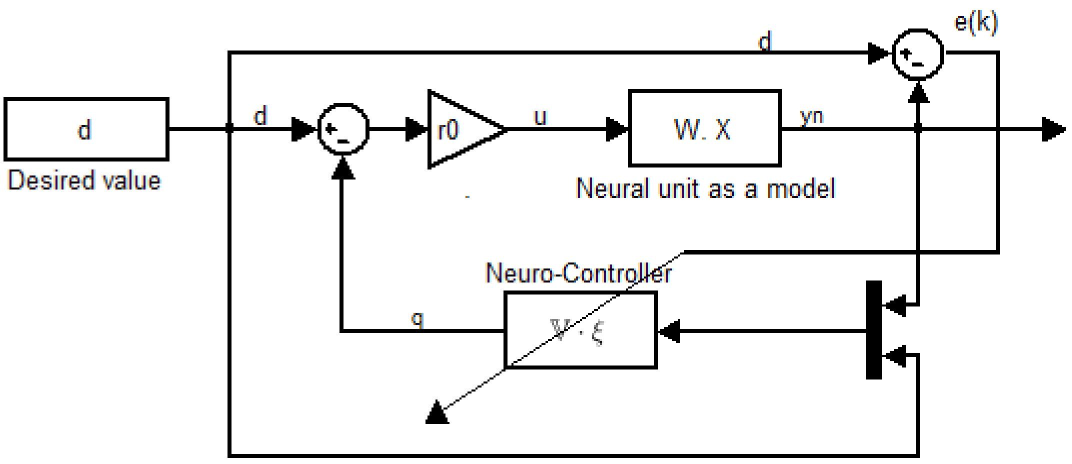


Fig. 13: Schematics for tuning neural control offline

Where q by its nature is similar to structure of y_n, therefore, to make a distinguishing difference, vector of neural weight for the controller will be denoted as $\mathbb{V}$ and input vector will be denoted as ξ. Equation of controller output will be as follows

$$q = v_0\xi_0 + v_1\xi_1 + v_2\xi_2 + v_3\xi_3 + ... + v_n\xi_n = \mathbb{V} \cdot \xi \, . \tag{5.11}$$

In this case, the neural weight update for controller will take the following form

$$v_{i+1} = v_i + \mu_c e_{reg}(k) \cdot \frac{\partial y_n(k)}{\partial v_i} \, , \tag{5.12}$$

where v_i are adaptable neural weights of the controller, $e_{reg}(k)$ is an error between the reference model/desired value and the output value of the plant. Last and the most important component of equation is $\frac{\partial y_n(k)}{\partial v_i}$, because it binds dependence of neural model of a plant to an output of a controller if the following manner

$$\frac{\partial y_n(k)}{\partial v_i} = \frac{\partial(\mathbb{W} \cdot \mathbb{X}_{(k)})}{\partial v_i} = \mathbb{W} \cdot \frac{\partial(\mathbb{X}_{(k)})}{\partial v_i} \tag{5.13}$$

$$\frac{\partial \mathbb{X}_{(k)}}{\partial v_i} = \begin{bmatrix} 1 \\ y_{(k)} \\ y_{(k-1)} \\ ... \\ (d_{(k)} - q_{(k)})r_0 \\ (d_{(k-1)} - q_{(k-1)})r_0 \\ ... \end{bmatrix} = -r_0 \frac{\partial}{\partial v_i} \cdot \begin{bmatrix} 0 \\ 0 \\ 0 \\ ... \\ q_{(k)} \\ q_{(k-1)} \\ ... \end{bmatrix} == -r_0 \cdot \begin{bmatrix} 0 \\ 0 \\ 0 \\ ... \\ \xi_{(k)} \\ \xi_{(k-1)} \\ ... \end{bmatrix} \, . \tag{5.14}$$

We choose vector ξ so that to feed previous samples of plants outputs to inputs in from of

$$\xi = \begin{bmatrix} y_n(k) & y_n(k-1) & ... & d(k) & d(k-1) & ... \end{bmatrix}^T \tag{5.15}$$

Amount of previous outputs $y_n(k)$s and previous inputs $d(k)$s corresponds to components of input vector in terms of ny_n and nd respectively.

5.4 Procedure of Algorithm Implementation

For system identification, real plant data should be gathered. Software application will identify the system offline in terms of neuron long vector of weights $\mathbb{W}$. This vector will be saved as a text file, and further will be used for offline tuning and identification of controller parameters in terms of neuron long vector of controller weights $\mathbb{V}$.

Sampling rate is trivial for the identification and control because it is directly related to size of neuron input vector. For instance, if sampling rate is 100 samples per second, for system identification and control we will require around 60 previous inputs and outputs to a neuron vector $\mathbb{X}$, what is not efficient for calculation. Therefore, re-sampling is required so that the size of neuron vector will be in range of 10 to 20 (inputs and outputs together).

Another important point, for controller identification, is that sampling rate for controller must be the same as in system identification. With a remark for the fact that learning rate μ_c and size of a neuron vector ξ for a controller could be different.
The last point to mention is that learning rate μ essentially corresponds to the speed of learning. If the value is high, system will be learned in a fast rate. If the value is small, system will be learned in slower rate. For learning efficiently, it is better to select smaller μ and due to the problems associated with instability during learning of neural unit, a good step to resolve this issue is to normalize learning rate as shown in the following form for LNU and QNU in equations 5.16 and 5.17 respectively

$$\mu_{LNU} = \frac{\mu}{\mathbb{X}(k)\mathbb{X}(k)^T + 1} \tag{5.16}$$

$$\mu_{QNU}\frac{\mu}{col\mathbb{X}(k)col\mathbb{X}(k)^T + 1} \tag{5.17}$$

For the purpose of better identification of controller parameters, preferably to have a reference model, which tends to represent ideal response of a system to a desired set value. Key factor for Reference Model is that it has to be feasible by a real plant/system. Therefore, schematics for identification of neuro-controller should be considered as shown in the following Fig. 14 below.

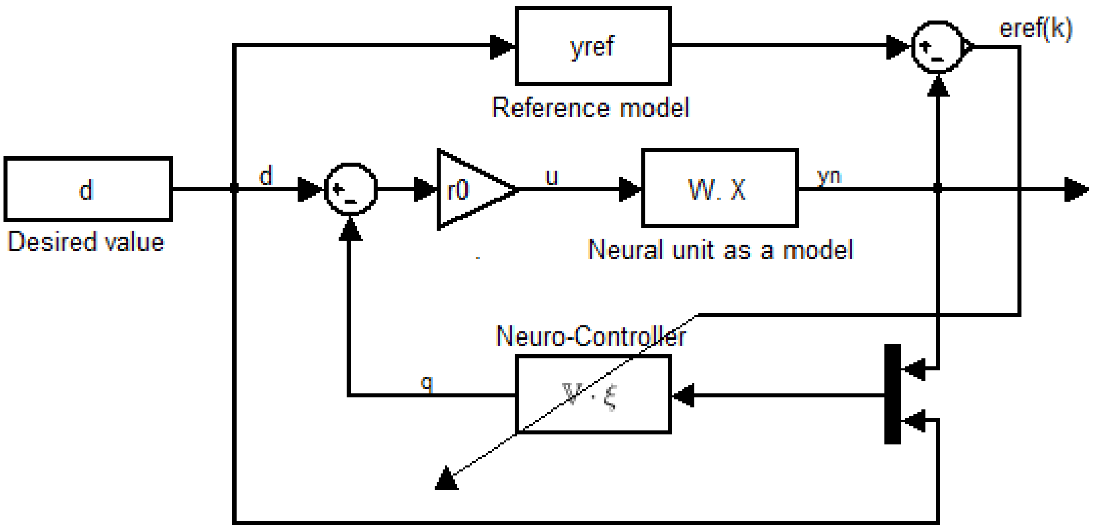


Fig. 14: Shematics of tuning Neuro-controller with respect to referensce model

Some engineering processes require adaptation of not only neural weights of a controller but also the gain r_0. As an update rule for the gain, e_{ref} is used, as it is used for the Neuro-Controller. The update of r_0 is realized with the same principal as weight update via GD method as follows

$$r_{0(i+1)} = r_{0(i)} - \frac{1}{2}\mu \cdot \frac{\partial e_{ref}^2(k)}{\partial r_{0(i)}}, \tag{5.18}$$

since $e_{ref} = y_{ref} - y_n$, thus modification of equation 5.18 we lead to the following

$$r_{0(i+1)} = r_{0(i)} - \frac{1}{2}\mu \cdot e_{ref} \cdot \frac{\partial(y_{ref} - y_n)}{\partial r_{0(i)}}. \tag{5.19}$$

Since y_{ref} is independent of adaptive gain, partial derivative of y_{ref} with respect to r_0 will be zero. And taking into account that $y_n = \mathbb{W} \cdot \mathbb{X}$. Further modification of equation 5.19 will lead us to the following formulation

$$r_{0(i+1)} = r_{0(i)} + \frac{1}{2}\mu \cdot r_0 \cdot e_{ref} \cdot \mathbb{W} \cdot \frac{\partial \mathbb{X}}{\partial r_{0(i)}}. \tag{5.20}$$

6 Proposition of a HW and SW Platform

6.1 Software

Python 3.5 [19]was selected as a low-cost software for realization of control of coupled motor system. This selection was made due to the fact that this software is free including its additional libraries. Python programming language is considered suitable for beginners with its simple syntax. This software combines advantages of programming in C++ and Java, therefore it is suitable for creation of complex applications.

In addition to standard package of Python software it is necessary to download extra library called **wxPython** [20] for creation of graphical user interface for simplicity of interaction with a code. And the last module to download is **LabJackPython**[21] for communication with LabJack U3-HV.

6.2 Hardware

LabJack U3-HV [22] which is shown on Fig. 15 was selected as a low-cost USB multifunction data acquisition and control device because it has flexible inputs which can be configured for any digital and analogue inputs and outputs. Command and response (software timed) analog input reads typically take 0.6-4.0 ms depending on the number of channels and communication configuration.

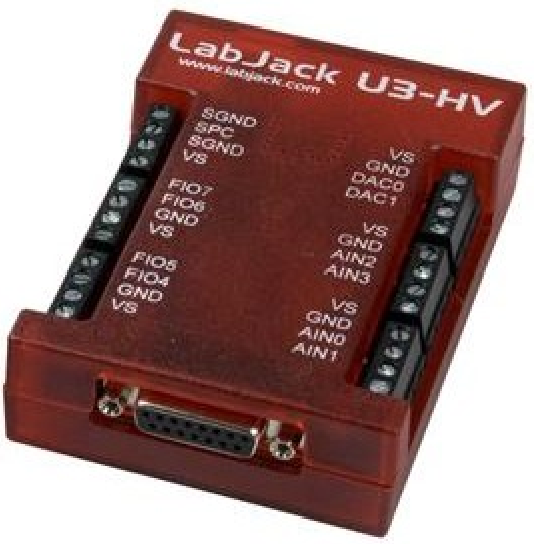


Fig. 15: LabJack U3-HV low-cost HW, adopted from [22]

6.2.1 Hardware Parameters

- First 4 Flexible I/O are Changed to Dedicated HV Analog Inputs.
- 4 HV Inputs have ±10 Volt or -10/+20 Volt Range.
- 12 LV Inputs (Flexible I/O) Still Available, for 16 Total Analog Inputs.
- Up to 2 Timers (Pulse Timing, PWM Output...)
- Up to 2 Counters (32-Bits Each)
- 4 Additional Digital I/O
- 2 Analog Outputs (10-Bit, 0-5 volts)
- Supports SPI, I2C, and Asynchronous Serial Protocols
- Supports Software or Hardware Timed Acquisition
- Maximum Input Stream Rate of 2.5-50 kHz (Depending on Resolution)
- Capable of Command/Response Times Less Than 1 Millisecond
- Built-In Screw Terminals for Some Signals
- OEM Version Available
- USB 2.0/1.1 Full Speed Interface
- Powered by USB Cable
- Drivers Available for Windows, Linux, Mac and Pocket PC
- Examples Available for C/C++, VB, LabVIEW, Python, and More
- Enclosure Size Approximately 3" x 4.5" x 1.2" (75mm x 115mm x 30mm)

6.3 Description of Communication with LJ

To open the devise, first of all it is necessary to install u3 class, which is available on LabJack's website [23].
To open devise on python

```
>>> import u3
>>> d = u3.U3()
```

The constructor for the U3 class will try to automatically open the first found U3. First, before we perform any other operation we will get LabJack device's calibration data. The calibration data will be used by functions that convert binary data to voltage and vice-versa.

```
>>> d.getCalibrationData()
```

As the documentation sates, the 8-bit DAC0 feedback command takes a value between 0 and 255 (inclusive) and the output of the DAC is between 0 and 5 V. So it is necessary to converts desired voltage between this two extremes. For instance, to set DAC0 to 5V

```
>>> d.getFeedback(u3.DAC8(0,255))  #0 is number of port (write 1 for DAC1)
```

Reading an analog input could be done using

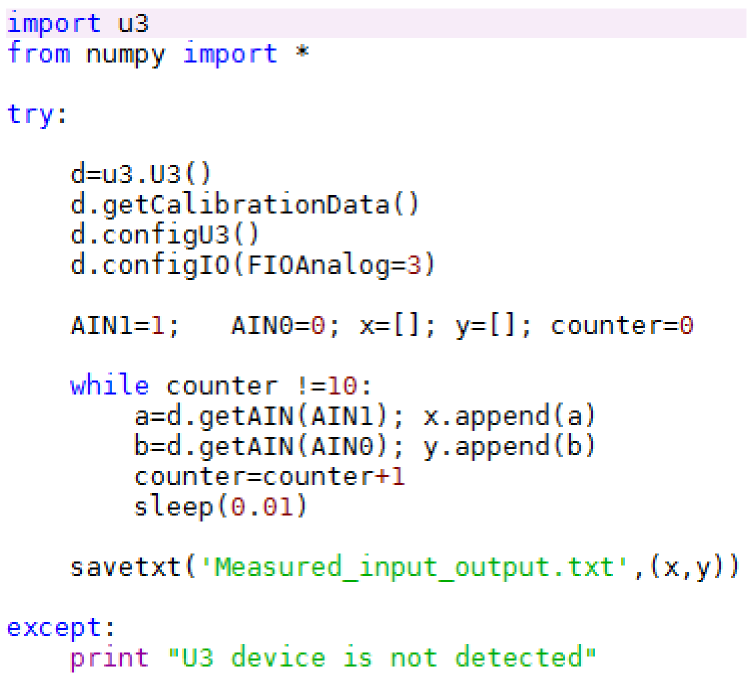
```
>>> input = d.getAIN(0)
```

In order to use AIN ports, we have to activate them first. For instance, to set the first two FIO's (0-1, which are indicated on devise as AIN0 and AIN1 respectively)to analog (3 = 00000011 binary) and the rest to digital

```
>>> d.configIO(FIOAnalog = 3)  #if to set first four (0-3), write 15 (=00001111)
```

6.4 Measuring of process data

The principal script of measuring input, output data and further saving acquired data to file, looks as follows



```
import u3
from numpy import *

try:

    d=u3.U3()
    d.getCalibrationData()
    d.configU3()
    d.configIO(FIOAnalog=3)

    AIN1=1;   AIN0=0; x=[]; y=[]; counter=0

    while counter !=10:
        a=d.getAIN(AIN1); x.append(a)
        b=d.getAIN(AIN0); y.append(b)
        counter=counter+1
        sleep(0.01)

    savetxt('Measured_input_output.txt',(x,y))

except:
    print "U3 device is not detected"
```

Code. 1 Principal of Data acquisition with LJ

Where x and y are dimensionless arrays for data storing, AIN0 & AIN1 are assigned to be a port number. Execution will lead to 10 samples of input and output with interval of 0.01 seconds.

6.4.1 Data Measuring Methods with LabJack

Stream mode

For fastest data sampling its useful to use 'stream' mode, which measures date in packages of 25 data samples [24]. By this method it is possible to obtain up to 20000 samples per second without missing data or errors in best case. For all experiments I was using ordinary USB cable and Core i5 2,4GHz M520 PC on Windows 7. There exists a possibility to increase the sampling up to 50000 samples per second, but in this case data gathered will have a lot of errors and missing values.

Constant sampling

Obviously, 'Stream' mode is not always appropriate for laboratory measurements, especially if we are interested in constant time sampling and error-less measurements. For these purposes it is better to use low level command [23] 'd.getAIN(0)' with introduction of 'sleep() ' command in order to compensate measurement time delays and other changeable parameters.

Sampling statistics

During experiments, if not on stream mode, sampling time most often is 0.003 second, but it can be even 0.1 due to uncertainties. In order to prevent multi-sampling at a time (3-4 samples in one time instance) it is necessary to use 'sleep()' command at least for 0.001 seconds. In this case, frequency of slightly more than 100 samples per second could be achieved, but still sampling time is not constant in this case. If to use 'sleep()' command for 0.01 second, sampling will take 0.009-0.01 second what will lead to almost 100 samples per second with 98% similar sampling time of 0.01 seconds.

Comparison of LabJackUD application to Python

Stream modes were tested in Python and in LJStreamUD [25] application. No significant difference was observed in terms of sampling and final amount of obtained data in a sampling period. The main advantage of LJStreamUD application is that obtained data are visualized with out any time loss. In contrast to Python, processing of data using LabJack application takes significantly longer time. Therefore, it could be stated that both programs are efficient depending on requirements and field of implementation.

6.5 Data visualization methods

In this section I will describe the basic principal of data visualization, more detailed codes are included in Appendix.

Offline visualization

Simplest method for data visualization is plotting saved data either at the end of execution of main code or separately saved to a data-sheet.

```
#for plotting from script
time.append(t)
input.append(u)
output.append(y)
plot_function(time, input, output)

#for plotting from file
data=loadtxt('data.txt')
time=data[:,1]
input=data[:,2]
output=data[:,3]
plot_function(time, input, output)

def plot_function(time,input,output):
        figure();plot(time,input);
         hold();grid();plot(time,output);
        show()
```

Code. 2: Offline visualization of measured data-set

Method of visualization described in Code. 2 is best in terms of time saving for code execution (should be used if the aim is to get as mach data as possible with constant time sampling).

Real time visualization

During laboratory experiments most often it is desirable to have visual understanding of ongoing process. Therefore, is it necessary to have a real-time graph or in other words updatable line, which will be refreshed at some frequency.

This could be achieved by using command 'ion()'[26] and some other commands for realizing graph update in the following manner described in Code. 3

```
ion()
f, ax= subplots()
line1, = ax.plot([],[])
line2, = ax.plot([],[])
ax.set_autoscaley_on(True)
ax.grid()

def update_line(time,input,output):
    T.append(time);   output.append(y);  input.append(x)
    line1.set_xdata(T); line1.set_ydata(input);
    line2.set_xdata(T); line2.set_ydata(output);
    ax.relim();
    ax.autoscale_view();
    draw()
```

Code. 3: Online vizualization of measured data-set

For instance, lines could be updated by function call 'update_line' in a main code where frequency of update should be specified.

Visualization of obtained data-set with this method is time consuming, therefore only 5 samples per second could be obtained with constant time sampling.

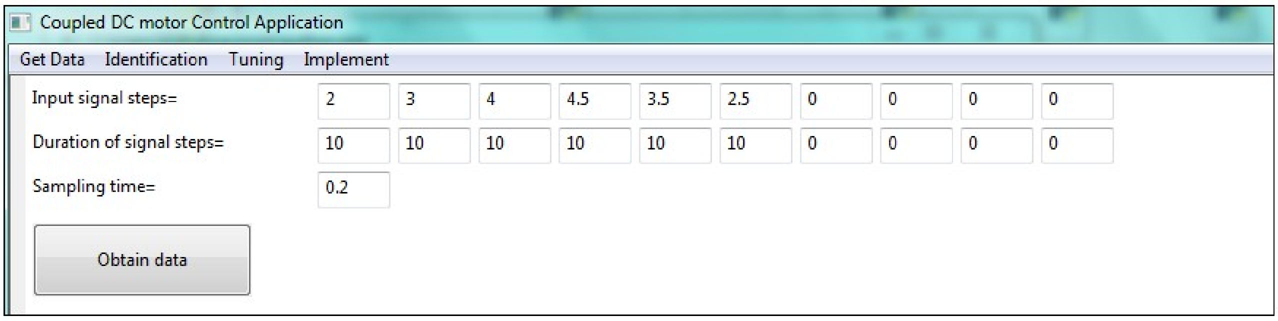
7 Graphical User Interface (ASPI Kit Extension)

The above SW solution was implemented as extension of currently exsiting open-source application. Aspi Kit is an open-source SW application in Python that is being developed by the ASPICC group (www.aspicc.fs.cvut.cz) for demonstration purposes of adaptive identification and control and other adaptive signal processing methods [27].

7.1 New Extension

For simplicity of interaction with a script I made an application which requires the wxPython, Matplotlib, Numpy and U3 modules for full functionality of all features to this software.

With this application it is possible to measure data with an option to set up variable step size and specify the duration of each particular step as shown in the following figure.


Coupled DC motor Control Application

Get Data Identification Tuning Implement

Input signal steps=	2	3	4	4.5	3.5	2.5	0	0	0	0
Duration of signal steps=	10	10	10	10	10	10	0	0	0	0
Sampling time=	0.2									

Obtain data

Fig. 16: GUI designed for experimental data acquisition. GetData Panel. Extension to ASPI Kit

At the end of the code execution figure of input (voltage supplied by LabJack) and output (system response to a given input) signals versus time(instance at which sample was taken) will appear as show in Fig. 17 and these data will be saved at the same time.

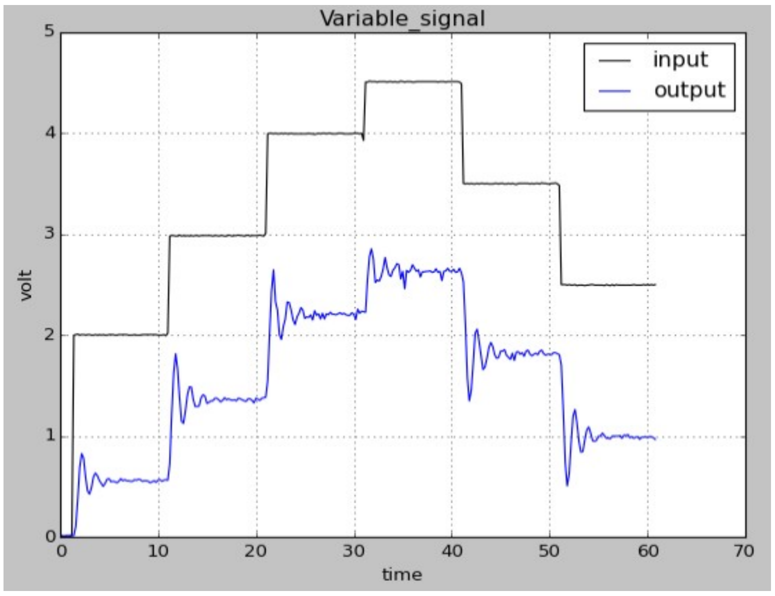
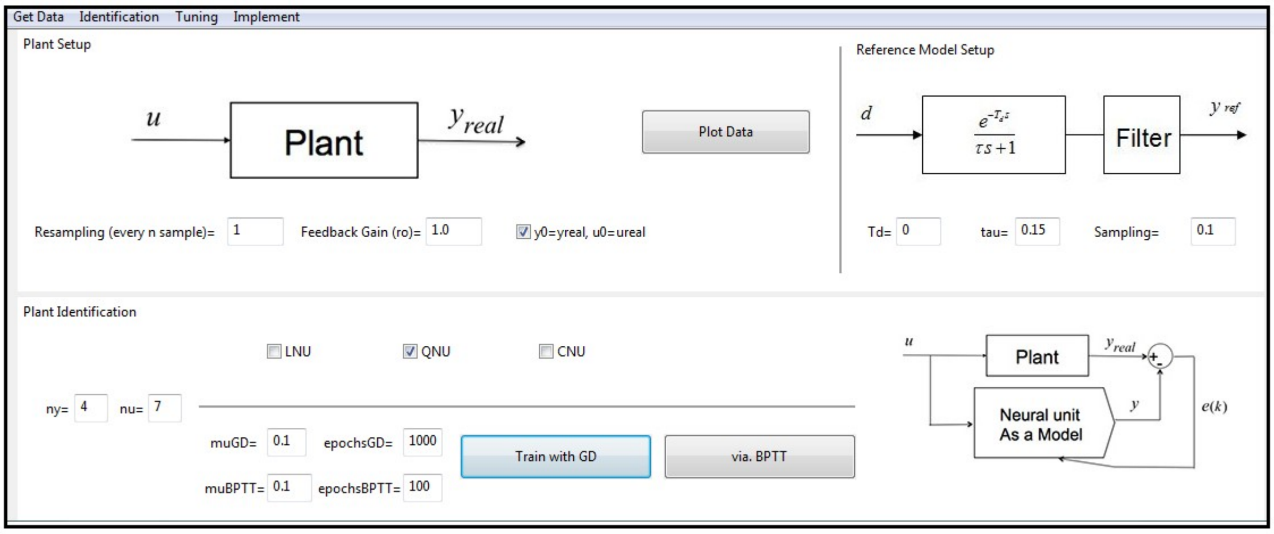


Fig. 17: Plots of obtained data sets (Plant response to variable voltage step inputs)

The second part of this application -'Identification', is dedicated to system identification. This module is split into 2 distinct panels. The first allows the user to see a visualization of uploaded process data by pressing “Plot Data” button. For the purpose of control and precise identification the user may require re-sampling of the data. Therefore there is an option to re-sample data by typing value in to “Resampling” box. More over, there is a choice for selection of feedback gain and initial values of neural output as shown in Fig. 18. And the second panel in for identification itself. Here user must fill in the learning rate for usage of gradient descent algorithm and number of epochs to tune the respective training algorithms. As the values of the learning rate and number of epochs differ between the RTRL and BPTT methods, separate edits are placed for each within each panel. There is also a feature for the user to define the length of the variables used in the model polynomial equations for the respective controller.



Resampling (every n sample)= 1
Feedback Gain (ro)= 1.0
[✓] y0=yreal, u0=ureal
Td= 0
tau= 0.15
Sampling= 0.1
[ ] LNU [✓] QNU [ ] CNU
ny= 4
nu= 7
muGD= 0.1
epochsGD= 1000
muBPTT= 0.1
epochsBPTT= 100

Fig. 18:GUI applications Identification panel adapted from ASPI Kit for plant identification

The main goal for using these adaptive methods, is to train the respective neural weights to fit the shape of the process data. The process can then be said to be identified successfully when the neural model resembles closely or even superimposes to original data for which the neural model is trained for as shown in Fig. 19 below.

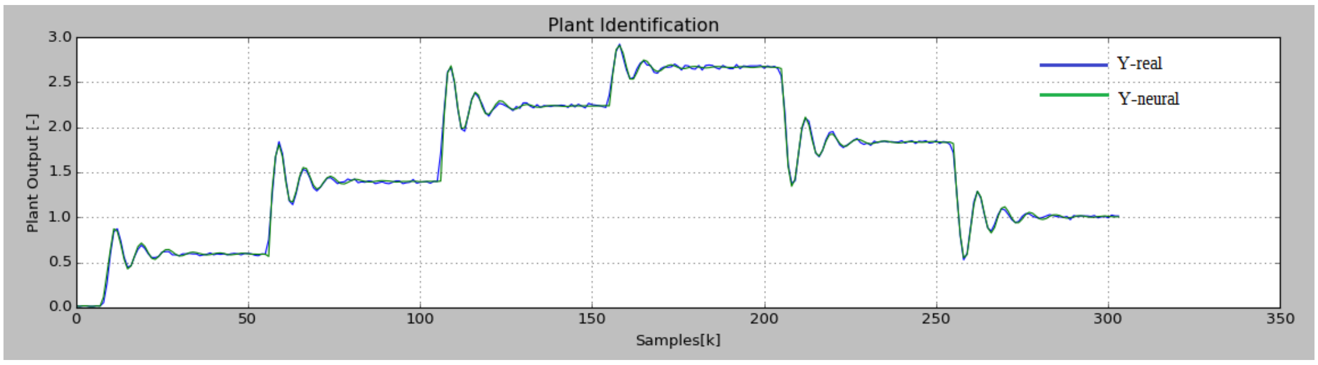
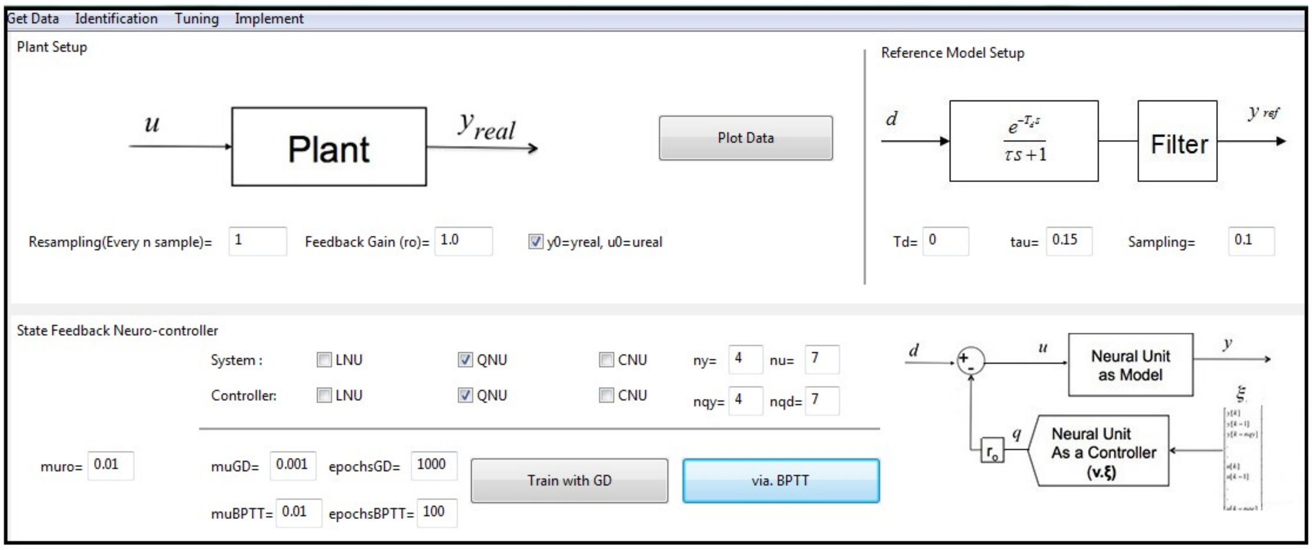


Fig. 19: Identified neural model (green line)

As soon as successful identification is achieved, then by means of third part -'Tuning' of application user can extend adaptive control method onto the newly identified neural model and test which method is suitable for realization of control on the engineering process.



Resampling(Every n sample)= 1 Feedback Gain (ro)= 1.0 ☑ y0=yreal, u0=ureal

Td= 0 tau= 0.15 Sampling= 0.1

System : ☐ LNU ☑ QNU ☐ CNU ny= 4 nu= 7

Controller: ☐ LNU ☑ QNU ☐ CNU nqy= 4 nqd= 7

muro= 0.01 muGD= 0.001 epochsGD= 1000

muBPTT= 0.01 epochsBPTT= 100

Fig. 20: GUI applications Tuning panel adapted from ASPI Kit for controller tuning

Both the identification and control processes are based on the Gradient Descent method of training the respective neural models weights. However, a second form of training incorporated into the application is the batch training method, which is form of training the neural models weights over each epoch, via an extension of the Gradient Descent method with the famous Levenberg-Marquardt equation.

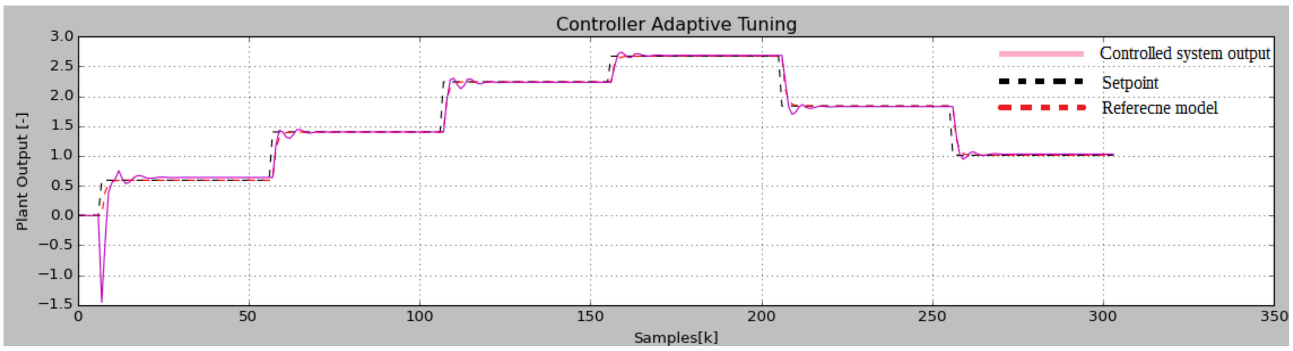
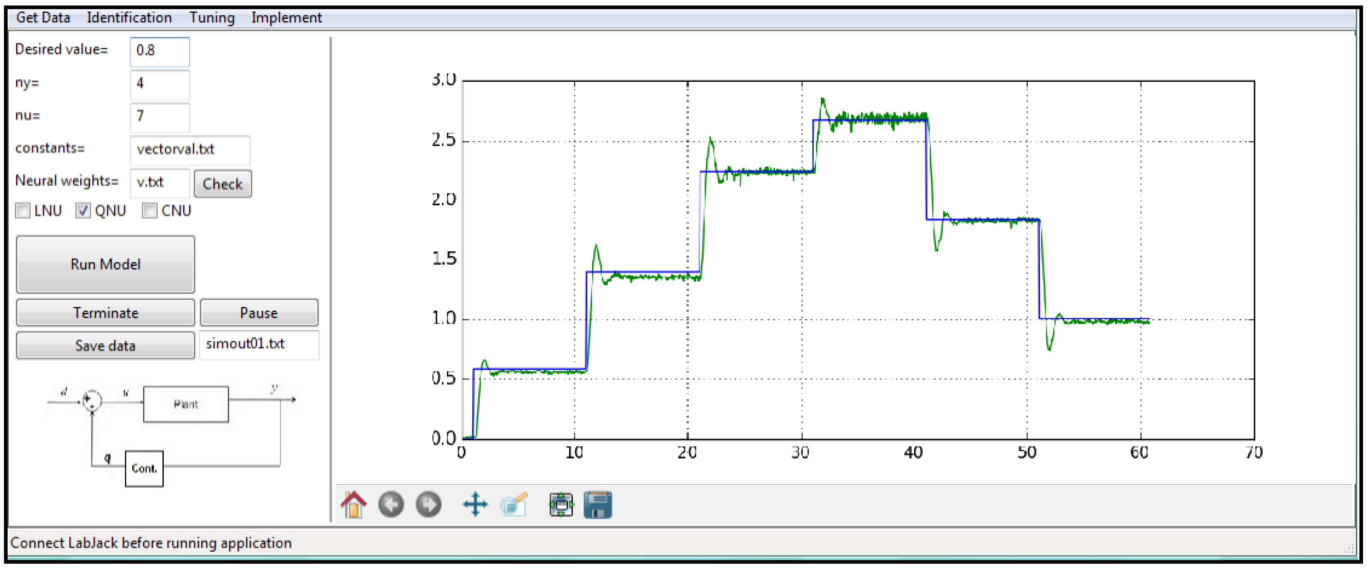


Fig. 21: Controlled system output (pink). Neural-Controller tuned with QNU via LM.

When successful controller tuning is achieved as shown in Fig. 21, it is possible to implement newly trained controller to a real system by means of forth part of an application- 'Implement'.

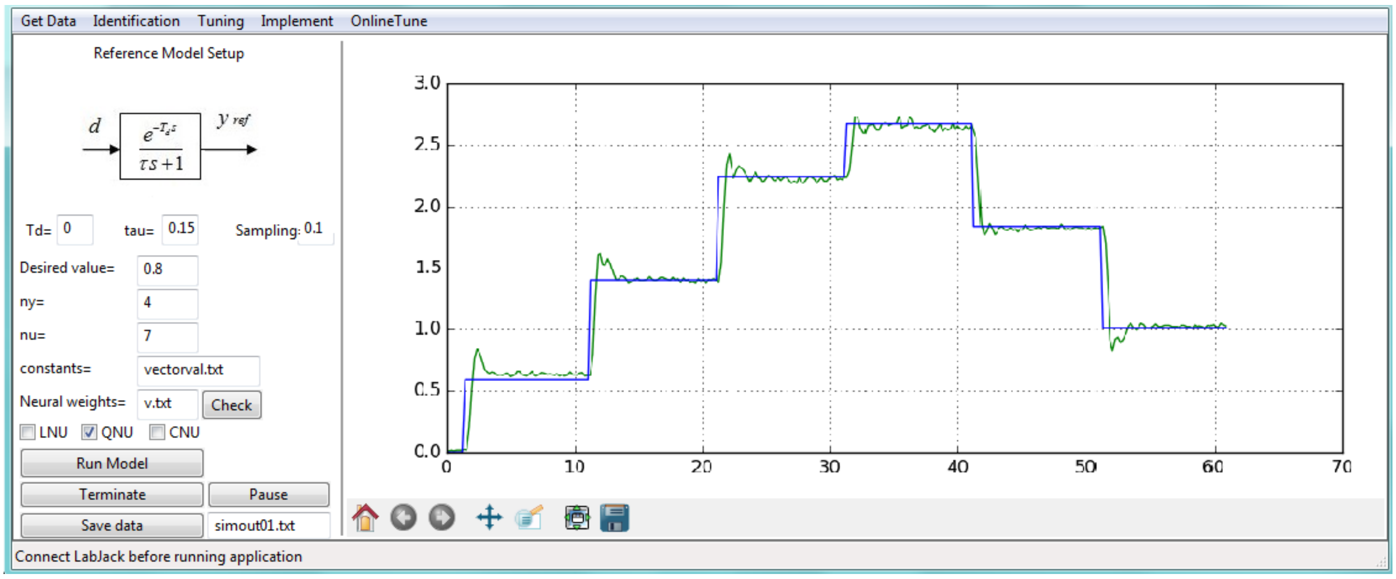


Desired value= 0.8
ny= 4
nu= 7
constants= vectorval.txt
Neural weights= v.txt
[✓ QNU]
simout01.txt
Connect LabJack before running application

Fig. 22: GUI application Implement Panel designed for implementation of trained algorithms to a real plant with online visualization of controlled process. QNU controller with constatn weights trained via LM. Extention to ASPI Kit.

In order to run this section it is necessary to connect the LabJack, otherwise it wont work. At the end of program, LabJack will automatically be switched off. Then dynamically plotted data could be saved as an Image and as a '.txt' file.

Same algorithms could be used for online learning and tuning with the last panel of the application called -'Online Tune' which is shown in Fig. 23. As it could be observed, adaptive tuning algorithms need time at the beginning to learn and then overall performance of system control is significantly better if it is compared with constant parameters of Neural-Controller.



Td= 0 tau= 0.15 Sampling: 0.1
Desired value= 0.8
ny= 4
nu= 7
constants= vectorval.txt
Neural weights= v.txt
☐ LNU ☑ QNU ☐ CNU
simout01.txt
Connect LabJack before running application

Fig. 23: GUI application OnlineTune Panel designed for implementation of trained GD algorithm with continuous adaptation to a real plant with online visualization of controlled process. QNU controller with apabtive weights trained via GD. Extention to ASPI Kit.

7.1.1 Notes for Algorithms Implementation

Levenberg-Marquardt algorithm is appropriate for offline tuning of Neural Controller and using it without any further update (constant neural weights). This approach is good for systems with stable behavior, because NC tuned with LM is tolerant to high disturbances or some change in systems behavior, in other words weights are not going to be updated for adaptation to new environment.

Gradient Descent algorithm is used for both offline and online tuning of a Neural controller. The best way of exploitation of NC tuned with GD is to pretrain neural weights before implementation of controller to real system. NC will be continuously adopted during exploitation, what is good for systems, output of which is highly dependent on material properties which are degrading over time (fatigue, elongation etc.).

8 Connection of LabJack to the Plant

For connection of LabJack to a coupled motor system, first of all safety measurements should be followed.

1. Switch off the system
2. Unplug all cables which could cause a voltage drop (measuring card)
3. Connect LabJack to a system as shown in Fig. 24 A) and B)
4. Connect LabJack to computed and switch on the system

Important note: ground (GND)should be common for the whole system.

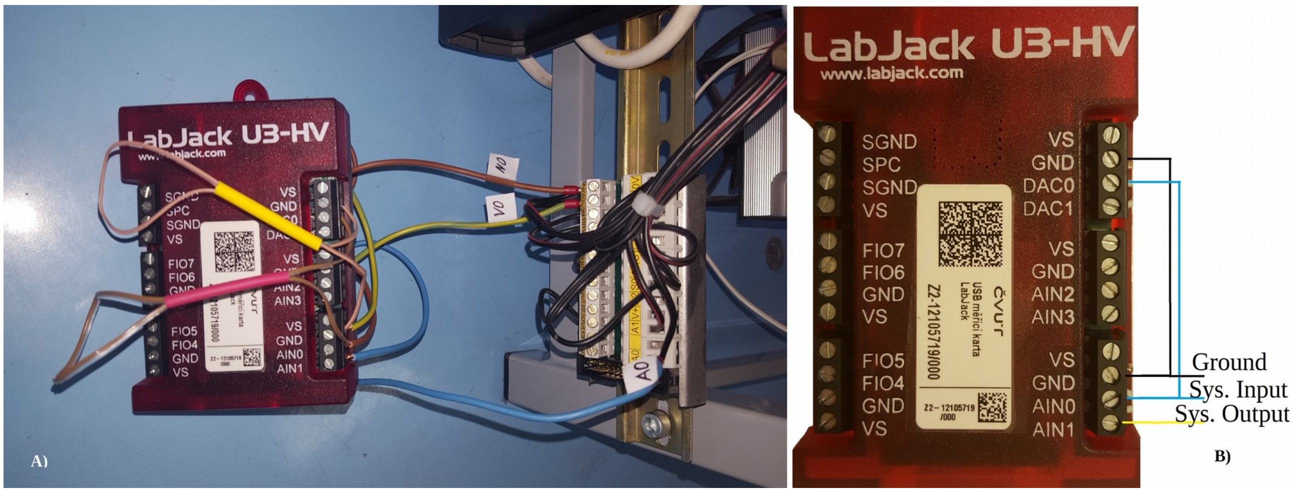


Fig. 24: Connection of LabJack

LabJack will produce input voltage to motor through DAC0 channel and this signal is in parallel connection with AIN0 channel for registering produced voltage value (in case if there is a problem, e.g. with ground, LJ will not produce nominated voltage). The output signals produced by the system -output voltage of tachometer is plugged to AIN1.

9 Plant Identification

For comparison of identification methods and identification order I used same number of learning epochs -1000, same amount of input and output values (ny,nu)-(4,7) and same learning rate μ -0.5 for each separate method and order of algorithm.

9.1 LNU Algorithm Trained with the GD method

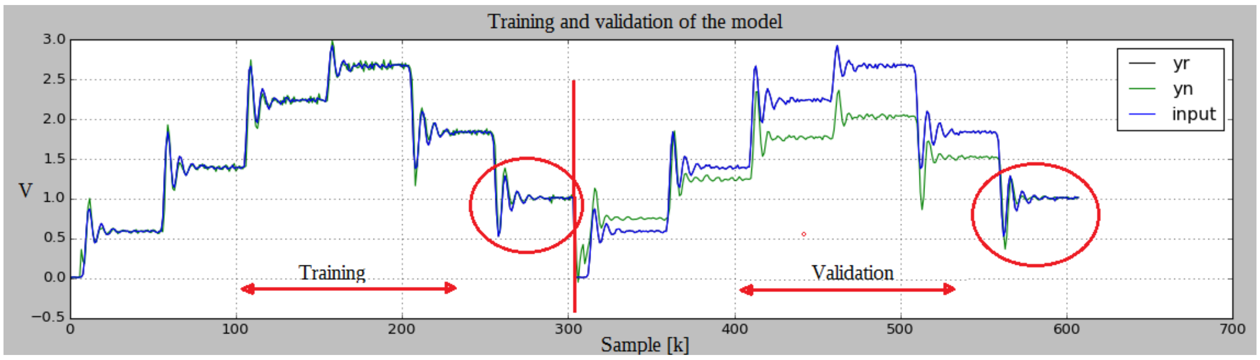


Fig. 25: Plant identification with LNU using GD showing the nature of sample-by-sample offline learning.

From Fig. 25 it is clearly seen that step-by-step training of the neural model works pretty good, but validation of the model (using constant trained neural weight)shows that only last step (circled red) was memorized by neurons. Due to fundamental structure of the Gradient Descent algorithm, identified constant weights could be used for systems with fixed working point. As for given system it is better to use adaptive (with weight update) identification.

9.2 LNU Algorithm Trained with LM method

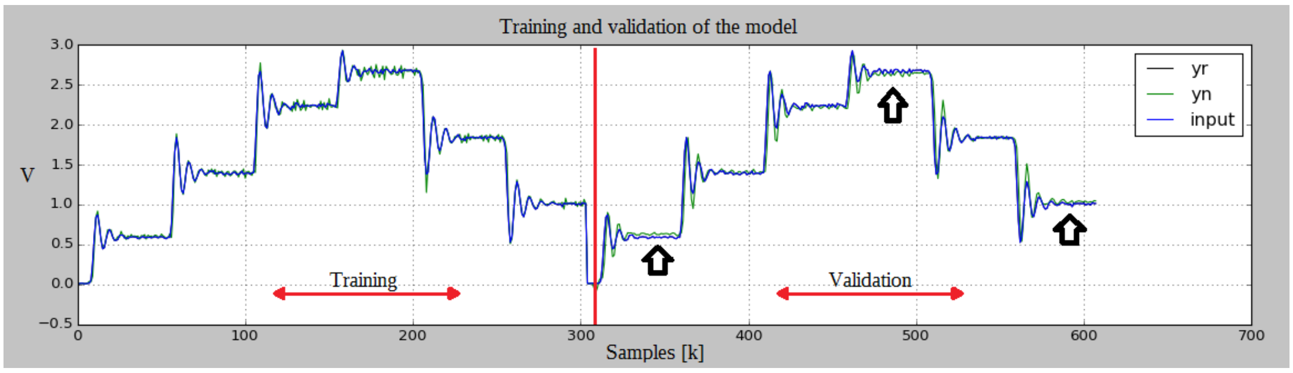


Fig. 26: Plant identification with LNU using LM showing the nature of batch learning.

From figure Fig. 26 we can observe that Lewenberg-Marquardt algorithms perform much better with constant weights and variable operating point. However, from both figures it is seen that Linear algorithm is not able to perfectly catch the non-linearity and the most obvious ones are indicated with black arrow.

9.3 Online Adaptive QNU Algorithm Trained with GD method

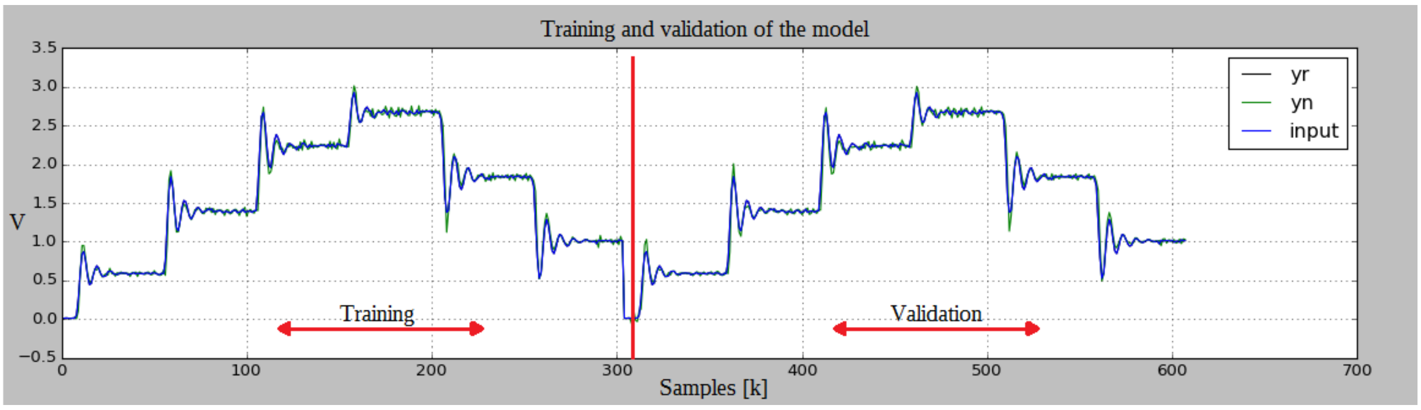


Fig. 27: Plant Identification with QNU using GD showing the nature of adaptive sample-by-sample online learning

Identification of plant model in this particular case with adaptive Gradient Descent and Quadratic neuron gave very accurate result.

9.4 Adaptive QNU Algorithm Trained with LM method

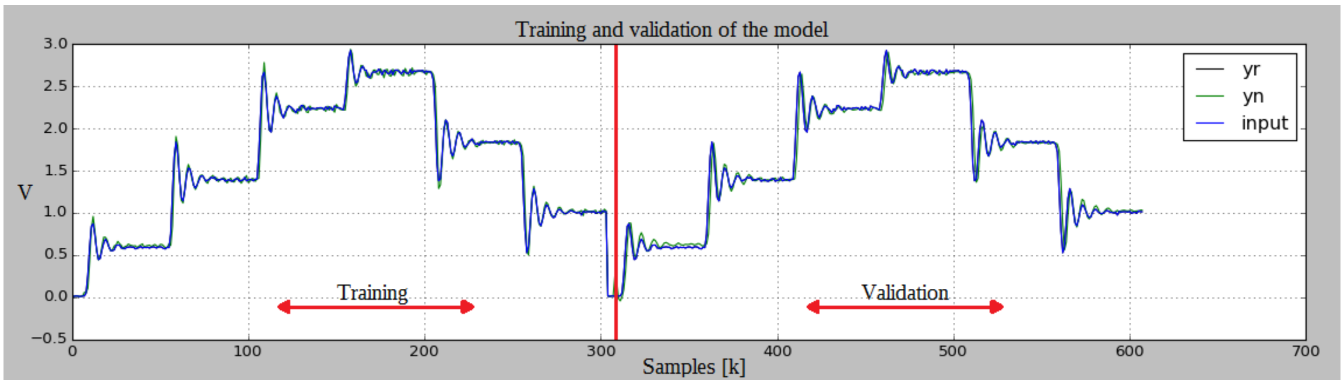


Fig. 28: System Identification with QNU using LM shows the superiority over LNU algorithm.

Results of system identification with Quadratic neuron are even more satisfactory than it was with GD. However, in contrast to GD, this algorithm will not perform that great if process will be expose to changes in dynamics e.g. high disturbance, fatigue. From here we can conclude that QNU algorithm is sufficient for identification of coupled motor system.

Since error of identification is small and not observable as it could be seen form Fig. 27 and Fig. 28 It is better to see the efficiency in terms of summation of square errors (SSE) of the last trained epoch in the following Table 6:

	LM	**GD**
LNU	0.6	0.8
QNU	0.24	0.26
CNU	0.22	0.24

Table 6: Table of SSE with respect to method and order of algorithm of identification.

10 Plant Control

In this section I am going to show real life implementation of control algorithms as it was described in section 5.3

10.1 Training Neuro-Controller Weights

In previous section I was describing system identification because system parameters (neural weights $\mathbb{W}$) are playing important role for offline tuning. Identified system will be simulating a real plant as it is shown in Fig. 29 and denoted as 'Neural unit as a model'

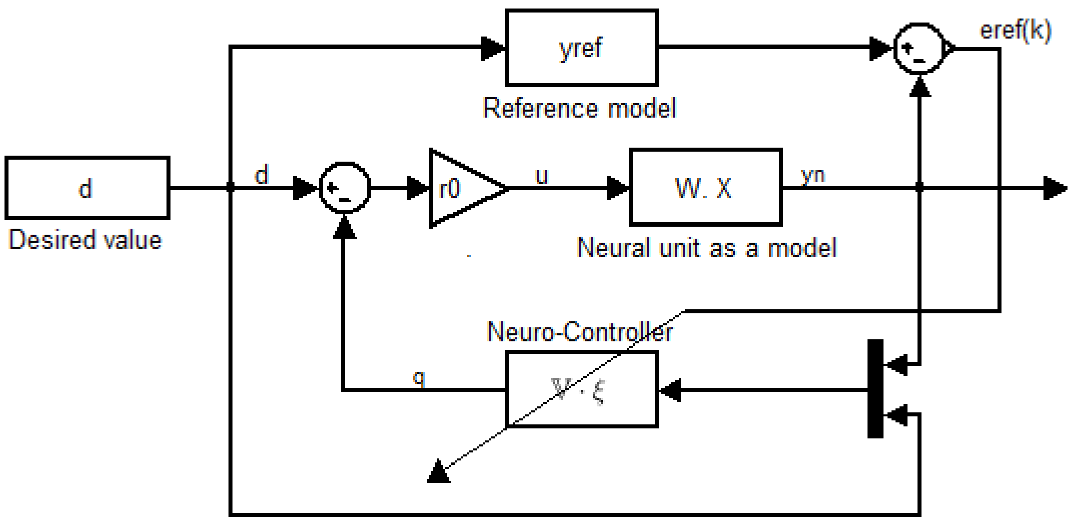


Fig. 29: Schematics of offline controller tuning

For precise controller tuning it is preferably to have a reference model y_{ref}, which works a s a filter to a desirable value. The principal schematics of algorithm to obtain y_{ref} is as follows.

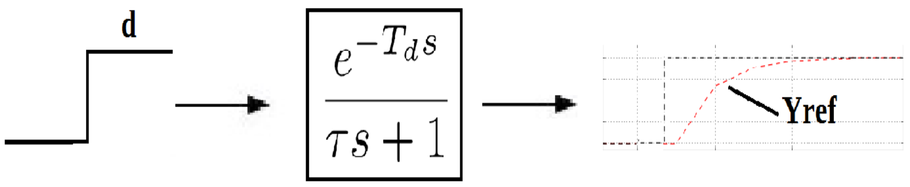


Fig. 30: Principal of algorithm to obtaing reference model

Reference model is a desirable behavior of a system after tuning, therefore, it should be feasible by a real system.

10.2 Adaptive Training of Algorithms with GD and LM

Adaptive training of algorithms is based on the error between reference model and the output of a system. As the auxiliary tool I am using adoptive proportional term r_0 for better performance, value of which is also dictated by e_{ref}.

In the following Fig. 31 is a magnified view of adaptive controller 'Tuning' section of GUI application. By experimental manipulation with tuning instruments (learning rate, training epoch etc.) it is possible to achieve better and worse results, it is trial and error procedure. As soon as we obtain satisfactory result of tuning which is denoted as 'Controlled Output' in Fig. 31, we can proceed to 'Implementation' part, where tuned weights will be automatically used by GUI application for real plant control.

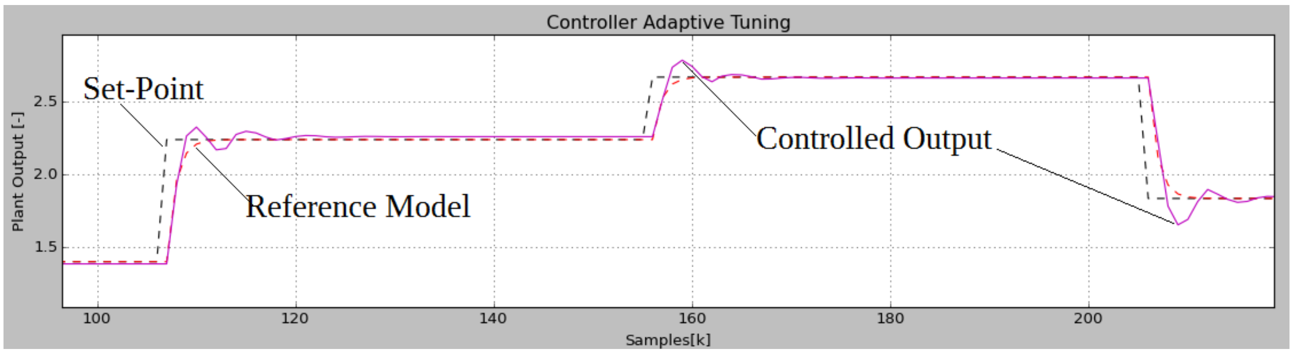


Fig. 31: Magnified view of QNU controller tuning via LM

Since we can not easily judge about efficiency of results of different algorithms by just looking to figure, I am presenting Table 7 below to show how well performing every single algorithm with respect to applied method in terms of Summation of Square Errors.

	GD	LM
LNU	3.56	3.4
QNU	3.1	3.09
CNU	3.08	3.06

Table 7: Table of SSE with respect to method and order of algorithm of Controller tuning

Due to the fact that there is no significant difference between Quadratic and Cubic Neural Units in contrast to Linear neural unit, we can state that QNU is optimal for controlling this system.

10.3 Implementing Neuro-QNUcontroller with constant weights LM

For system control in this section I will use fixed controller parameters. This will work fine if the system will not be exposed to physical changes. As it could be observed form Fig. 32 Neuro-Controller performs fast (approx. 1,5 seconds to steady state) and for the whole set of step changes, deviation from set-point is less then 5%. The overall behavior is following tuned pattern as it was shown in Fig. 31

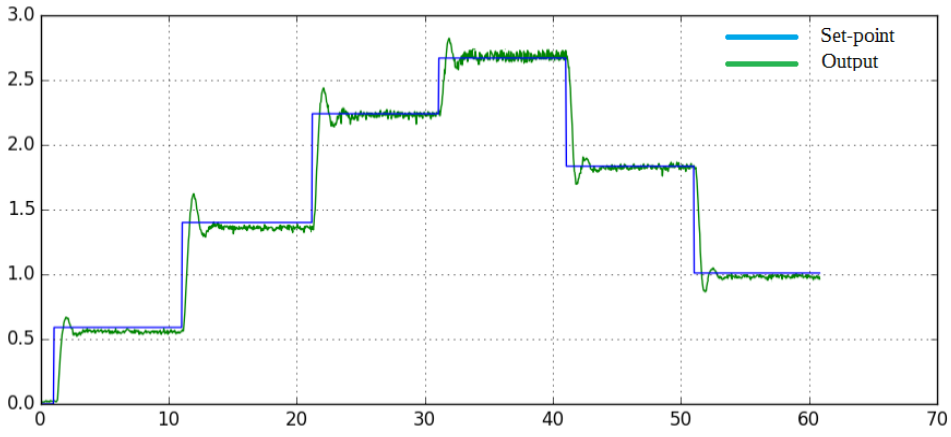


Fig. 32: Controlled system with fixed controller parameters. Blue line is a set-point, Green line -system output

10.4 Implementing Neuro-QNUcontroller with continuously adapting weights GD

In this section I am implementing GD algorithms with continuous training of neural weights all over the process. This method allows the controller fast adaptation to changes in the system behavior. As it could be observed form Fig. 33 that Adaptive Neuro-QNUcontroller performs better then the controller with fixed parameters -smaller deviation from desired set-point, faster response and less overshoot.

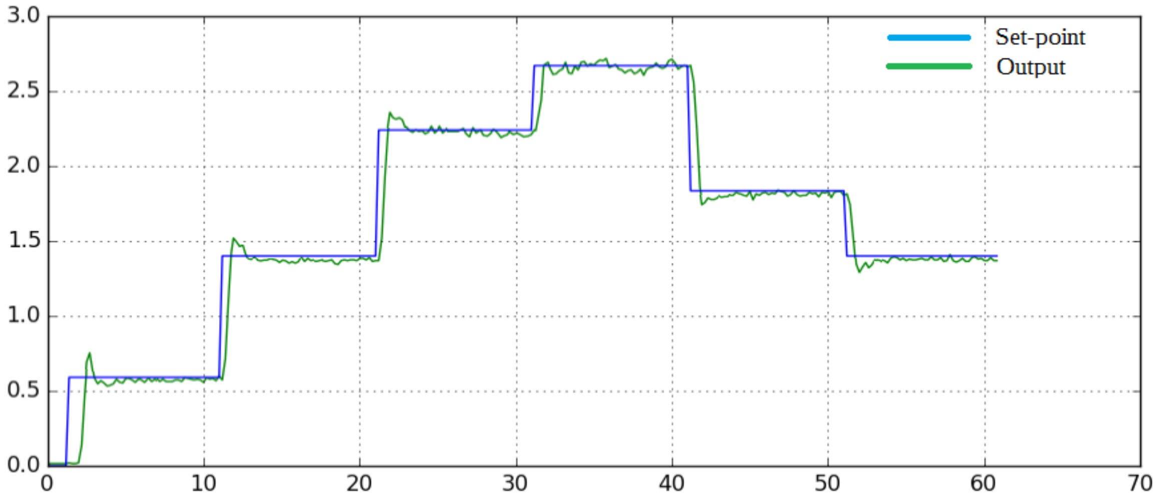


Fig. 33: Controlled system. With adaptable controller parameters. Blue line-setpoint, Green line -system output

11 Discussion

Further improvements could be achieved with neural network approach by reanalyzing new series of experiments with more efficient selection of sampling time interval, learning rates and length of neural inputs. It is necessary to be careful while working with algorithms and their parameters. For instance, Levenberg-Marquardt algorithm is appropriate for offline tuning of Neural Controller and using it without any further update (constant neural weights). This approach is good for systems with stable behavior, because NC tuned with LM is tolerant to high disturbances or some change in systems behavior, in other words weights are not going to be updated for adaptation to new environment. Gradient Descent algorithm is used for both offline and online tuning of a Neural controller. The best way of exploitation of NC tuned with GD is to pretrain neural weights before implementation of controller to real system. NC will be continuously adopted during exploitation, what is good for systems, output of which is highly dependent on material properties which are degrading over time (fatigue, elongation etc.). Sampling rate is trivial for the identification and control because it is directly related to size of neuron input vector. For instance, if sampling rate is high e.g. 100 samples per second, for system identification and control we will require around 60 previous inputs and outputs to a neuron vector $\mathbb{X}$, what is not efficient for calculation. Therefore, re-sampling is required so that the size of neuron vector will be in range of 10 to 20 (inputs and outputs together). Another important point, for controller identification, is that sampling rate for controller should be the same as in system identification. With a remark for the fact that learning rate μ_c and size of a neuron vector ξ for a controller could be different from those for identification. The last point to mention is that learning rate μ essentially corresponds to the speed of learning. If the value is high, system will be learned in a fast rate. If the value is small, system will be learned in slower rate. Since neural networks are not bounded to be implemented alone but to be used as an auxiliary tool with PID regulator as shown in Fig. 34 below.

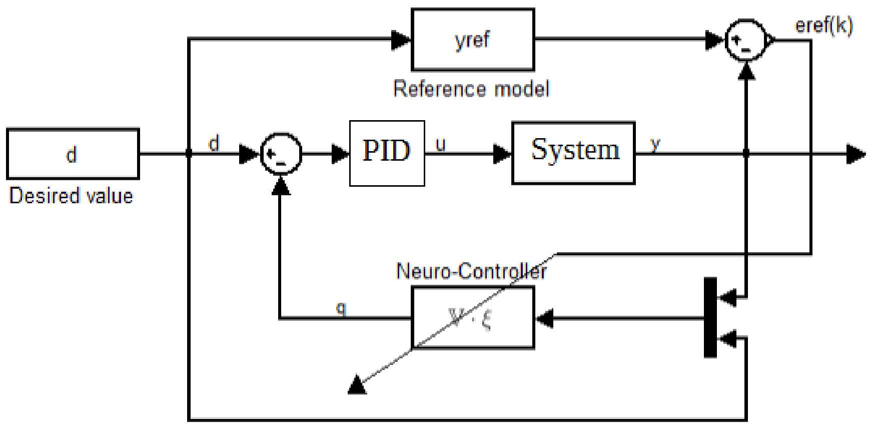


Fig. 34: Schematics of using PID with Neuro-Controller as proposal for future investigations

Where system will be identified with already applied PID and neural controllers work will be focused on reduction of time constant and elimination of overshoots.

In Appendix I have included schematics of Simulink models for analysis and implementation of neural approach to system control, what gives fundamental idea of introduction of neural units to a controlled system and to use this method with varieties of other measuring devices.

12 Conclusion

After completion of the key objectives within the scope of the thesis, I want to outline main accomplished points. First and the most important achievement is that I have successfully implemented nonlinear adaptive control algorithms to a coupled motor system with a flexible joint. I have designed, implemented and tested HW and SW solution using LabJack and Python with GUI (implemented as ASPI Kit extension). In relation to experimental measurements, I can stated that neural approach is convenient for control of nonlinear oscillatory system as well as for its identification. Not to be strictly bounded to idea of controlling the given system with neural networks, I made series of experiments with conventional PID controller, to compare and see the performance of both. Results of experiments showed that neural controller is as good as PID if not better and that new approach improves the dynamic characteristics of controlled system, as it was in works [17][16]where neural networks demonstrated perfect performance of control of non-linear system. Besides that, I summarized literature review of recently published works regarding adaptive algorithms. I have described the process form physical and mathematical points of views and built a virtual analogue of coupled motor system in Simulink environment, what could be helpful for those who will make a further research in his field.

Reference

[1] “Create Simple Model - MATLAB & Simulink.” [Online]. Available: http://www.mathworks.com/help/simulink/gs/create-a-simple-model.html. [Accessed: 10-Jun-2016].

[2] Osvald Modrlák and Lukáš Hubka, “Otáčky DC motoru – „DC motor se zátěží“.” .

[3] “DC motors and generators.” [Online]. Available: http://pemclab.cn.nctu.edu.tw/W3lib/books/ElectroCraft85/ch2.dcmotor.pdf. [Accessed: 14-Dec-2015].

[4] “Control Tutorials for MATLAB and Simulink - Motor Speed: System Modeling.” [Online]. Available: http://ctms.engin.umich.edu/CTMS/index.php?example=MotorSpeed&section=SystemModeling. [Accessed: 08-Dec-2015].

[5] “Control Tutorials for MATLAB and Simulink - Motor Position: State-Space Methods for Controller Design.” [Online]. Available: http://ctms.engin.umich.edu/CTMS/index.php?example=MotorPosition&section=ControlStateSpace. [Accessed: 14-Dec-2015].

[6] “Prediction of the twisting moment and axial force in a circular rubber cylinder for combined extension and torsion based on the logarithmic strain approach - Kakavas - 2008 - Journal of Applied Polymer Science - Wiley Online Library.” [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1002/app.28685/full. [Accessed: 24-Mar-2016].

[7] C. O. H. Landon M. Kanner, “On Extension and Torsion of Strain-Stiffening Rubber-Like Elastic Circular Cylinders,” Oct. 2008.

[8] “What Are State-Space Models? - MATLAB & Simulink.” [Online]. Available: http://www.mathworks.com/help/ident/ug/what-are-state-space-models.html. [Accessed: 10-Jun-2016].

[9] P. Klan and R. Gorez, *Process Control*. Prague, 2011.

[10] R.Toscano, “A simple robust PI/PID controller design via numerical optimization approach.” [Online]. Available: http://rosario.toscano.free.fr/PidRob.pdf. [Accessed: 12-Apr-2016].

[11] “PID systems tutorial.” [Online]. Available: http://webber.physik.uni-freiburg.de/~hon/vorlss02/Literatur/Ingenieurswiss/pid/pid+matlab/PID%20systems%20tutorial.htm. [Accessed: 30-May-2016].

[12] M. Gupta, L. Jin, and N. Homma, *Static and Dynamic Neural Networks: From Fundamentals to Advanced Theory*. John Wiley & Sons, 2004.

[13] Ivo Bukovsky, “Modeling of Complex Dynamic Systems by Nonconventional Artificial Neural Architectures and Adaptive Approach to Evaluation of Chaotic Time Series.” Czech Technical University in Prague, Faculty of Mechanical Engineering, (PhD thesis), 2007.

[14] Madan M. Gupta, Ivo Bukovsky, Noriyasu Homma, Ashu M. G. Solo, and Zeng-Guang Hou, “Fundamentals of Higher Order Neural Networks for Modeling and Simulation,” in *Artificial Higher Order Neural Networks for Modeling and Simulation*, Ming Zhang, Ed. Hershey, PA, USA: IGI Global, 2013, pp. 103–133.

[15] I. Bukovsky and N. Homma, “An Approach to Stable Gradient Descent Adaptation of Higher-Order Neural Units,” *IEEE Trans. Neural Netw. Learn. Syst.*, accepted 2016.

[16] I. Bukovsky, P. Benes, and M. Slama, “Laboratory Systems Control with Adaptively Tuned Higher Order Neural Units,” in *Intelligent Systems in Cybernetics and Automation Theory*, R. Silhavy, R. Senkerik, Z. K. Oplatkova, Z. Prokopova, and P. Silhavy, Eds. Springer International Publishing, 2015, pp. 275–284.

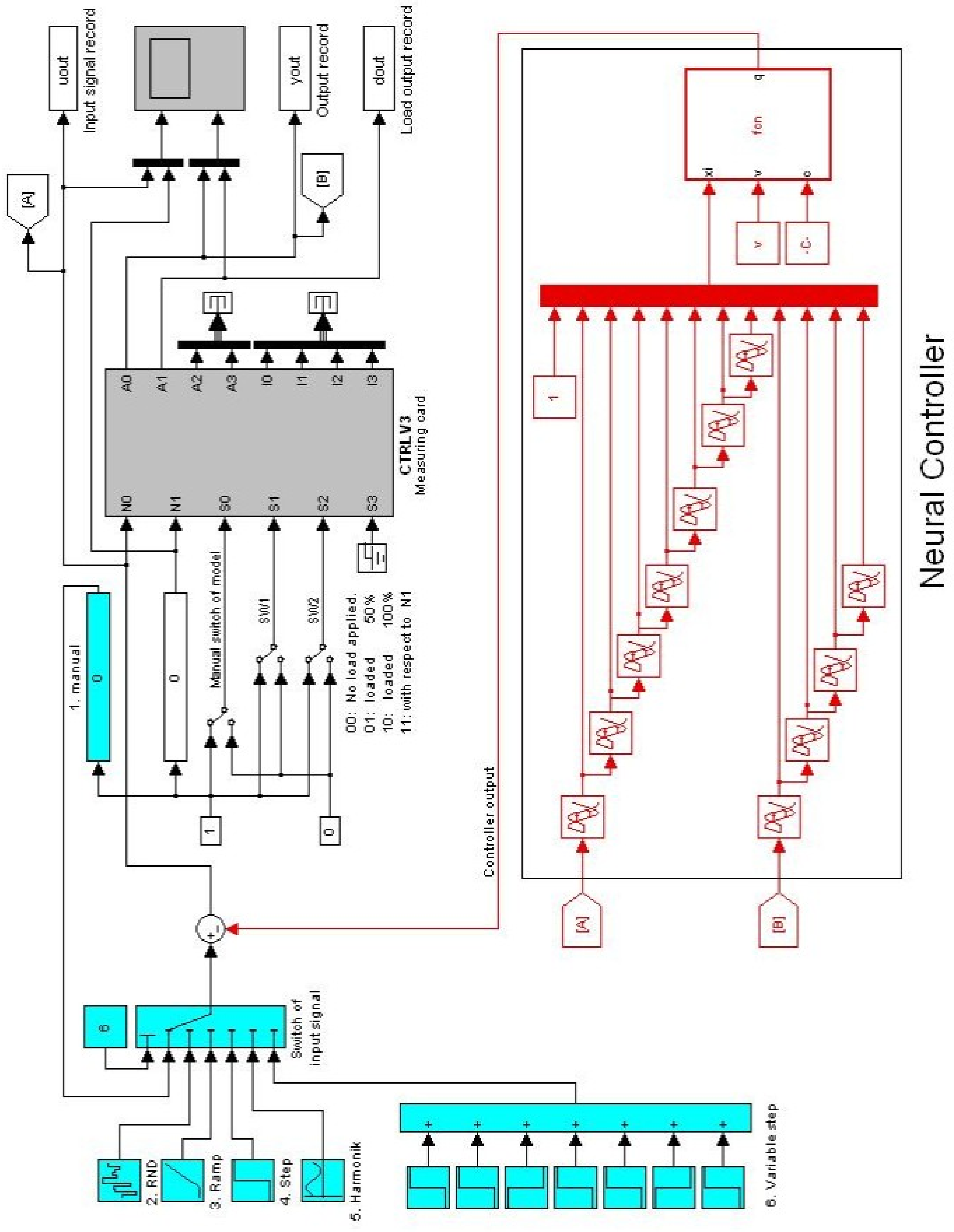
[17] I. Bukovsky, N. Homma, L. Smetana, R. Rodriguez, M. Mironovova, and S. Vrana, “Quadratic neural unit is a good compromise between linear models and neural networks for industrial applications,” in *2010 9th IEEE International Conference on Cognitive Informatics (ICCI)*, 2010, pp. 556–560.

[18] “Jacobian matrix and determinant,” *Wikipedia, the free encyclopedia*. 08-Jun-2016.

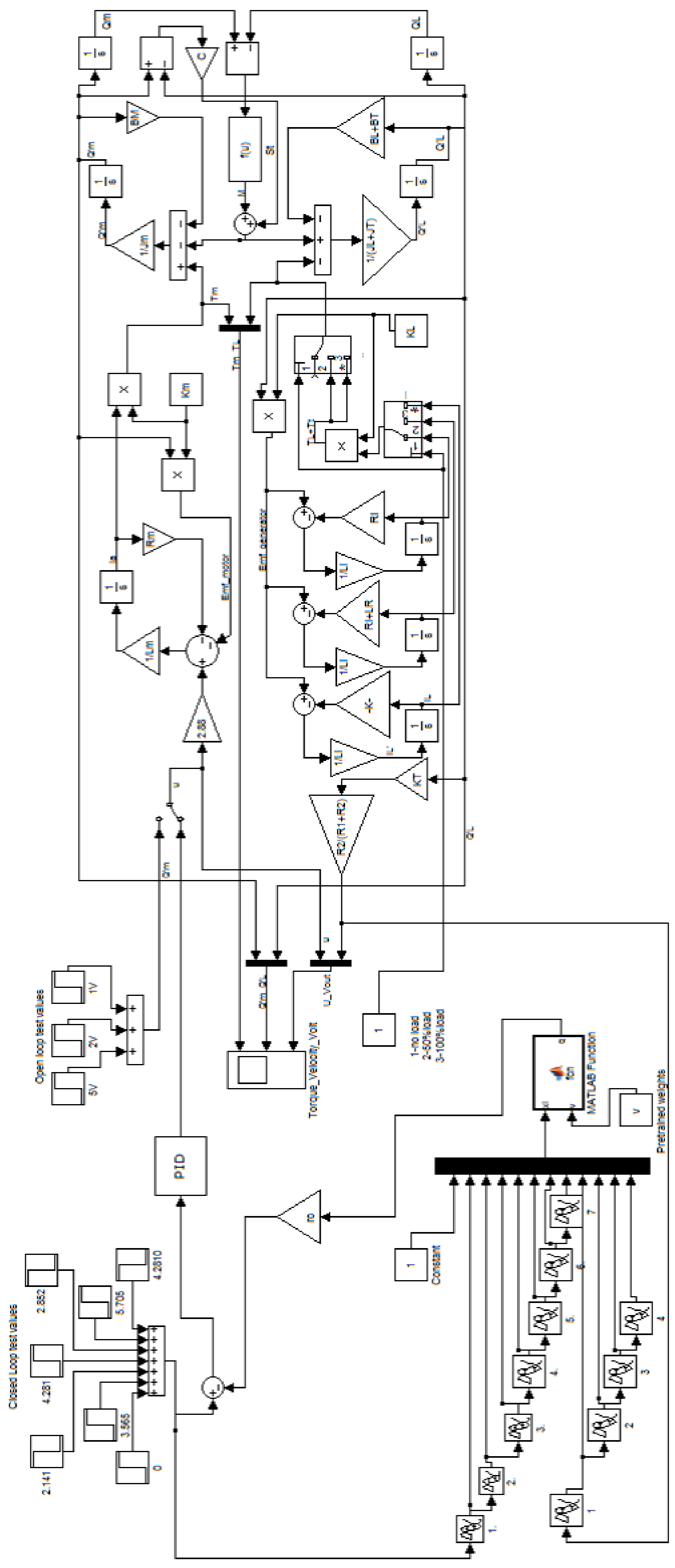
[19] “Python Release Python 3.5.0 | Python.org.” [Online]. Available: https://www.python.org/downloads/release/python-350/. [Accessed: 10-Jun-2016].
[20] “wxGlade: a GUI builder for wxWidgets/wxPython.” [Online]. Available: http://wxglade.sourceforge.net/tutorial.php. [Accessed: 10-Jun-2016].
[21] “LabJackPython (Windows UD, Mac, Linux) | LabJack.” [Online]. Available: https://labjack.com/support/software/examples/ud/labjackpython. [Accessed: 10-Jun-2016].
[22] “U3 Series | LabJack.” [Online]. Available: https://labjack.com/products/u3. [Accessed: 10-Jun-2016].
[23] “Low-level Commands Quickstart | LabJack.” [Online]. Available: https://labjack.com/support/software/examples/ud/labjackpython/low-level. [Accessed: 14-Dec-2015].
[24] “3.2.1 - Streaming Digital Inputs, Timers, and Counters | LabJack.” [Online]. Available: http://li51-60.members.linode.com/support/u3/users-guide/3.2.1. [Accessed: 10-Dec-2015].
[25] “LJStreamUD - Windows Only | LabJack.” [Online]. Available: https://labjack.com/support/software/applications/ljstreamud. [Accessed: 10-Jun-2016].
[26] “Usage — Matplotlib 1.5.1 documentation.” [Online]. Available: http://matplotlib.org/faq/usage_faq.html. [Accessed: 10-Jun-2016].
[27] P. . Benes, “Software Application for Adaptive Identification and Controller Tuning,” 2013.

Appendix

Neural-Controller in Matlab with measuring card



Neural Controller in Matlab with PID in simulated model



Matlab Function Code for Simulink Model

```
%V, is a vector of pretrained weights, have to be uploaded
separately

function q = fcn(xi,v)
u=xi(2:8);
y=xi(9:12);
meanu=mean(u);
meany=mean(y);
stdu=std(u);
stdy=std(y);
if stdu==0
    stdu=0.0001;
end
if stdy==0
    stdy=0.0001;
end

xi(2:length(y)+1)=(y(1:length(y))-meany)/3/stdy;
xi(length(y)+2:12)=(u(1:length(u))-meanu)/3/stdu;

n=0;
for i=1:length(xi)
    for j=i:length(xi)
        n=n+1;
    end
end
index=1;
colx=zeros(78,1);
for a=1:length(xi)
    for b=a:length(xi)
        colx(index)=xi(a)*xi(b);
        index=index+1;
    end
end

if dot(v,colx)*3*stdy+meany==NaN
    q=0;
else
    q=dot(v,colx)*3*stdy+meany;
end

q = q;
```

MatLab Script Formulation of State-Space Model:

```
close all; clc;

Lm = 1.6*10^-3; Rm = 0.605; Km = 0.72; Jm = 2.8*10^-2; BM = 4.6*10^-4;
Ll = 1.6*10^-3; Rl = 0.605; KL = 0.72; JL = 2.8*10^-2; BL = 4.6*10^-4;
JT = 2.8*10^-5; KT = 1.2; BT = 2.3*10^-4;
LR = 6.6; R1 = 656300;R2 = 114700;
C=0.00;
St=5; %expected value. Variable parameter can not be implemented


A = [0  1 0 0 0 0;
     -St/Jm -(BM+C)/Jm St/Jm C/Jm Km/Jm 0;
    0 0 0 1 0 0;
     St/(JL+JT) C/(JL+JT) -St/(JL+JT) -(BL+BT+C)/(JL+JT) 0 -KL/(JL+JT);
     0 -Km/Lm 0 0 -Rm/Lm 0;
     0 0 0 KL/Ll 0 -(Rl+LR/2)/Ll;];
B = [0;
     0;
     0;
     0;
     2.88/L;
   0;];

Ce = [0 0 0 KT*R2/(R1+R2) 0 0];
D = [0];
Xi=[0;0;0;0;0;0];
EigenVal = eig(A)
state_space = ss(A,B,Ce,D);
transfer=tf(state_space)
t=0.001:0.01:5;
syms s
StateTransition=inv((s*eye(size(A))-A))
syms es
u=5*es/es;
Us=laplace(u)
Xh=ilaplace(StateTransition)*Xi;
Xhf=ilaplace(StateTransition*B*Us);
Xt=ilaplace(StateTransition*Xi+StateTransition*B*Us);
Vout=KT*R2/(R1+R2)*Xt(4)

subplot(311)
plot(t,subs(Km*Xt(5)),'k');hold on;
plot(t,subs(KL*Xt(6)),'b');
legend('T motor','T load');xlabel('time'); ylabel('Torque');grid on;
subplot(312)
plot(t,subs(Xt(4)),'k');hold on;
plot(t,subs(Xt(2)),'b');
legend('W load','W motor');xlabel('time'); ylabel('Angular velocity W');grid
on;
subplot(313)
plot(t,u,'p');hold on;
plot(t,subs(Vout));
legend('input','output');xlabel('time'); ylabel('Volt V');grid on
```

Code for GUI apps instrumental file responsible for data acquisition

```
import u3
import time
from time import sleep
from datetime import datetime
from numpy import *
from matplotlib.pyplot import *


def Experiment_Plot(t,y,u):

    figure()
    subplots_adjust(hspace=0.35)
    subplot(111)
    plot(t,u,'k',label='input'),title('Experimental Data')
    plot(t,y,'b',label='output'),xlabel('Time[s]'),ylabel('Volt [v])')
    legend()
    grid()
    return show()


def start_sampling(ST,t1,t2,t3,t4,t5,t6,t7,t8,t9,t10,val1,val2,val3,val4,val5,val6,val7,val8,val9,val10):

    Sti=ST;     ti1=t1;    ti2=t2;    ti3=t3;    ti4=t4;    ti5=t5;
    ti6=t6;    ti7=t7;    ti8=t8;    ti9=t9;    ti10=t10;
    vali1=val1;     vali2=val2  ;     vali3=val3  ;    vali4=val4 ;    vali5=val5   ;
    vali6=val6 ;    vali7=val7;    vali8=val8;    vali9=val9;    vali10=val10

 try:
        AIN=1;   uin=0;
        x=[];    y=[];    t_g=[];
        dead=1; test1=1+ti1; test2=test1+ti2;    test3=test2+ti3;
        test4=test3+ti4; test5=test4+ti5; test6=test5+ti6;
        test7=test6+ti7;    test8=test7+ti8;    test9=test8+ti9; t=test9+ti10;


        d=u3.U3()
        d.getCalibrationData()
        d.configU3()
        d.configIO(FIOAnalog=3)
        print "U3 is connected"
        start=time.time()

        sb1=d.voltageToDACBits(vali1,dacNumber = 0, is16Bits = False)
        sb2=d.voltageToDACBits(vali2,dacNumber = 0, is16Bits = False)
        sb3=d.voltageToDACBits(vali3,dacNumber = 0, is16Bits = False)
        sb4=d.voltageToDACBits(vali4,dacNumber = 0, is16Bits = False)
        sb5=d.voltageToDACBits(vali5,dacNumber = 0, is16Bits = False)
        sb6=d.voltageToDACBits(vali6,dacNumber = 0, is16Bits = False)
        sb7=d.voltageToDACBits(vali7,dacNumber = 0, is16Bits = False)
        sb8=d.voltageToDACBits(vali8,dacNumber = 0, is16Bits = False)
        sb9=d.voltageToDACBits(vali9,dacNumber = 0, is16Bits = False)
        sb10=d.voltageToDACBits(vali10,dacNumber = 0, is16Bits = False)
    print "started at:    ",datetime.now()

    while(t>= time.time()-start):
        i=datetime.now()
        print 't=',i.second+i.microsecond/1000000.
        if time.time()-start < dead:
            d.getFeedback(u3.DAC8(0,0))
        if dead<time.time()-start < test1:
            d.getFeedback(u3.DAC8(0,sb1))
        if test1<time.time()-start < test2:
            d.getFeedback(u3.DAC8(0,sb2))
        if test2 <time.time()-start < test3:
            d.getFeedback(u3.DAC8(0,sb3))
        if test3< time.time()-start < test4:
            d.getFeedback(u3.DAC8(0,sb4))
        if test4< time.time()-start < test5:
            d.getFeedback(u3.DAC8(0,sb5))
        if test5 <time.time()-start < test6:
            d.getFeedback(u3.DAC8(0,sb6))
        if test6< time.time()-start < test7:
            d.getFeedback(u3.DAC8(0,sb7))
        if test7< time.time()-start < test8:
            d.getFeedback(u3.DAC8(0,sb8))
        if test8< time.time()-start < test9:
            d.getFeedback(u3.DAC8(0,sb9))
        if test9< time.time()-start < t:
            d.getFeedback(u3.DAC8(0,sb10))


        a=d.getAIN(AIN); y.append(a)
        c=d.getAIN(uin); x.append(c)
        t_g.append(time.time()-start)
        print a

        end=time.time()-start
        if end>=(t-0.08) or end >=(t+0.2):
            print "Terminated at: ", datetime.now()
        f=datetime.now
        w=(f.second+f.microsecond/1000000.)-(i.second+i.microsecond/1000000.)
        dt=STi-w
        if dt<0.001:
            dt=0
        sleep(dt)

    savetxt('time.txt',t_g)
    savetxt('u.txt',x)
    savetxt('y.txt',y)
    print 'Data aquisition finished'
    return y, x, t_g
    d.getFeedback(u3.DAC8(0,0))
    d.close()


except:
    print "U3 device is not detected"
```

Code for GUI apps instrumental file responsible for Identification and Control

```
from numpy import savetxt,zeros,ones,eye,dot,mean,std
from numpy.random import randn
from numpy.linalg import inv
from matplotlib.pyplot import plot,figure,subplot,show,grid,xlabel,ylabel,title,subplots_adjust

def fn(x,r):
    if r==1:
        n=len(x)
    if r==2:
        n=(len(x)**2+len(x))/2
    if r==3:
        n=0
        for i in range(len(x)):
            for j in range(i,len(x)):
                for l in range(j,len(x)):
                    n=n+1
    return(n)

def fcolx(x,r):
    if r==1: # LNU
        colx=x
    if r==2:
        n=len(x)
        pom=0
        colx=zeros((n*n+n)/2)
        for i in range(n):
            for j in range(i,n):
                colx[pom]=x[i]*x[j]
                pom+=1
    if r==3:
        n=len(x)
        pom=0
        nw=0
        for i in range(n):
            for j in range(i,n):
                for l in range(j,n):
                    nw=nw+1

        colx=zeros(nw)
        for i in range(n):
            for j in range(i,n):
                for l in range(j,n):
                    colx[pom]=x[i]*x[j]*x[l]
                    pom+=1
    return(colx)


def ref_model(u,dT,tau,Td):

    d = u;    Td = Td;    dt = dT
    tau = tau;    rmNTd=int(Td/dt);   Su1=1
    yref=d.copy()
    d[1:]=d[0:-1]
    for k in range(rmNTd,len(d)-1):
        yref[k+1]=yref[k]+dt/tau*(Su1*d[k-rmNTd]-yref[k])
    return yref

def ref_model2(f,di,dT,tau,Td):

    d = di;    Td = Td;    dt = dT
    tau = tau;  rmNTd=int(Td/dt)
    Su1=1;    yref0=f; yref=d[-2]

    if yref0==-1:
        yref=d[-1]
    else:
        yref=yref0+dt/tau*(Su1*di[-2-rmNTd]-yref0)
    return yref

def IDENT_GD(u,y,yinit,mu,epochs,ny,nu,r):

    N=len(y)
    """
    Normalisation of data
    """
    meanu=mean(u);    stdu=std(u)
    meany=mean(y);    stdy=std(y)
    """
    #Initialisation of parameters#
    """
    ny=ny;    nd=nu;    mu=mu;    epochs=epochs

    N=len(y)
    nx=1+ny+nd;    x=ones(nx);    nw=fn(x,r)
    w=randn(nw)/nw;    colx=zeros(nw)
    yn=yinit
    e=zeros(N);   SSE=zeros(epochs);  dydw=zeros((N,nw))

    print 'starting plant identification'
    for epoch in range(epochs):
        for k in range(max(ny,nd),N):
            x[1:1+ny]=(yn[k-ny:k]-meany)/3/stdy
            x[1+ny:]=(u[k-nd:k]-meanu)/3/stdu
            colx=fcolx(x,r)
            yn[k]=dot(w,colx)*3*stdy+meany
            e[k]=y[k]-yn[k]
            dydw[k,:]=colx
            dw = mu/(1+dot(colx,colx))*e[k]*dydw[k,:]/3/stdy

            if epoch>0 and SSE[epoch]>=SSE[epoch-1]:
                w=w-dw*0.9
            else:
                w=w+dw

        SSE[epoch]=sum(e*e)
        print SSE[epoch]
    savetxt('w.txt',w)
    print 'identification finished'
    return yn, SSE, e

def IDENT_LM(u,y,yinit,mu,epochs,ny,nu,r):

    N=len(y)
    """
    Normalisation of data
    """
    meanu=mean(u);    stdu=std(u)
    meany=mean(y)    stdy=std(y)
    """
    Initialisation of parameters
    """
    ny=ny;    nd=nu;    mu=mu;    epochs=epochs

    N=len(y)
    nx=1+ny+nd;    x=ones(nx);    nw=fn(x,r);    w=randn(nw)/nw
    colx=zeros(nw);    yn=yinit
    e=zeros(N);   SSE=zeros(epochs);
    J=zeros((N,nw));    L=eye(nw)

    print 'starting plant identification'
    for epoch in range(epochs):
        for k in range(max(ny,nd),N):
            x[1:1+ny]=(yn[k-ny:k]-meany)/3/stdy
            x[1+ny:]=(u[k-nd:k]-meanu)/3/stdu
            colx=fcolx(x,r)
            yn[k]=dot(w,colx)*3*stdy+meany
            e[k]=y[k]-yn[k]
            J[k,:]=colx


        SSE[epoch]=sum(e*e)
        if epoch>0 and SSE[epoch]>=SSE[epoch-1]:
            dw=dot(dot(inv(dot(J.T,J)+1./mu*L),J.T),e/3/stdy)
            w=w-dw*0.9
        else:
            dw=dot(dot(inv(dot(J.T,J)+1./mu*L),J.T),e/3/stdy)
            w=w+dw
        print SSE[epoch]

    savetxt('w.txt',w)
    print 'identification finished'
    return yn, SSE, e
```

```
def CONT_GD(u,y,yinit,yref,di,w,muv,epochsc,ro,muro,ny,nu,nxiy,nxiu,rp,r):

    N=len(y);    meanu=mean(u);    stdu=std(u)
    meany=mean(y);    stdy=std(y)
    """
    Initialisation of parameters
    """
    muv=muv;    muro=muro;    ro=ro;    epochs=epochsc
    ny=ny;    nd=nu;    nw=len(w);    nx=1+ny+nd;    x=ones(nx)
    nxiy=nxiy;    nxid=nxiu;    nxi=1+nxiy+nxid;    xi=ones(nxi)
    nv=fn(xi,r);    v=randn(nv)/nv;    setpoint=di.copy();    dydv=zeros(nv)
    dxdv=zeros((nx,nv));    dcolxdv=zeros((nw,nv))
    eref=zeros(N);    SSE=zeros(epochs);    dxdro=zeros(nx)
    dcolxdro=zeros(nw);    dydro=0;    q=0;    yn=di.copy()

print 'starting controller tuning'

    for epoch in range(epochs):
        v0=v.copy()
        ro0=ro
        for k in range(max(ny,nd,nxiy,nxiu),N):
            #q-controller
            xi[1:1+nxiy]=(yn[k-nxiy:k]-meany)/3/stdy
            xi[1+nxiy:]=(di[k-nxid:k]-meanu)/3/stdu
            colxi=fcolx(xi,r)
            q=dot(v,colxi)*3*stdy+meany
            setpoint[k]=di[k]-q*ro #feedback
            #simulates real plant
            x[1:1+ny]=(yn[k-ny:k]-meany)/3/stdy
            x[1+ny:]=(setpoint[k-nd:k]-meanu)/3/stdu
            colx=fcolx(x,rp)
            yn[k]=dot(w,colx)*3*stdy+meany
            eref[k]=yref[k]-yn[k]
            #q-updates
            if rp==1:
                dydv=dot(w,dxdv)*3*stdy
                dxdv[1:ny,:]=dxdv[2:1+ny,:];dxdv[ny,:]=dydv/3/stdy
                dxdv[1+ny:-1,:]=dxdv[2+ny:,:];dxdv[-1,:]=-ro*stdy/stdu*colxi
                dydro=dot(w,dxdro)*3*stdy
                dxdro[1:ny]=dxdro[2:1+ny];dxdro[ny]=dydro/3/stdy
                dxdro[1+ny:-1]=dxdro[2+ny:];dxdro[-1]=-q/3/stdu
                dv=muv*eref[k]*dydv/3/stdy
                v=v+dv
                dro=muro*eref[k]*dydro/3/stdy
                ro=ro+dro

            if rp==2:
                pom=0
                for i in range(nx):
                    for j in range(i,nx):
                        dcolxdv[pom]=dxdv[i,:]*x[j]+x[i]*dxdv[j,:]
                        dcolxdro[pom]=dxdro[i]*x[j]+x[i]*dxdro[j]
                        pom+=1
                dydv=dot(w,dcolxdv)*3*stdy
                dv=muv*eref[k]*dydv/3/stdy
                v=v+dv
                dxdv[1:ny,:]=dxdv[2:1+ny,:];dxdv[ny,:]=dydv/3/stdy
                dxdv[1+ny:-1,:]=dxdv[2+ny:,:];dxdv[-1,:]=-ro*stdy/stdu*colxi
                dydro=dot(w,dcolxdro)*3*stdy
                dro=muro*eref[k]*dydro/3/stdy
                ro=ro+dro
                dxdro[1:ny]=dxdro[2:1+ny];dxdro[ny]=dydro/3/stdy
                dxdro[1+ny:-1]=dxdro[2+ny:];dxdro[-1]=-q/3/stdu

            if rp==3:
                pom=0
                for i in range(nx):
                    for j in range(i,nx):
                        for k in range(j,nx):
                            dcolxdv[pom]=dxdv[i,:]*x[j]*x[k] + x[i]*dxdv[j,:]*x[k] + x[i]*x[j]*dxdv[k,:]
                            dcolxdro[pom]=dxdro[i]*x[j]*x[k] + x[i]*dxdro[j]*x[k] + x[i]*x[j]*dxdro[k]
                            pom+=1

                dydv=dot(w,dcolxdv)*3*stdy
                dv=muv*eref[k]*dydv/3/stdy
                v=v+dv
                dxdv[1:ny,:]=dxdv[2:1+ny,:];dxdv[ny,:]=dydv/3/stdy
                dxdv[1+ny:-1,:]=dxdv[2+ny:,:];dxdv[-1,:]=-ro*stdy/stdu*colxi
                dydro=dot(w,dcolxdro)*3*stdy
                dro=muro*eref[k]*dydro/3/stdy
                ro=ro+dro
                dxdro[1:ny]=dxdro[2:1+ny];dxdro[ny]=dydro/3/stdy
                dxdro[1+ny:-1]=dxdro[2+ny:];dxdro[-1]=-q/3/stdu

        SSE[epoch]=sum(eref*eref)
        if epoch>0 and SSE[epoch]>=SSE[epoch-1]:
            dv=v-v0;dro=ro-ro0
            v=v-dv*0.999
            ro=ro-dro*0.999
            muv=muv*0.9
            muro=muro*0.9
            print 'reduced'

        print(SSE[epoch])

    vectorval=[ro,meany,stdy,meanu,stdu,muv,muro,ny,nu,dydro]
    savetxt('dydv.txt',dydv)
    savetxt('dxdv.txt',dxdv)
    savetxt('dcolxdv.txt',dcolxdv)
    savetxt('dxdro.txt',dxdro)
    savetxt('dcolxdro.txt',dcolxdro)
    savetxt('vectorval.txt',vectorval)
    savetxt('v.txt',v)
    print ro
    print 'controller tuning finished'

    return yn, SSE, eref
```

```
def CONT_LM(u,y,yinit,yref,di,w,muv,epochsc,ro,muro,ny,nu,nxiy,nxiu,rp,r):

    N=len(y);    meanu=mean(u);    stdu=std(u)
    meany=mean(y);    stdy=std(y)
    """
    Initialisation of parameters
    """
    muv=muv;    muro=muro;    ro=ro;    epochs=epochsc
    ny=ny;    nd=nu;    nw=len(w);    nx=1+ny+nd;    x=ones(nx)
    nxiy=nxiy;    nxid=nxiu;    nxi=1+nxiy+nxid;    xi=ones(nxi)
    nv=fn(xi,r);    v=randn(nv)/nv;    setpoint=di.copy();    dydv=zeros(nv)
    dxdv=zeros((nx,nv));    dcolxdv=zeros((nw,nv))
    eref=zeros(N);    SSE=zeros(epochs);    dxdro=zeros(nx)
    dcolxdro=zeros(nw);    dydro=0;
    Jv=zeros((N,nv));    Jro=zeros(N);    q=0;    yn=di.copy()

    print 'starting controller tuning'
    for epoch in range(epochs):
        for k in range(max(ny,nd,nxiy,nxiu),N):

            xi[1:1+nxiy]=(yn[k-nxiy:k]-meany)/3/stdy
            xi[1+nxiy:]=(di[k-nxid:k]-meanu)/3/stdu
            colxi=fcolx(xi,r)
            q=dot(v,colxi)*3*stdy+meany

            setpoint[k]=di[k]-q*ro
            x[1:1+ny]=(yn[k-ny:k]-meany)/3/stdy
            x[1+ny:]=(setpoint[k-nd:k]-meanu)/3/stdu

            colx=fcolx(x,rp)
            yn[k]=dot(w,colx)*3*stdy+meany
            eref[k]=yref[k]-yn[k]

            if rp==1:
                dydv=dot(w,dxdv)*3*stdy
                Jv[k,:]=dydv
                dxdv[1:ny,:]=dxdv[2:1+ny,:];dxdv[ny,:]=dydv/3/stdy
                dxdv[1+ny:-1,:]=dxdv[2+ny:,:];dxdv[-1,:]=-ro*stdy/stdu*colxi
                dydro=dot(w,dxdro)*3*stdy
                Jro[k]=dydro
                dxdro[1:ny]=dxdro[2:1+ny];dxdro[ny]=dydro/3/stdy
                dxdro[1+ny:-1]=dxdro[2+ny:];dxdro[-1]=-q/3/stdu
            if rp==2:
                pom=0
                for i in range(nx):
                    for j in range(i,nx):
                        dcolxdv[pom]=dxdv[i,:]*x[j]+x[i]*dxdv[j,:]
                        dcolxdro[pom]=dxdro[i]*x[j]+x[i]*dxdro[j]
                        pom+=1
            dydv=dot(w,dcolxdv)*3*stdy
            Jv[k,:]=dydv
            dxdv[1:ny,:]=dxdv[2:1+ny,:];dxdv[ny,:]=dydv/3/stdy
            dxdv[1+ny:-1,:]=dxdv[2+ny:,:];dxdv[-1,:]=-ro*stdy/stdu*colxi
            dydro=dot(w,dcolxdro)*3*stdy
            Jro[k]=dydro
            dxdro[1:ny]=dxdro[2:1+ny];dxdro[ny]=dydro/3/stdy
            dxdro[1+ny:-1]=dxdro[2+ny:];dxdro[-1]=-q/3/stdu

        if rp==3:
            pom=0
            for i in range(nx):
                for j in range(i,nx):
                    for k in range(j,nx):
                        dcolxdv[pom]=dxdv[i,:]*x[j]*x[k] + x[i]*dxdv[j,:]*x[k] + x[i]*x[j]*dxdv[k,:]
                        dcolxdro[pom]=dxdro[i]*x[j]*x[k] + x[i]*dxdro[j]*x[k] + x[i]*x[j]*dxdro[k]
                        pom+=1

            dydv=dot(w,dcolxdv)*3*stdy
            Jv[k,:]=dydv
            dxdv[1:ny,:]=dxdv[2:1+ny,:];dxdv[ny,:]=dydv/3/stdy
            dxdv[1+ny:-1,:]=dxdv[2+ny:,:];dxdv[-1,:]=-ro*stdy/stdu*colxi
            dydro=dot(w,dcolxdro)*3*stdy
            Jro[k]=dydro
            dxdro[1:ny]=dxdro[2:1+ny];dxdro[ny]=dydro/3/stdy
            dxdro[1+ny:-1]=dxdro[2+ny:];dxdro[-1]=-q/3/stdu

    SSE[epoch]=sum(eref*eref)
    if epoch>10 and SSE[epoch]>=SSE[epoch-1]:
        dv=dot(dot(inv(dot(Jv.T,Jv)+1./muv*eye(nv)),Jv.T),eref/3/stdy)
        v=v-dv*0.9
        dro=sum(Jro*eref/3/stdy)/(sum(Jro*Jro)+1./muro)
        ro=ro-dro*0.9
        print 'reduced'
    else:
        dv=dot(dot(inv(dot(Jv.T,Jv)+1./muv*eye(nv)),Jv.T),eref/3/stdy)
        v=v+dv
        dro=sum(Jro*eref/3/stdy)/(sum(Jro*Jro)+1./muro)
        ro=ro+dro
    print(SSE[epoch])

vectorval=[ro,meany,stdy,meanu,stdu,muv,muro,ny,nu,dydro]
savetxt('dydv.txt',dydv)
savetxt('dxdv.txt',dxdv)
savetxt('dcolxdv.txt',dcolxdv)
savetxt('dxdro.txt',dxdro)
savetxt('dcolxdro.txt',dcolxdro)
savetxt('vectorval.txt',vectorval)
savetxt('v.txt',v)
print ro
print 'controller tuning finished'

return yn, SSE, eref
```

```
def Ident_Plot(y,yn,SSE):

    figure()
    subplots_adjust(hspace=0.35)
    subplot(211)
    plot(y),title('HONU Plant Identification')
    plot(yn,'g'),xlabel('Samples[k]'),ylabel('Plant Output [-]')
    grid()
    subplot(212)
    plot(SSE,'k'),xlabel('Epochs'),ylabel('SSE')
    grid()
    return show()

def Control_Plot(y,yn,yref,d,SSE):

    figure()
    subplots_adjust(hspace=0.35)
    subplot(211),title('Controller Adaptive Tuning')
    plot(d,'--k')
    plot(yref,'--r')
#    plot(y,'b')
    plot(yn,'m'),xlabel('Samples[k]'),ylabel('Plant Output [-]')
    grid()
    subplot(212)
    plot(SSE,'k'),xlabel('Epochs'),ylabel('SSE')
    grid()
    return show()
```

Code for Main file of GUI application

```
import u3
import time
import os
from io import StringIO
from time import sleep
from datetime import datetime
from numpy import *
from matplotlib.backends.backend_wxagg import FigureCanvasWxAgg as FigureCanvas,
NavigationToolbar2WxAgg as NavigationToolbar
import wx
import wx.lib.scrolledpanel
from Control import *
from Getting_data import *
from matplotlib.pyplot import plot,figure,subplot,show,grid,xlabel,title,subplots_adjust,ion,close

origWD = os.getcwd()

class console(wx.Frame):
    def __init__(self,parent,id,title):
        #----
        self.n_first_out=0
        #----
        wx.Frame.__init__(self,parent,id,title,size=(1130,490))

        panel1 = wx.lib.scrolledpanel.ScrolledPanel(self,-
1,pos=(0,0),size=(1130,490),style=wx.SIMPLE_BORDER)
        panel1.SetBackgroundColour("White")
        panel1.SetupScrolling()
        panel1.GetEffectiveMinSize()
        Image10 = wx.Image('title.jpg', wx.BITMAP_TYPE_ANY).ConvertToBitmap()
        wx.StaticBitmap(panel1, -1, Image10, pos=(0,0), size=(1130,490))

        box = wx.BoxSizer(wx.VERTICAL)
        box.Add(panel1, 1, wx.EXPAND)

        self.SetAutoLayout(True)
        self.SetSizer(box)
        self.Layout()

        menuBar = wx.MenuBar()
        GetDataMenu=wx.Menu()
        IdentificationMenu = wx.Menu()
        TuningMenu = wx.Menu()
        ImplementMenu = wx.Menu()
        SelfControllingMenu=wx.Menu()

        menuBar.Append(GetDataMenu,"&Get Data")
        menuBar.Append(IdentificationMenu,"&Identification")
        menuBar.Append(TuningMenu, "&Tuning")
        menuBar.Append(ImplementMenu, "&Implement")
        menuBar.Append(SelfControllingMenu, "&OnlineTune")

        MenuItem1 = IdentificationMenu.Append(101,"Plant adaptive Identification","Open the
application",wx.ITEM_NORMAL)
        MenuItem2 = TuningMenu.Append(201,"Plant adaptive Control","Open the
Application",wx.ITEM_NORMAL)
        MenuItem3 = ImplementMenu.Append(401,"Execution of controled plant","Open the
Application",wx.ITEM_NORMAL)
        MenuItem4 = GetDataMenu.Append(301,"Obtain data through experiment","Open the
application",wx.ITEM_NORMAL)
        MenuItem5 = SelfControllingMenu.Append(501,"Online controller update","Open the
application",wx.ITEM_NORMAL)

        self.Bind(wx.EVT_MENU, self.Identify, MenuItem1)
        self.Bind(wx.EVT_MENU, self.Tune, MenuItem2)
        self.Bind(wx.EVT_MENU, self.Execute, MenuItem3)
        self.Bind(wx.EVT_MENU, self.GetData, MenuItem4)
        self.Bind(wx.EVT_MENU, self.OnlineUpdate, MenuItem5)
        self.SetMenuBar(menuBar)
        self.Centre()

    def Identify(self,event):
        self.DestroyChildren()

        if os.getcwd() == origWD:
            panel1 = wx.lib.scrolledpanel.ScrolledPanel(self,-
1,pos=(0,0),size=(1130,490),style=wx.SIMPLE_BORDER)
            panel1.SetupScrolling()
            panel1.GetEffectiveMinSize()

            panel12 = wx.Panel(panel1, -1, pos=(10,0),size=(1100,230))
            panel12.SetBackgroundColour("White")
            Title1 = wx.StaticText(panel12, label='Plant Setup', pos=(5, 5))
            button1 = wx.Button(panel12,label="Plot Data",pos=(550,70),size=(150,40))
            Title2 = wx.StaticText(panel12, label='Resampling (every n sample)=', pos=(15,170))
            self.text1 = wx.TextCtrl(panel12, -1, pos=(185,165),size=(50,25),value='1')
            Title2b = wx.StaticText(panel12, label='Feedback Gain (ro)=', pos=(250,170))
            self.text1b = wx.TextCtrl(panel12, -1, pos=(360,165),size=(50,25),value='1.0')
            self.cb0 = wx.CheckBox(panel12, -1, 'y0=yreal, u0=ureal', (440, 170))
            self.cb0.SetValue(True)
            Title1c = wx.StaticText(panel12, label='Reference Model Setup', pos=(740, 10))
            Image0 = wx.Image('Plant.jpg', wx.BITMAP_TYPE_ANY).ConvertToBitmap()
            wx.StaticBitmap(panel12, -1, Image0, pos=(100,15), size=(350,150))
            Image1 = wx.Image('ref.jpg', wx.BITMAP_TYPE_ANY).ConvertToBitmap()
            wx.StaticBitmap(panel12, -1, Image1, pos=(740,30), size=(350,130))
            line0 = wx.StaticLine(panel12, -1, pos=(725,15), size=(3,200))
            Title2c = wx.StaticText(panel12, label='Td=', pos=(750, 170))
            Title2d = wx.StaticText(panel12, label='tau=', pos=(850, 170))
            Title2e = wx.StaticText(panel12, label='Sampling=', pos=(950, 170))
            self.text2c = wx.TextCtrl(panel12, -1, pos=(775, 165),size=(40,25),value='0')
            self.text2d = wx.TextCtrl(panel12, -1, pos=(880, 165),size=(40,25),value='0.15')
            self.text2e = wx.TextCtrl(panel12, -1, pos=(1035, 165),size=(40,25),value='0.1')
            self.Bind(wx.EVT_BUTTON, self.plot_data, button1)
            self.Bind(wx.EVT_CLOSE, self.closewindow)

            panel13 = wx.Panel(panel1, -1, pos=(10,235),size=(1100,200))
            panel13.SetBackgroundColour("White")
            Title6 = wx.StaticText(panel13,label='Plant Identification', pos=(5, 5))
            Title7 = wx.StaticText(panel13, label='ny=', pos=(25, 90))
            Title8 = wx.StaticText(panel13, label='nu=', pos=(90, 90))
            self.text5 = wx.TextCtrl(panel13, -1, pos=(50, 85),size=(30,25),value='4')
            self.text6 = wx.TextCtrl(panel13, -1, pos=(115, 85),size=(30,25),value='7')
            self.cb1 = wx.CheckBox(panel13, -1, 'LNU', (220, 40))
            self.cb2 = wx.CheckBox(panel13, -1, 'QNU', (340, 40))
            self.cb2.SetValue(True)
            self.cb3 = wx.CheckBox(panel13, -1, 'CNU', (460, 40))

            Title14 = wx.StaticText(panel13, label='muGD=', pos=(170, 120))
            Title15 = wx.StaticText(panel13, label='epochsGD=', pos=(270, 120))
            Title16 = wx.StaticText(panel13, label='muBPTT=', pos=(165, 160))
            Title17 = wx.StaticText(panel13, label='epochsBPTT=', pos=(265, 160))
            self.text9 = wx.TextCtrl(panel13, -1, pos=(220, 155),size=(40,25),value='0.1')
            self.text10 = wx.TextCtrl(panel13, -1, pos=(340, 155),size=(35,25),value='100')
            line1 = wx.StaticLine(panel13, -1, pos=(160,95), size=(580,3))

            self.text14 = wx.TextCtrl(panel13, -1, pos=(220, 115),size=(35,25),value='0.1')
            self.text15 = wx.TextCtrl(panel13, -1, pos=(340, 115),size=(35,25),value='1000')
            button2 = wx.Button(panel13,label="Train with GD",pos=(390,120),size=(170,40))
            button3 = wx.Button(panel13,label="via. BPTT",pos=(570,120),size=(170,40))
            Image2 = wx.Image('Plant_Id.jpg', wx.BITMAP_TYPE_ANY).ConvertToBitmap()
            wx.StaticBitmap(panel13, -1, Image2, pos=(770,15), size=(300,150))
            self.Bind(wx.EVT_BUTTON, self.GD, button2)
            self.Bind(wx.EVT_BUTTON, self.LM, button3)

        else:
            os.chdir(str(origWD))

            panel1 = wx.lib.scrolledpanel.ScrolledPanel(self,-
1,pos=(0,0),size=(1130,490),style=wx.SIMPLE_BORDER)
            panel1.SetupScrolling()
            panel1.GetEffectiveMinSize()

            panel12 = wx.Panel(panel1, -1, pos=(10,0),size=(1100,230))
            panel12.SetBackgroundColour("White")
            Title1 = wx.StaticText(panel12, label='Plant Setup', pos=(5, 5))
            button1 = wx.Button(panel12,label="Plot Data",pos=(550,70),size=(150,40))
            Title2 = wx.StaticText(panel12, label='Resampling(Every n Sample)=', pos=(15,170))
            self.text1 = wx.TextCtrl(panel12, -1, pos=(185,165),size=(50,25),value='1')
            Title2b = wx.StaticText(panel12, label='Feedback Gain (ro)=', pos=(250,170))
            self.text1b = wx.TextCtrl(panel12, -1, pos=(360,165),size=(50,25),value='1.0')
            self.cb0 = wx.CheckBox(panel12, -1, 'y0=yreal, u0=ureal', (440, 170))
            self.cb0.SetValue(True)
            Title1c = wx.StaticText(panel12, label='Reference Model Setup', pos=(740, 10))
            Image0 = wx.Image('Plant.jpg', wx.BITMAP_TYPE_ANY).ConvertToBitmap()
            wx.StaticBitmap(panel12, -1, Image0, pos=(100,15), size=(350,150))
            Image1 = wx.Image('ref.jpg', wx.BITMAP_TYPE_ANY).ConvertToBitmap()
            wx.StaticBitmap(panel12, -1, Image1, pos=(740,30), size=(350,130))
            line0 = wx.StaticLine(panel12, -1, pos=(725,15), size=(3,200))
            Title2c = wx.StaticText(panel12, label='Td=', pos=(750, 170))
            Title2d = wx.StaticText(panel12, label='tau=', pos=(850, 170))
            Title2e = wx.StaticText(panel12, label='Sampling=', pos=(950, 170))
            self.text2c = wx.TextCtrl(panel12, -1, pos=(775, 165),size=(40,25),value='0')
            self.text2d = wx.TextCtrl(panel12, -1, pos=(880, 165),size=(40,25),value='0.15')
            self.text2e = wx.TextCtrl(panel12, -1, pos=(1035, 165),size=(40,25),value='0.1')
            self.Bind(wx.EVT_BUTTON, self.plot_data, button1)
            self.Bind(wx.EVT_CLOSE, self.closewindow)
            panel13 = wx.Panel(panel1, -1, pos=(10,235),size=(1100,200))
            panel13.SetBackgroundColour("White")

            Title6 = wx.StaticText(panel13,label='Plant Identification', pos=(5, 5))
            Title7 = wx.StaticText(panel13, label='ny=', pos=(25, 90))
            Title8 = wx.StaticText(panel13, label='nu=', pos=(90, 90))
            self.text5 = wx.TextCtrl(panel13, -1, pos=(50, 85),size=(30,25),value='4')
            self.text6 = wx.TextCtrl(panel13, -1, pos=(115, 85),size=(30,25),value='7')
            self.cb1 = wx.CheckBox(panel13, -1, 'LNU', (220, 40))
            self.cb2 = wx.CheckBox(panel13, -1, 'QNU', (340, 40))
            self.cb2.SetValue(True)
            self.cb3 = wx.CheckBox(panel13, -1, 'CNU', (460, 40))

            Title14 = wx.StaticText(panel13, label='muGD=', pos=(170, 120))
            Title15 = wx.StaticText(panel13, label='epochsGD=', pos=(270, 120))
            Title16 = wx.StaticText(panel13, label='muBPTT=', pos=(165, 160))
            Title17 = wx.StaticText(panel13, label='epochsBPTT=', pos=(265, 160))
            self.text9 = wx.TextCtrl(panel13, -1, pos=(220, 155),size=(40,25),value='0.1')
            self.text10 = wx.TextCtrl(panel13, -1, pos=(340, 155),size=(35,25),value='100')
            line1 = wx.StaticLine(panel13, -1, pos=(160,95), size=(580,3))

            self.text14 = wx.TextCtrl(panel13, -1, pos=(220, 115),size=(35,25),value='0.1')
            self.text15 = wx.TextCtrl(panel13, -1, pos=(340, 115),size=(35,25),value='1000')
            button2 = wx.Button(panel13,label="Train with GD",pos=(390,120),size=(170,40))
            button3 = wx.Button(panel13,label="via. BPTT",pos=(570,120),size=(170,40))
            Image2 = wx.Image('Plant_Id.jpg', wx.BITMAP_TYPE_ANY).ConvertToBitmap()
            wx.StaticBitmap(panel13, -1, Image2, pos=(770,15), size=(300,150))
            self.Bind(wx.EVT_BUTTON, self.GD, button2)
            self.Bind(wx.EVT_BUTTON, self.LM, button3)


        def plot_data(self, event):
        os.chdir(str(origWD + '/Data_files'))

        u=loadtxt('u.txt')[self.n_first_out:]
        yrp=loadtxt('y.txt')[self.n_first_out:]
        d=loadtxt('d.txt')[self.n_first_out:]
        N=len(yrp)

        dn=int(self.text1.GetValue())
        yrp=yrp[range(0,N,dn)]
        u=u[range(0,N,dn)]
        d=d[range(0,N,dn)]
        N=len(yrp)

        dT = float(self.text2e.GetValue())
        tau = float(self.text2d.GetValue())
        Td = float(self.text2c.GetValue())
        yref = ref_model(d,dT,tau,Td)
        figure()
        subplots_adjust(hspace=0.65)
        subplot(311)
        plot(u,'g'),title('u...Control Input (measured data)'),xlabel('k'),grid()
        subplot(312)
        plot(yrp),title('y...Controlled Output (measured data)'),xlabel('k'),grid()
        subplot(313)
        plot(yref,'--r')
        plot(d,'--k'),title('d...Desired Variable, yref...Reference Model'),xlabel('k'),grid()
        show()

        os.chdir(str(origWD))

        def GD(self, event):
        os.chdir(str(origWD + '/Data_files'))

        u=loadtxt('u.txt')[self.n_first_out:]
        y=loadtxt('y.txt')[self.n_first_out:]
        N=len(y)

        dn=int(self.text1.GetValue())
        y=y[range(0,N,dn)]
        u=u[range(0,N,dn)]
        N=len(y)

        r=0
        if self.cb0.GetValue() == True:
            yinit = y.copy()
        else:
            yinit = zeros(N)

        if self.cb1.GetValue() == True:
            r=1
        if self.cb2.GetValue() == True:
            r=2
        if self.cb3.GetValue() == True:
            r=3

        mu=float(self.text14.GetValue())
        epochs=int(self.text15.GetValue())
        ny=int(self.text5.GetValue())
        nu=int(self.text6.GetValue())
        yn, SSE, e = IDENT_GD(u,y,yinit,mu,epochs,ny,nu,r)
        Ident_Plot(y,yn,SSE)
        os.chdir(str(origWD))

        def LM(self, event):
        os.chdir(str(origWD + '/Data_files'))

        u=loadtxt('u.txt')[self.n_first_out:]
        y=loadtxt('y.txt')[self.n_first_out:]
        N=len(y)

        dn=int(self.text1.GetValue())
        y=y[range(0,N,dn)]
        u=u[range(0,N,dn)]
        N=len(y)
        r=0
        if self.cb0.GetValue() == True:
            yinit = y.copy()
        else:
            yinit = zeros(N)
        if self.cb1.GetValue() == True:
            r=1
        if self.cb2.GetValue() == True:
            r=2
        if self.cb3.GetValue() == True:
            r=3

        mu=float(self.text9.GetValue())
        epochs=int(self.text10.GetValue())
        ny=int(self.text5.GetValue())
        nu=int(self.text6.GetValue())
        yn, SSE, e = IDENT_LM(u,y,yinit,mu,epochs,ny,nu,r)

        Ident_Plot(y,yn,SSE)

        os.chdir(str(origWD))

    def Tune_GD(self, event):
        os.chdir(str(origWD + '/Data_files'))

        u=loadtxt('u.txt')[self.n_first_out:]
        y=loadtxt('y.txt')[self.n_first_out:]
        di=loadtxt('d.txt')[self.n_first_out:]
        N=len(y)

        dn=int(self.text1.GetValue())
        y=y[range(0,N,dn)]
        u=u[range(0,N,dn)]
        di=di[range(0,N,dn)]
        N=len(y)

        dT = float(self.text2e.GetValue())
        tau = float(self.text2d.GetValue())
        Td = float(self.text2c.GetValue())
        yref = ref_model(di,dT,tau,Td)

        rp=0
        if self.cb0.GetValue() == True:
            yinit = y.copy()
        else:
            yinit = zeros(N)

        if self.cb7.GetValue() == True:
            rp=1
        if self.cb8.GetValue() == True:
            rp=2
        if self.cb9.GetValue() == True:
            rp=3
        r=0
        if self.cb4.GetValue() == True:
            r=1
        if self.cb5.GetValue() == True:
            r=2
        if self.cb6.GetValue() == True:
            r=3

        print rp, r

        ny=int(self.text21f.GetValue())
        nu=int(self.text21g.GetValue())
        w=loadtxt('w.txt')
        epochsc=int(self.text26.GetValue())
        muv=float(self.text25.GetValue())
        muro = float(self.text22c.GetValue())
        nxiy=int(self.text22.GetValue())
        nxiu=int(self.text22b.GetValue())
        ro=float(self.text1b.GetValue())
        yn, SSE, eref = CONT_GD(u,y,yinit,yref,di,w,muv,epochsc,ro,muro,ny,nu,nxiy,nxiu,rp,r)

        Control_Plot(y,yn,yref,di,SSE)

        os.chdir(str(origWD))

    def Tune_LM(self, event):
        os.chdir(str(origWD + '/Data_files'))

        u=loadtxt('u.txt')[self.n_first_out:]
        y=loadtxt('y.txt')[self.n_first_out:]
        di=loadtxt('d.txt')[self.n_first_out:]
        N=len(y)

        dn=int(self.text1.GetValue())
        y=y[range(0,N,dn)]
        u=u[range(0,N,dn)]
        di=di[range(0,N,dn)]
        N=len(y)

        dT = float(self.text2e.GetValue())
        tau = float(self.text2d.GetValue())
        Td = float(self.text2c.GetValue())
        yref = ref_model(di,dT,tau,Td)

        rp=0
        if self.cb0.GetValue() == True:
            yinit = y.copy()
        else:
            yinit = zeros(N)

        if self.cb7.GetValue() == True:
            rp=1
        if self.cb8.GetValue() == True:
            rp=2
        if self.cb9.GetValue() == True:
            rp=3

        r=0
        if self.cb4.GetValue() == True:
            r=1
        if self.cb5.GetValue() == True:
            r=2
        if self.cb6.GetValue() == True:
            r=3

        print rp, r

        ny=int(self.text21f.GetValue())
        nu=int(self.text21g.GetValue())
        w=loadtxt('w.txt')

        epochsc=int(self.text19b.GetValue())
        muv=float(self.text20.GetValue())
        muro = float(self.text22c.GetValue())
        nxiy=int(self.text22.GetValue())
        nxiu=int(self.text22b.GetValue())
        ro=float(self.text1b.GetValue())
        yn, SSE, eref = CONT_LM(u,y,yinit,yref,di,w,muv,epochsc,ro,muro,ny,nu,nxiy,nxiu,rp,r)

        Control_Plot(y,yn,yref,di,SSE)

        os.chdir(str(origWD))

    def closewindow(self, event):
        self.Destroy()

    def Tune(self,event):
        self.DestroyChildren()

        if os.getcwd() == origWD:
            panel1 = wx.lib.scrolledpanel.ScrolledPanel(self,-
1,pos=(0,0),size=(1130,490),style=wx.SIMPLE_BORDER)
            panel1.SetupScrolling()
            panel1.GetEffectiveMinSize()

            panel12 = wx.Panel(panel1, -1, pos=(10,0),size=(1100,230))
            panel12.SetBackgroundColour("White")
            Title1 = wx.StaticText(panel12, label='Plant Setup', pos=(5, 5))
            button1 = wx.Button(panel12,label="Plot Data",pos=(550,70),size=(150,40))
            Title2 = wx.StaticText(panel12, label='Resampling(Every n sample)=', pos=(15,170))
            self.text1 = wx.TextCtrl(panel12, -1, pos=(185,165),size=(50,25),value='1')
            Title2b = wx.StaticText(panel12, label='Feedback Gain (ro)=', pos=(250,170))
            self.text1b = wx.TextCtrl(panel12, -1, pos=(360,165),size=(50,25),value='1.0')
            self.cb0 = wx.CheckBox(panel12, -1, 'y0=yreal, u0=ureal', (440, 170))
            self.cb0.SetValue(True)
            Title1c = wx.StaticText(panel12, label='Reference Model Setup', pos=(740, 10))
            Image0 = wx.Image('Plant.jpg', wx.BITMAP_TYPE_ANY).ConvertToBitmap()
            wx.StaticBitmap(panel12, -1, Image0, pos=(100,15), size=(350,150))
            Image1 = wx.Image('ref.jpg', wx.BITMAP_TYPE_ANY).ConvertToBitmap()
            wx.StaticBitmap(panel12, -1, Image1, pos=(740,30), size=(350,130))
            line0 = wx.StaticLine(panel12, -1, pos=(725,15), size=(3,200))
            Title2c = wx.StaticText(panel12, label='Td=', pos=(750, 170))
            Title2d = wx.StaticText(panel12, label='tau=', pos=(850, 170))
            Title2e = wx.StaticText(panel12, label='Sampling=', pos=(950, 170))
            self.text2c = wx.TextCtrl(panel12, -1, pos=(775, 165),size=(40,25),value='0')
            self.text2d = wx.TextCtrl(panel12, -1, pos=(880, 165),size=(40,25),value='0.15')
            self.text2e = wx.TextCtrl(panel12, -1, pos=(1035, 165),size=(40,25),value='0.1')
            self.Bind(wx.EVT_BUTTON, self.plot_data, button1)
            self.Bind(wx.EVT_CLOSE, self.closewindow)

            panel14 = wx.Panel(panel1, -1, pos=(10,240),size=(1100,220))
            panel14.SetBackgroundColour("White")
            Title18 = wx.StaticText(panel14,label='State Feedback Neuro-controller', pos=(5, 5))
            Title21 = wx.StaticText(panel14, label='nqy=', pos=(580, 65))
            Title21b = wx.StaticText(panel14, label='nqd=', pos=(645, 65))
            Title21c = wx.StaticText(panel14, label='muro=', pos=(25, 120))
            Title21d = wx.StaticText(panel14, label='System :', pos=(170, 30))
            Title21e = wx.StaticText(panel14, label='Controller:', pos=(170, 60))
            Title21f = wx.StaticText(panel14, label='ny=', pos=(580, 30))
            Title21g = wx.StaticText(panel14, label='nu=', pos=(645, 30))
            self.cb4 = wx.CheckBox(panel14, -1, 'LNU', (260, 60))
            self.cb5 = wx.CheckBox(panel14, -1, 'QNU', (380, 60))
            self.cb5.SetValue(True)
            self.cb6 = wx.CheckBox(panel14, -1, 'CNU', (500, 60))
            self.cb7 = wx.CheckBox(panel14, -1, 'LNU', (260, 30))
            self.cb8 = wx.CheckBox(panel14, -1, 'QNU', (380, 30))
            self.cb8.SetValue(True)
            self.cb9 = wx.CheckBox(panel14, -1, 'CNU', (500, 30))

            self.text19b = wx.TextCtrl(panel14, -1, pos=(345, 155),size=(35,25),value='100')
            Title26 = wx.StaticText(panel14, label='muGD=', pos=(170, 120))
            Title27 = wx.StaticText(panel14, label='epochsGD=', pos=(270, 120))
            Title28 = wx.StaticText(panel14, label='muBPTT=', pos=(170, 160))
            Title29 = wx.StaticText(panel14, label='epochsBPTT=', pos=(270, 160))
            self.text20 = wx.TextCtrl(panel14, -1, pos=(225, 155),size=(40,25),value='0.01')
            line2 = wx.StaticLine(panel14, -1, pos=(160,95), size=(580,3))
            self.text21f = wx.TextCtrl(panel14, -1, pos=(610, 25),size=(30,25),value='4')
            self.text21g = wx.TextCtrl(panel14, -1, pos=(675, 25),size=(30,25),value='7')
            self.text22 = wx.TextCtrl(panel14, -1, pos=(610, 60),size=(30,25),value='4')
            self.text22b = wx.TextCtrl(panel14, -1, pos=(675, 60),size=(30,25),value='7')
            self.text22c = wx.TextCtrl(panel14, -1, pos=(65, 115),size=(40,25),value='0.01')
            self.text25 = wx.TextCtrl(panel14, -1, pos=(220, 115),size=(40,25),value='0.001')
            self.text26 = wx.TextCtrl(panel14, -1, pos=(340, 115),size=(40,25),value='1000')
            button6 = wx.Button(panel14,label="Train with GD",pos=(390,120),size=(170,40))
            button6b = wx.Button(panel14,label="via. BPTT",pos=(570,120),size=(170,40))
            Image3 = wx.Image('Control.jpg', wx.BITMAP_TYPE_ANY).ConvertToBitmap()
            wx.StaticBitmap(panel14, -1, Image3, pos=(760,15), size=(340,150))
            self.Bind(wx.EVT_BUTTON, self.Tune_GD, button6)
            self.Bind(wx.EVT_BUTTON, self.Tune_LM, button6b)
        else:
            os.chdir(str(origWD))

            panel1 = wx.lib.scrolledpanel.ScrolledPanel(self,-
1,pos=(0,0),size=(1130,490),style=wx.SIMPLE_BORDER)
```

```
panel1.SetupScrolling()
panel1.GetEffectiveMinSize()

panel12 = wx.Panel(panel1, -1, pos=(10,0),size=(1100,230))
panel12.SetBackgroundColour("White")
Title1 = wx.StaticText(panel12, label='Plant Setup', pos=(5, 5))
button1 = wx.Button(panel12,label="Plot Data",pos=(550,70),size=(150,40))
Title2 = wx.StaticText(panel12, label='Resampling(Every n sample)=', pos=(15,170))
self.text1 = wx.TextCtrl(panel12, -1, pos=(185,165),size=(50,25),value='1')
Title2b = wx.StaticText(panel12, label='Feedback Gain (ro)=', pos=(250,170))
self.text1b = wx.TextCtrl(panel12, -1, pos=(360,165),size=(50,25),value='1.0')
self.cb0 = wx.CheckBox(panel12, -1, 'y0=yreal, u0=ureal', (440, 170))
self.cb0.SetValue(True)
Title1c = wx.StaticText(panel12, label='Reference Model Setup', pos=(740, 10))
Image0 = wx.Image('Plant.jpg', wx.BITMAP_TYPE_ANY).ConvertToBitmap()
wx.StaticBitmap(panel12, -1, Image0, pos=(100,15), size=(350,150))
Image1 = wx.Image('ref.jpg', wx.BITMAP_TYPE_ANY).ConvertToBitmap()
wx.StaticBitmap(panel12, -1, Image1, pos=(740,30), size=(350,130))
line0 = wx.StaticLine(panel12, -1, pos=(725,15), size=(3,200))
Title2c = wx.StaticText(panel12, label='Td=', pos=(750, 170))
Title2d = wx.StaticText(panel12, label='tau=', pos=(850, 170))
Title2e = wx.StaticText(panel12, label='Sampling=', pos=(950, 170))
self.text2c = wx.TextCtrl(panel12, -1, pos=(775, 165),size=(40,25),value='0')
self.text2d = wx.TextCtrl(panel12, -1, pos=(880, 165),size=(40,25),value='0.15')
self.text2e = wx.TextCtrl(panel12, -1, pos=(1035, 165),size=(40,25),value='0.1')
self.Bind(wx.EVT_BUTTON, self.plot_data, button1)
self.Bind(wx.EVT_CLOSE, self.closewindow)

panel14 = wx.Panel(panel1, -1, pos=(10,240),size=(1100,220))
panel14.SetBackgroundColour("White")
Title18 = wx.StaticText(panel14,label='State Feedback Neuro-controller', pos=(5, 5))
Title21 = wx.StaticText(panel14, label='nqy=', pos=(580, 65))
Title21b = wx.StaticText(panel14, label='nqd=', pos=(645, 65))
Title21c = wx.StaticText(panel14, label='muro=', pos=(25, 120))
Title21d = wx.StaticText(panel14, label='System :', pos=(170, 30))
Title21e = wx.StaticText(panel14, label='Controller:', pos=(170, 60))
Title21f = wx.StaticText(panel14, label='ny=', pos=(580, 30))
Title21g = wx.StaticText(panel14, label='nu=', pos=(645, 30))
self.cb4 = wx.CheckBox(panel14, -1, 'LNU', (260, 60))
self.cb5 = wx.CheckBox(panel14, -1, 'QNU', (380, 60))
self.cb5.SetValue(True)
self.cb6 = wx.CheckBox(panel14, -1, 'CNU', (500, 60))
self.cb7 = wx.CheckBox(panel14, -1, 'LNU', (260, 30))
self.cb8 = wx.CheckBox(panel14, -1, 'QNU', (380, 30))
self.cb8.SetValue(True)
self.cb9 = wx.CheckBox(panel14, -1, 'CNU', (500, 30))

self.text19b = wx.TextCtrl(panel14, -1, pos=(345, 155),size=(35,25),value='100')
Title26 = wx.StaticText(panel14, label='muGD=', pos=(170, 120))
Title27 = wx.StaticText(panel14, label='epochsGD=', pos=(270, 120))
Title28 = wx.StaticText(panel14, label='muBPTT=', pos=(170, 160))
Title29 = wx.StaticText(panel14, label='epochsBPTT=', pos=(270, 160))
self.text20 = wx.TextCtrl(panel14, -1, pos=(225, 155),size=(40,25),value='0.01')
line2 = wx.StaticLine(panel14, -1, pos=(160,95), size=(580,3))
self.text21f = wx.TextCtrl(panel14, -1, pos=(610, 25),size=(30,25),value='4')
```

```
        self.text21g = wx.TextCtrl(panel14, -1, pos=(675, 25),size=(30,25),value='7')
        self.text22 = wx.TextCtrl(panel14, -1, pos=(610, 60),size=(30,25),value='4')
        self.text22b = wx.TextCtrl(panel14, -1, pos=(675, 60),size=(30,25),value='7')
        self.text22c = wx.TextCtrl(panel14, -1, pos=(65, 115),size=(40,25),value='0.01')
        self.text25 = wx.TextCtrl(panel14, -1, pos=(220, 115),size=(40,25),value='0.001')
        self.text26 = wx.TextCtrl(panel14, -1, pos=(340, 115),size=(40,25),value='1000')
        button6 = wx.Button(panel14,label="Train with GD",pos=(390,120),size=(170,40))
        button6b = wx.Button(panel14,label="via. BPTT",pos=(570,120),size=(170,40))
        Image3 = wx.Image('Control.jpg', wx.BITMAP_TYPE_ANY).ConvertToBitmap()
        wx.StaticBitmap(panel14, -1, Image3, pos=(760,15), size=(340,150))
        self.Bind(wx.EVT_BUTTON, self.Tune_GD, button6)
        self.Bind(wx.EVT_BUTTON, self.Tune_LM, button6b)

    def Execute(self,event):
        self.DestroyChildren()

        if os.getcwd() == origWD:

            panel1 = wx.lib.scrolledpanel.ScrolledPanel(self,-
1,pos=(0,0),size=(1130,490),style=wx.SIMPLE_BORDER)
            panel1.SetupScrolling()
            panel1.SetBackgroundColour("White")
            box = wx.BoxSizer(wx.VERTICAL)
            box.Add(panel1, 1, wx.EXPAND)
            self.SetAutoLayout(True)
            self.SetSizer(box)
            self.Layout()

            panel0 = wx.Panel(panel1, -1, pos=(0,0),size=(1110,480) )

            wx.StaticText(panel0, label='Desired value=',pos=(5,7))
            self.text1 = wx.TextCtrl(panel0,-1, pos=(100,5), size=(50,25), value='0.8')
            wx.StaticText(panel0, label='Variable signlal',pos=(155,7))
            self.cb4 = wx.CheckBox(panel0, -1, '', (240,5))
            self.cb4.SetValue(True)
            wx.StaticText(panel0, label='ny=',pos=(5,34))
            self.text2 = wx.TextCtrl(panel0,-1, pos=(100,32), size=(50,25), value='4')
            wx.StaticText(panel0, label='nu=',pos=(5,61))
            self.text3 = wx.TextCtrl(panel0,-1, pos=(100,59), size=(50,25), value='7')
            wx.StaticText(panel0, label='constants=',pos=(5,88))
            self.text4 = wx.TextCtrl(panel0,-1, pos=(100,86), size=(100,25), value='vectorval.txt')
            wx.StaticText(panel0, label='Neural weights=',pos=(5,115))
            self.text5 = wx.TextCtrl(panel0,-1, pos=(100,113), size=(50,25), value='v.txt')
            button1 = wx.Button(panel0,label="Check",pos=(152,113),size=(50,25))
            self.Bind(wx.EVT_BUTTON, self.load_data, button1)

            self.cb1 = wx.CheckBox(panel0, -1, 'LNU', (5, 140))
            self.cb2 = wx.CheckBox(panel0, -1, 'QNU', (55, 140))
            self.cb3 = wx.CheckBox(panel0, -1, 'CNU', (110, 140))
            self.cb2.SetValue(True)

            button2=wx.Button(panel0,label="Run Model", pos=(5,167), size=(150,50))
            self.Bind(wx.EVT_BUTTON, self.adaptive_controller,button2)
            button3=wx.Button(panel0,label="Terminate", pos=(5,219), size=(150,25))
            self.Bind(wx.EVT_BUTTON, self.on_stop,button3)
            button4=wx.Button(panel0,label="Save data", pos=(5,246), size=(150,25))
            self.Bind(wx.EVT_BUTTON, self.on_save,button4)
            button5=wx.Button(panel0,label="Pause", pos=(157,219), size=(100,25))
            self.Bind(wx.EVT_BUTTON, self.on_pause,button5)
            self.text6 = wx.TextCtrl(panel0,-1, pos=(157,246), size=(100,25), value='simout01.txt')

            Image1 = wx.Image('PlantP.jpg', wx.BITMAP_TYPE_ANY).ConvertToBitmap()
            wx.StaticBitmap(panel0, -1, Image1, pos=(5,265), size=(260,135))
            wx.StaticLine(panel0,-1,pos=(265,5),size=(3,400))

            ion()
            self.panel01 = wx.Panel(panel0,-1,pos=(270,5),size=(840,400))
            self.panel01.figure = figure()
            self.panel01.figure.set_size_inches((10.6,4))
            self.panel01.axes = self.panel01.figure.add_subplot(111)
            self.panel01.axes.grid()
            self.line1,=self.panel01.axes.plot([],[],'g')
            self.line2,=self.panel01.axes.plot([],[],'b')
            self.panel01.canvas =FigureCanvas(self.panel01,-1,self.panel01.figure)
            self.panel01.toolbar = NavigationToolbar(self.panel01.canvas)
            self.panel01.toolbar.Realize()
            self.panel01.sizer = wx.BoxSizer(wx.VERTICAL)
            self.panel01.sizer.Add(self.panel01.canvas,1, wx.LEFT | wx.TOP | wx.GROW)
            self.panel01.sizer.Add(self.panel01.toolbar,0, wx.LEFT | wx.EXPAND)
            self.panel01.SetSizer(self.panel01.sizer)
            self.panel01.Fit()
            close("all")
            self.CreateStatusBar()
            self.SetStatusText("Connect LabJack before running application")

            d=u3.U3()
            d.getCalibrationData()
            d.configU3()
            d.configIO(FIOAnalog=3)
            d.getFeedback(u3.DAC8(0,0))

        else:
            os.chdir(str(origWD))

            panel1 = wx.lib.scrolledpanel.ScrolledPanel(self,-
1,pos=(0,0),size=(1130,490),style=wx.SIMPLE_BORDER)
            panel1.SetupScrolling()
            panel1.SetBackgroundColour("White")
            box = wx.BoxSizer(wx.VERTICAL)
            box.Add(panel1, 1, wx.EXPAND)
            self.SetAutoLayout(True)
            self.SetSizer(box)
            self.Layout()

            panel0 = wx.Panel(panel1, -1, pos=(0,0),size=(1110,480) )

            wx.StaticText(panel0, label='Desired value=',pos=(5,7))
            self.text1 = wx.TextCtrl(panel0,-1, pos=(100,5), size=(50,25), value='0.8')
            wx.StaticText(panel0, label='Variable signlal',pos=(155,7))
            self.cb4 = wx.CheckBox(panel0, -1, '', (240,5))
            self.cb4.SetValue(True)
            wx.StaticText(panel0, label='ny=',pos=(5,34))
            self.text2 = wx.TextCtrl(panel0,-1, pos=(100,32), size=(50,25), value='4')
            wx.StaticText(panel0, label='nu=',pos=(5,61))
            self.text3 = wx.TextCtrl(panel0,-1, pos=(100,59), size=(50,25), value='7')
            wx.StaticText(panel0, label='constants=',pos=(5,88))
            self.text4 = wx.TextCtrl(panel0,-1, pos=(100,86), size=(100,25), value='vectorval.txt')
            wx.StaticText(panel0, label='Neural weights=',pos=(5,115))
            self.text5 = wx.TextCtrl(panel0,-1, pos=(100,113), size=(50,25), value='v.txt')
            button1 = wx.Button(panel0,label="Check",pos=(152,113),size=(50,25))
            self.Bind(wx.EVT_BUTTON, self.load_data, button1)

            self.cb1 = wx.CheckBox(panel0, -1, 'LNU', (5, 140))
            self.cb2 = wx.CheckBox(panel0, -1, 'QNU', (55, 140))
            self.cb3 = wx.CheckBox(panel0, -1, 'CNU', (110, 140))
            self.cb2.SetValue(True)

            button2=wx.Button(panel0,label="Run Model", pos=(5,167), size=(150,50))
            self.Bind(wx.EVT_BUTTON, self.adaptive_controller,button2)
            button3=wx.Button(panel0,label="Terminate", pos=(5,219), size=(150,25))
            self.Bind(wx.EVT_BUTTON, self.on_stop,button3)
            button4=wx.Button(panel0,label="Save data", pos=(5,246), size=(150,25))
            self.Bind(wx.EVT_BUTTON, self.on_save,button4)
            button5=wx.Button(panel0,label="Pause", pos=(157,219), size=(100,25))
            self.Bind(wx.EVT_BUTTON, self.on_pause,button5)
            self.text6 = wx.TextCtrl(panel0,-1, pos=(157,246), size=(100,25), value='simout01.txt')

            Image1 = wx.Image('PlantP.jpg', wx.BITMAP_TYPE_ANY).ConvertToBitmap()
            wx.StaticBitmap(panel0, -1, Image1, pos=(5,265), size=(260,135))
            wx.StaticLine(panel0,-1,pos=(265,5),size=(3,400))

            ion()
            self.panel01 = wx.Panel(panel0,-1,pos=(270,5),size=(840,400))
            self.panel01.figure = figure()
            self.panel01.figure.set_size_inches((10.6,4))
            self.panel01.axes = self.panel01.figure.add_subplot(111)
            self.panel01.axes.grid()
            self.line1,=self.panel01.axes.plot([],[],'g')
            self.line2,=self.panel01.axes.plot([],[],'b')
            self.panel01.canvas =FigureCanvas(self.panel01,-1,self.panel01.figure)
            self.panel01.toolbar = NavigationToolbar(self.panel01.canvas)
            self.panel01.toolbar.Realize()
            self.panel01.sizer = wx.BoxSizer(wx.VERTICAL)
            self.panel01.sizer.Add(self.panel01.canvas,1, wx.LEFT | wx.TOP | wx.GROW)
            self.panel01.sizer.Add(self.panel01.toolbar,0, wx.LEFT | wx.EXPAND)
            self.panel01.SetSizer(self.panel01.sizer)
            self.panel01.Fit()
            close("all")
            self.CreateStatusBar()
            self.SetStatusText("Connect LabJack before running application")

            d=u3.U3()
            d.getCalibrationData()
            d.configU3()
            d.configIO(FIOAnalog=3)
```

```
        d.getFeedback(u3.DAC8(0,0))

def adaptive_controller(self, event):
    os.chdir(str(origWD + '/Data_files'))

    cons=self.text4.GetValue()
    nxiy=int(self.text2.GetValue())
    nxid=int(self.text3.GetValue())
    vi=self.text5.GetValue()

    if self.cb1.GetValue() == True:
        r=1
    if self.cb2.GetValue() == True:
        r=2
    if self.cb3.GetValue() == True:
        r=3
    if self.cb4.GetValue() == True:
        desired=0
    else:
        desired=float(self.text1.GetValue())


    print 'desired=',desired
    print 'ny=',nxiy
    print 'nu=',nxid
    print 'order=',r

    try:
        d=u3.U3()
        d.getCalibrationData()
        d.configU3()
        d.configIO(FIOAnalog=3)
        print "U3 is connected"

        #parameter initialization
        out_port=1;   inp_port=0;
        self.inp=[];    self.out=[];    self.t_g=[];
        ci=[];    bi=[];    ai=[]; des=[];
        dead=1;    test1=11;    test2=21;    test3=31;    test4=41;
        test5=51;  t=61
        nxi=1+nxiy+nxid
        xi=ones(nxi)
        v=loadtxt(vi)
        c=loadtxt(cons)

        start=time.time()
        print "started at:    ",datetime.now()
        counter=0

        def update_line(t_g,out,inp):
            ai.append(t_g);   bi.append(out);  ci.append(inp)
            self.line1.set_xdata(ai); self.line1.set_ydata(bi);
            self.line2.set_ydata(ci); self.line2.set_xdata(ai);
            self.panel01.figure.set_size_inches((10.6,4.5))
            self.panel01.axes.relim()
    self.panel01.axes.autoscale_view()
    self.panel01.canvas.draw()
    self.panel01.Fit()
    self.panel01.canvas.flush_events()

def fcolx(x,r):
    if r==1:
        colx=x
    if r==2:
        n=len(x)
        pom=0
        colx=zeros((n*n+n)/2)
        for i in range(n):
            for j in range(i,n):
                colx[pom]=x[i]*x[j]
                pom+=1
    if r==3:
        n=len(x)
        pom=0
        nw=0
        for i in range(n):
            for j in range(i,n):
                for l in range(j,n):
                    nw=nw+1

        colx=zeros(nw)
        for i in range(n):
            for j in range(i,n):
                for l in range(j,n):
                    colx[pom]=x[i]*x[j]*x[l]
                    pom+=1
    return(colx)

desired_bit = d.voltageToDACBits(desired,dacNumber = 0, is16Bits = False)
d.getFeedback(u3.DAC8(0,desired_bit))


while(t>= time.time()-start):
    i=datetime.now()

    a=d.getAIN(out_port); self.out.append(a)
    b=d.getAIN(inp_port); self.inp.append(b)
    des.append(desired)

    self.t_g.append(time.time()-start)
    meany=c[1]
    stdy=c[2]
    meanu=c[3]
    stdu=c[4]
    ro=c[0]

    if len(self.out)<max(nxiy,nxid):
        q=0;
    else:
        for n1 in range(1,nxiy+1):
                xi[n1]=(self.out[-n1]-meany)/3/stdy
            for n2 in range(nxiy+1,nxiy+nxid+1):
                xi[n2]=(des[nxiy-n2]-meanu)/3/stdu
            colxi=fcolx(xi,r)
            q=dot(v,colxi)*3*stdy+meany

        setval = desired-q*ro
        setbit=d.voltageToDACBits(setval,dacNumber = 0, is16Bits = False)
        d.getFeedback(u3.DAC8(0,setbit))

        counter=counter+1
        end=time.time()-start

        if counter==1:
            update_line(self.t_g[-1], self.out[-1],desired)
            counter=0
        if self.cb4.GetValue() == True:
            if end < dead:
                desired=0;
            if dead < end < test1:
                desired=0.59;
            if test1< end < test2:
                desired=1.4;
            if test2 <end < test3:
                desired=2.24;
            if test3< end < test4:
                desired=2.67;
            if test4< end < test5:
                desired=1.835;
            if test5< end < t:
                desired=1.01;

        if (t-0.2)<=end<=t:
            print "Terminated at: ", datetime.now()
            print "I got: ",len(self.out)," samples"
            print "in: ",end, " seconds"

        f=datetime.now()
        w=(f.second+f.microsecond/1000000.)-(i.second+i.microsecond/1000000.)
        dt=0.2-w
        if dt<0.001:
            dt=0
        sleep(dt)

    except:
        d.getFeedback(u3.DAC8(0,0))
        d.close()
        wx.MessageBox("U3 device is not detected",
            "Connection Problem", wx.OK | wx.ICON_INFORMATION, self)
    os.chdir(str(origWD))

def OnlineUpdate(self,event):
    self.DestroyChildren()

    if os.getcwd() == origWD:
        panel1 = wx.lib.scrolledpanel.ScrolledPanel(self,-
1,pos=(0,0),size=(1130,490),style=wx.SIMPLE_BORDER)
        panel1.SetupScrolling()
        panel1.SetBackgroundColour("White")
        box = wx.BoxSizer(wx.VERTICAL)
        box.Add(panel1, 1, wx.EXPAND)
        self.SetAutoLayout(True)
        self.SetSizer(box)
        self.Layout()

        panel20 = wx.Panel(panel1, -1, pos=(0,0),size=(1110,480) )
        Title20 = wx.StaticText(panel20, label='Reference Model Setup', pos=(65, 7))
        Image2 = wx.Image('ref2.jpg', wx.BITMAP_TYPE_ANY).ConvertToBitmap()
        wx.StaticBitmap(panel20, -1, Image2, pos=(5,30), size=(260,130))
        Title21 = wx.StaticText(panel20, label='Td=', pos=(10, 150))
        Title22 = wx.StaticText(panel20, label='tau=', pos=(90, 150))
        Title23 = wx.StaticText(panel20, label='Sampling=', pos=(180, 150))
        self.text7 = wx.TextCtrl(panel20, -1, pos=(35, 145),size=(30,25),value='0')
        self.text8 = wx.TextCtrl(panel20, -1, pos=(120, 145),size=(30,25),value='0.15')
        self.text9 = wx.TextCtrl(panel20, -1, pos=(230, 145),size=(30,25),value='0.1')

        wx.StaticText(panel20, label='Desired value=',pos=(5,180))
        self.text1 = wx.TextCtrl(panel20,-1, pos=(100,178), size=(50,25), value='0.8')
        wx.StaticText(panel20, label='ny=',pos=(5,207))
        self.text2 = wx.TextCtrl(panel20,-1, pos=(100,205), size=(50,25), value='4')
        wx.StaticText(panel20, label='nu=',pos=(5,234))
        self.text3 = wx.TextCtrl(panel20,-1, pos=(100,232), size=(50,25), value='7')
        wx.StaticText(panel20, label='constants=',pos=(5,261))
        self.text4 = wx.TextCtrl(panel20,-1, pos=(100,259), size=(100,25), value='vectorval.txt')
        wx.StaticText(panel20, label='Neural weights=',pos=(5,288))
        self.text5 = wx.TextCtrl(panel20,-1, pos=(100,286), size=(50,25), value='v.txt')
        button1 = wx.Button(panel20,label="Check",pos=(152,286),size=(50,25))
        self.Bind(wx.EVT_BUTTON, self.load_data, button1)
        self.cb1 = wx.CheckBox(panel20, -1, 'LNU', (5, 313))
        self.cb2 = wx.CheckBox(panel20, -1, 'QNU', (55, 313))
        self.cb3 = wx.CheckBox(panel20, -1, 'CNU', (110, 313))
        self.cb2.SetValue(True)

        button2=wx.Button(panel20,label="Run Model", pos=(5,333), size=(150,25))
        self.Bind(wx.EVT_BUTTON, self.update,button2)
        button3=wx.Button(panel20,label="Terminate", pos=(5,360), size=(150,22.5))
        self.Bind(wx.EVT_BUTTON, self.on_stop,button3)
        button4=wx.Button(panel20,label="Save data", pos=(5,385), size=(150,22.5))
        self.Bind(wx.EVT_BUTTON, self.on_save,button4)
        button5=wx.Button(panel20,label="Pause", pos=(157,360), size=(100,22.5))
        self.Bind(wx.EVT_BUTTON, self.on_pause,button5)
        self.text6 = wx.TextCtrl(panel20,-1, pos=(157,385), size=(100,22.5), value='simout01.txt')

        wx.StaticLine(panel20,-1,pos=(265,5),size=(3,400))

        ion()
        self.panel011 = wx.Panel(panel20,-1,pos=(270,5),size=(840,400))
        self.panel011.figure = figure()
        self.panel011.figure.set_size_inches((10.6,4))
        self.panel011.axes = self.panel011.figure.add_subplot(111)
        self.panel011.axes.grid()
        self.line1,=self.panel011.axes.plot([],[],'g')
        self.line2,=self.panel011.axes.plot([],[],'b')
        self.panel011.canvas =FigureCanvas(self.panel011,-1,self.panel011.figure)
        self.panel011.toolbar = NavigationToolbar(self.panel011.canvas)
        self.panel011.toolbar.Realize()
        self.panel011.sizer = wx.BoxSizer(wx.VERTICAL)
        self.panel011.sizer.Add(self.panel011.canvas,1, wx.LEFT | wx.TOP | wx.GROW)
        self.panel011.sizer.Add(self.panel011.toolbar,0, wx.LEFT | wx.EXPAND)
        self.panel011.SetSizer(self.panel011.sizer)
        self.panel011.Fit()
        close("all")
        self.CreateStatusBar()
        self.SetStatusText("Connect LabJack before running application")


        d=u3.U3()
        d.getCalibrationData()
        d.configU3()
        d.configIO(FIOAnalog=3)
        d.getFeedback(u3.DAC8(0,0))

    else:
        os.chdir(str(origWD))
        panel1 = wx.lib.scrolledpanel.ScrolledPanel(self,-
1,pos=(0,0),size=(1130,490),style=wx.SIMPLE_BORDER)
        panel1.SetupScrolling()
        panel1.SetBackgroundColour("White")
        box = wx.BoxSizer(wx.VERTICAL)
        box.Add(panel1, 1, wx.EXPAND)
        self.SetAutoLayout(True)
        self.SetSizer(box)
        self.Layout()


        panel20 = wx.Panel(panel1, -1, pos=(0,0),size=(1110,480) )
        Title20 = wx.StaticText(panel20, label='Reference Model Setup', pos=(65, 7))
        Image2 = wx.Image('ref2.jpg', wx.BITMAP_TYPE_ANY).ConvertToBitmap()
        wx.StaticBitmap(panel20, -1, Image2, pos=(5,30), size=(260,130))
        Title21 = wx.StaticText(panel20, label='Td=', pos=(10, 150))
        Title22 = wx.StaticText(panel20, label='tau=', pos=(90, 150))
        Title23 = wx.StaticText(panel20, label='Sampling=', pos=(180, 150))
        self.text7 = wx.TextCtrl(panel20, -1, pos=(35, 145),size=(30,25),value='0')
        self.text8 = wx.TextCtrl(panel20, -1, pos=(120, 145),size=(30,25),value='0.15')
        self.text9 = wx.TextCtrl(panel20, -1, pos=(230, 145),size=(30,25),value='0.1')

        wx.StaticText(panel20, label='Desired value=',pos=(5,180))
        self.text1 = wx.TextCtrl(panel20,-1, pos=(100,178), size=(50,25), value='0.8')
        wx.StaticText(panel20, label='ny=',pos=(5,207))
        self.text2 = wx.TextCtrl(panel20,-1, pos=(100,205), size=(50,25), value='4')
        wx.StaticText(panel20, label='nu=',pos=(5,234))
        self.text3 = wx.TextCtrl(panel20,-1, pos=(100,232), size=(50,25), value='7')
        wx.StaticText(panel20, label='constants=',pos=(5,261))
        self.text4 = wx.TextCtrl(panel20,-1, pos=(100,259), size=(100,25), value='vectorval.txt')
        wx.StaticText(panel20, label='Neural weights=',pos=(5,288))
        self.text5 = wx.TextCtrl(panel20,-1, pos=(100,286), size=(50,25), value='v.txt')
        button1 = wx.Button(panel20,label="Check",pos=(152,286),size=(50,25))
        self.Bind(wx.EVT_BUTTON, self.load_data, button1)

        self.cb1 = wx.CheckBox(panel20, -1, 'LNU', (5, 313))
        self.cb2 = wx.CheckBox(panel20, -1, 'QNU', (55, 313))
        self.cb3 = wx.CheckBox(panel20, -1, 'CNU', (110, 313))
        self.cb2.SetValue(True)


        button2=wx.Button(panel20,label="Run Model", pos=(5,333), size=(150,25))
        self.Bind(wx.EVT_BUTTON, self.update,button2)
        button3=wx.Button(panel20,label="Terminate", pos=(5,360), size=(150,22.5))
        self.Bind(wx.EVT_BUTTON, self.on_stop,button3)
        button4=wx.Button(panel20,label="Save data", pos=(5,385), size=(150,22.5))
        self.Bind(wx.EVT_BUTTON, self.on_save,button4)
        button5=wx.Button(panel20,label="Pause", pos=(157,360), size=(100,22.5))
        self.Bind(wx.EVT_BUTTON, self.on_pause,button5)
        self.text6 = wx.TextCtrl(panel20,-1, pos=(157,385), size=(100,22.5), value='simout01.txt')

        wx.StaticLine(panel20,-1,pos=(265,5),size=(3,400))

        ion()
        self.panel011 = wx.Panel(panel20,-1,pos=(270,5),size=(840,400))
        self.panel011.figure = figure()
        self.panel011.figure.set_size_inches((10.6,4))
        self.panel011.axes = self.panel011.figure.add_subplot(111)
        self.panel011.axes.grid()
        self.line1,=self.panel011.axes.plot([],[],'g')
        self.line2,=self.panel011.axes.plot([],[],'b')
        self.panel011.canvas =FigureCanvas(self.panel011,-1,self.panel011.figure)
        self.panel011.toolbar = NavigationToolbar(self.panel011.canvas)
        self.panel011.toolbar.Realize()
        self.panel011.sizer = wx.BoxSizer(wx.VERTICAL)
        self.panel011.sizer.Add(self.panel011.canvas,1, wx.LEFT | wx.TOP | wx.GROW)
        self.panel011.sizer.Add(self.panel011.toolbar,0, wx.LEFT | wx.EXPAND)
        self.panel011.SetSizer(self.panel011.sizer)
        self.panel011.Fit()
        close("all")
        self.CreateStatusBar()
        self.SetStatusText("Connect LabJack before running application")

        d=u3.U3()
        d.getCalibrationData()
        d.configU3()
        d.configIO(FIOAnalog=3)
        d.getFeedback(u3.DAC8(0,0))


    def on_stop(self, event):
        d=u3.U3()
        d.getCalibrationData()
        d.configU3()
        d.configIO(FIOAnalog=3)
        d.getFeedback(u3.DAC8(0,0))
        os._exit(0)

    def on_pause(self, event):
        os.system("pause")

    def on_save(self,event):
        name=self.text6.GetValue()
        savetxt(name,(self.inp,self.out,self.t_g))
        print 'saved'

    def load_data(self, event):
        os.startfile(os.getcwd())

    def update(self, event):
        os.chdir(str(origWD + '/Data_files'))

        desired=0#float(self.text1.GetValue())
        yn=desired
        cons=self.text4.GetValue()
        nxiy=int(self.text2.GetValue())
        nxid=int(self.text3.GetValue())
        vi=self.text5.GetValue()
        dT = float(self.text9.GetValue())
        tau = float(self.text8.GetValue())
        Td = float(self.text7.GetValue())
        w=loadtxt('w.txt')
        dydv=loadtxt('dydv.txt')
        dxdv=loadtxt('dxdv.txt')
        dcolxdv=loadtxt('dcolxdv.txt')
        dxdro=loadtxt('dxdro.txt')
        dcolxdro= loadtxt('dcolxdro.txt')

        if self.cb1.GetValue() == True:
            r=1
        if self.cb2.GetValue() == True:
            r=2
        if self.cb3.GetValue() == True:
            r=3
        print 'desired=',desired
        print 'ny=',nxiy
        print 'nu=',nxid
        print 'order=',r

        try:

            d=u3.U3()
            d.getCalibrationData()
            d.configU3()
            d.configIO(FIOAnalog=3)
            print "U3 is connected"

            #parameter initialization
            out_port=1;   inp_port=0;
            self.inp=[];    self.out=[];    self.t_g=[];
```

```
ci=[];    bi=[];    ai=[]; des=[];
dead=1;    test1=11;    test2=21;    test3=31;    test4=41;
test5=51;  t=61; flag=-1;
nxi=1+nxiy+nxid
xi=ones(nxi)
v=loadtxt(vi)
c=loadtxt(cons)
ro=c[0]
meany=c[1];    stdy=c[2];   meanu=c[3];   stdu=c[4];
muv=c[5];  muro=c[6] ;  ny=int(c[7]);  nd=int(c[8]); dydro=c[9]
nx=int(1+ny+nd);   x=ones(nx)
yn=[];  SSE=[]

des.append(desired)
yref0 = 0
start=time.time()
print "started at:    ",datetime.now()
counter=0

def update_line(t_g,out,inp):
    ai.append(t_g);   bi.append(out);  ci.append(inp)
    self.line1.set_xdata(ai); self.line1.set_ydata(bi);
    self.line2.set_ydata(ci); self.line2.set_xdata(ai);
    self.panel011.figure.set_size_inches((10.6,4.5))
    self.panel011.axes.relim()
    self.panel011.axes.autoscale_view()
    self.panel011.canvas.draw()
    self.panel011.Fit()
    self.panel011.canvas.flush_events()

def fcolx(x,r):
    if r==1:
        colx=x
    if r==2:
        n=len(x)
        pom=0
        colx=zeros((n*n+n)/2)
        for i in range(n):
            for j in range(i,n):
                colx[pom]=x[i]*x[j]
                pom+=1
    if r==3:
        n=len(x)
        pom=0
        nw=0
        for i in range(n):
            for j in range(i,n):
                for l in range(j,n):
                    nw=nw+1

        colx=zeros(nw)
        for i in range(n):
            for j in range(i,n):
                for l in range(j,n):
                    colx[pom]=x[i]*x[j]*x[l]
            pom+=1
    return(colx)

desired_bit = d.voltageToDACBits(desired,dacNumber = 0, is16Bits = False)
d.getFeedback(u3.DAC8(0,desired_bit))

v0=v.copy()
ro0=ro

while(t>= time.time()-start):
    i=datetime.now()
    a=d.getAIN(out_port); self.out.append(a)
    b=d.getAIN(inp_port); self.inp.append(b)

    self.t_g.append(time.time()-start)
    if len(self.out)<max(ny,nd,nxiy,nxid):
        colxi=fcolx(xi,r)
        q=0;
    else:
        for n1 in range(1,nxiy+1):
            xi[n1]=(self.out[-n1]-meany)/3/stdy
        for n2 in range(nxiy+1,nxiy+nxid+1):
            xi[n2]=(des[nxiy-n2]-meanu)/3/stdu
        for n3 in range(1,ny+1):
            x[n3]=(self.out[-n3]-meany)/3/stdy
        for n4 in range(ny+1,ny+nd+1):
            x[n4]=(self.inp[ny-n4]-meanu)/3/stdu

        colxi=fcolx(xi,r)
        q=dot(v,colxi)*3*stdy+meany

    setval = desired-q*ro
    setbit=d.voltageToDACBits(setval,dacNumber = 0, is16Bits = False)
    d.getFeedback(u3.DAC8(0,setbit))

    counter=counter+1
    end=time.time()-start

    if counter==1:
        update_line(self.t_g[-1], self.out[-1],desired)
        counter=0

    if end < dead:
        desired=0;
    if dead < end < test1:
        desired=0.59;
    if test1< end < test2:
        desired=1.4;
    if test2 <end < test3:
        desired=2.24;
    if test3< end < test4:
        desired=2.67;
    if test4< end < test5:
        desired=1.835;
    if test5< end < t:
```

```
    desired=1.4;
des.append(desired)

    #q-updates
colx=fcolx(x,r)
yneural=dot(w,colx)*3*stdy+meany
yn.append(yneural)

if yref0==desired:
    yref = ref_model2(flag,des,dT,tau,Td)
    yref0=yref
else:
    yref=ref_model2(yref0,des,dT,tau,Td)
    yref0=yref

eref=yref-yn[-1]#self.out[-1]#

if r==1:
    dydv=dot(w,dxdv)*3*stdy
    dxdv[1:ny,:]=dxdv[2:1+ny,:];dxdv[ny,:]=dydv/3/stdy
    dxdv[1+ny:-1,:]=dxdv[2+ny:,:];dxdv[-1,:]=-ro*stdy/stdu*colxi
    dydro=dot(w,dxdro)*3*stdy
    dxdro[1:ny]=dxdro[2:1+ny];dxdro[ny]=dydro/3/stdy
    dxdro[1+ny:-1]=dxdro[2+ny:];dxdro[-1]=-q/3/stdu
    dv=muv/(1+dot(dydv,dydv))*eref*dydv/3/stdy
    v=v+dv
    dro=muro/(1+dot(dydro,dydro))*eref*dydro/3/stdy
    ro=ro+dro

if r==2:
    pom=0
    for m1 in range(nx):
        for m2 in range(m1,nx):
            dcolxdv[pom]=dxdv[m1,:]*x[m2]+x[m1]*dxdv[m2,:]
            dcolxdro[pom]=dxdro[m1]*x[m2]+x[m1]*dxdro[m2]
    pom+=1
    dydv=dot(w,dcolxdv)*3*stdy
    dv=muv/(1+dot(dydv,dydv))*eref*dydv/3/stdy
    v=v+dv
    dxdv[1:ny,:]=dxdv[2:1+ny,:];dxdv[ny,:]=dydv/3/stdy
    dxdv[1+ny:-1,:]=dxdv[2+ny:,:];dxdv[-1,:]=-ro*stdy/stdu*colxi
    dydro=dot(w,dcolxdro)*3*stdy
    dro=muro/(1+dot(dydro,dydro))*eref*dydro/3/stdy
    ro=ro+dro
    dxdro[1:ny]=dxdro[2:1+ny];dxdro[ny]=dydro/3/stdy
    dxdro[1+ny:-1]=dxdro[2+ny:];dxdro[-1]=-q/3/stdu

if r==3:
    pom=0
    for i in range(nx):
        for j in range(i,nx):
            for k in range(j,nx):
                dcolxdv[pom]=dxdv[i,:]*x[j]*x[k] + x[i]*dxdv[j,:]*x[k] + x[i]*x[j]*dxdv[k,:]
                dcolxdro[pom]=dxdro[i]*x[j]*x[k] + x[i]*dxdro[j]*x[k] + x[i]*x[j]*dxdro[k]
                pom+=1
                dydv=dot(w,dcolxdv)*3*stdy
                dv=muv/(1+dot(dydv,dydv))*eref*dydv/3/stdy
                v=v+dv
                dxdv[1:ny,:]=dxdv[2:1+ny,:];dxdv[ny,:]=dydv/3/stdy
                dxdv[1+ny:-1,:]=dxdv[2+ny:,:];dxdv[-1,:]=-ro*stdy/stdu*colxi
                dydro=dot(w,dcolxdro)*3*stdy
                dro=muro/(1+dot(dydro,dydro))*eref*dydro/3/stdy
                ro=ro+dro
                dxdro[1:ny]=dxdro[2:1+ny];dxdro[ny]=dydro/3/stdy
                dxdro[1+ny:-1]=dxdro[2+ny:];dxdro[-1]=-q/3/stdu

            error2=sum(eref*eref)
            SSE.append(error2)
            if end>2 and SSE[-1]>=SSE[-2]:
                dv=v-v0;dro=ro-ro0
                v=v-dv*0.999
                ro=ro-dro*0.999
                muv=muv*0.9
                muro=muro*0.9
                print 'reduced'

            f=datetime.now()
            diff=(f.second+f.microsecond/1000000.)-(i.second+i.microsecond/1000000.)
            dt=0.2-diff
            if dt<0.001:
                dt=0
            sleep(dt)

            if (t-0.2)<=end<=t:
                print "Terminated at: ", datetime.now()
                print "I got: ",len(self.out)," samples"
                print "in: ",end, " seconds"

        except:
            d.getFeedback(u3.DAC8(0,0))
            d.close()
            wx.MessageBox("U3 device is not detected",
                "Connection Problem", wx.OK | wx.ICON_INFORMATION, self)
        os.chdir(str(origWD))

    def GetData(self,event):
        self.DestroyChildren()

        if os.getcwd() == origWD:

            panel1 = wx.lib.scrolledpanel.ScrolledPanel(self,-
1,pos=(0,0),size=(1130,490),style=wx.SIMPLE_BORDER)
            panel1.SetupScrolling()
            panel1.GetEffectiveMinSize()

            panel16 = wx.Panel(panel1, -1, pos=(10,0),size=(1100,460))
            panel16.SetBackgroundColour("White")

            wx.StaticText(panel16, label='Input signal steps=',pos=(5,7))
            self.text30 = wx.TextCtrl(panel16,-1, pos=(200,5), size=(50,25), value='2')
            self.text31 = wx.TextCtrl(panel16,-1, pos=(255,5), size=(50,25), value='3')
            self.text32 = wx.TextCtrl(panel16,-1, pos=(310,5), size=(50,25), value='4')
            self.text33 = wx.TextCtrl(panel16,-1, pos=(365,5), size=(50,25), value='4.5')
            self.text34 = wx.TextCtrl(panel16,-1, pos=(420,5), size=(50,25), value='3.5')
            self.text35 = wx.TextCtrl(panel16,-1, pos=(475,5), size=(50,25), value='2.5')
            self.text36 = wx.TextCtrl(panel16,-1, pos=(530,5), size=(50,25), value='0')
            self.text37 = wx.TextCtrl(panel16,-1, pos=(585,5), size=(50,25), value='0')
            self.text38 = wx.TextCtrl(panel16,-1, pos=(640,5), size=(50,25), value='0')
            self.text39 = wx.TextCtrl(panel16,-1, pos=(695,5), size=(50,25), value='0')

            wx.StaticText(panel16, label='Duration of signal steps=',pos=(5,37))
            self.text40 = wx.TextCtrl(panel16,-1, pos=(200,35), size=(50,25), value='10')
            self.text41 = wx.TextCtrl(panel16,-1, pos=(255,35), size=(50,25), value='10')
            self.text42 = wx.TextCtrl(panel16,-1, pos=(310,35), size=(50,25), value='10')
            self.text43 = wx.TextCtrl(panel16,-1, pos=(365,35), size=(50,25), value='10')
            self.text44 = wx.TextCtrl(panel16,-1, pos=(420,35), size=(50,25), value='10')
            self.text45 = wx.TextCtrl(panel16,-1, pos=(475,35), size=(50,25), value='10')
            self.text46 = wx.TextCtrl(panel16,-1, pos=(530,35), size=(50,25), value='0')
            self.text47 = wx.TextCtrl(panel16,-1, pos=(585,35), size=(50,25), value='0')
            self.text48 = wx.TextCtrl(panel16,-1, pos=(640,35), size=(50,25), value='0')
            self.text49 = wx.TextCtrl(panel16,-1, pos=(695,35), size=(50,25), value='0')

            wx.StaticText(panel16, label='Sampling time=',pos=(5,67))
            self.text50 = wx.TextCtrl(panel16,-1, pos=(200,65), size=(50,25), value='0.2')
            button1=wx.Button(panel16,label="Obtain data", pos=(5,100), size=(150,50))
            self.Bind(wx.EVT_BUTTON, self.Obtain,button1)

        else:
            os.chdir(str(origWD))
            panel1 = wx.lib.scrolledpanel.ScrolledPanel(self,-
1,pos=(0,0),size=(1130,490),style=wx.SIMPLE_BORDER)
            panel1.SetupScrolling()
            panel1.GetEffectiveMinSize()

            panel16 = wx.Panel(panel1, -1, pos=(10,0),size=(1100,460))
            panel16.SetBackgroundColour("White")
            wx.StaticText(panel16, label='Input signal steps=',pos=(5,7))
            self.text30 = wx.TextCtrl(panel16,-1, pos=(200,5), size=(50,25), value='2')
            self.text31 = wx.TextCtrl(panel16,-1, pos=(255,5), size=(50,25), value='3')
            self.text32 = wx.TextCtrl(panel16,-1, pos=(310,5), size=(50,25), value='4')
            self.text33 = wx.TextCtrl(panel16,-1, pos=(365,5), size=(50,25), value='4.5')
            self.text34 = wx.TextCtrl(panel16,-1, pos=(420,5), size=(50,25), value='3.5')
            self.text35 = wx.TextCtrl(panel16,-1, pos=(475,5), size=(50,25), value='2.5')
            self.text36 = wx.TextCtrl(panel16,-1, pos=(530,5), size=(50,25), value='0')
            self.text37 = wx.TextCtrl(panel16,-1, pos=(585,5), size=(50,25), value='0')
            self.text38 = wx.TextCtrl(panel16,-1, pos=(640,5), size=(50,25), value='0')
            self.text39 = wx.TextCtrl(panel16,-1, pos=(695,5), size=(50,25), value='0')

            wx.StaticText(panel16, label='Duration of signal steps=',pos=(5,37))
            self.text40 = wx.TextCtrl(panel16,-1, pos=(200,35), size=(50,25), value='10')
            self.text41 = wx.TextCtrl(panel16,-1, pos=(255,35), size=(50,25), value='10')
            self.text42 = wx.TextCtrl(panel16,-1, pos=(310,35), size=(50,25), value='10')
            self.text43 = wx.TextCtrl(panel16,-1, pos=(365,35), size=(50,25), value='10')
            self.text44 = wx.TextCtrl(panel16,-1, pos=(420,35), size=(50,25), value='10')
            self.text45 = wx.TextCtrl(panel16,-1, pos=(475,35), size=(50,25), value='10')
            self.text46 = wx.TextCtrl(panel16,-1, pos=(530,35), size=(50,25), value='0')
            self.text47 = wx.TextCtrl(panel16,-1, pos=(585,35), size=(50,25), value='0')
            self.text48 = wx.TextCtrl(panel16,-1, pos=(640,35), size=(50,25), value='0')
            self.text49 = wx.TextCtrl(panel16,-1, pos=(695,35), size=(50,25), value='0')

            wx.StaticText(panel16, label='Sampling time=',pos=(5,67))
            self.text50 = wx.TextCtrl(panel16,-1, pos=(200,65), size=(50,25), value='0.2')

            button1=wx.Button(panel16,label="Obtain data", pos=(5,100), size=(150,50))
            self.Bind(wx.EVT_BUTTON, self.Obtain,button1)

    def Obtain(self, event):
        os.chdir(str(origWD + '/Data_files'))

        ST=float(self.text50.GetValue());   t1=int(self.text40.GetValue())
        t2=int(self.text41.GetValue());   t3=int(self.text42.GetValue())
        t4=int(self.text43.GetValue());   t5=int(self.text44.GetValue())
        t6=int(self.text45.GetValue());   t7=int(self.text46.GetValue())
        t8=int(self.text47.GetValue());   t9=int(self.text48.GetValue())
        t10=int(self.text49.GetValue())

        val1=float(self.text30.GetValue()); val2=float(self.text31.GetValue())
        val3=float(self.text32.GetValue()); val4=float(self.text33.GetValue())
        val5=float(self.text34.GetValue()); val6=float(self.text35.GetValue())
        val7=float(self.text36.GetValue()); val8=float(self.text37.GetValue())
        val9=float(self.text38.GetValue());  val10=float(self.text39.GetValue())

        y, u, t = 
start_sampling(ST,t1,t2,t3,t4,t5,t6,t7,t8,t9,t10,val1,val2,val3,val4,val5,val6,val7,val8,val9,val10)
        Experiment_Plot(t,u,y)
        os.chdir(str(origWD))


if __name__ == "__main__":
    app = wx.App(False)
    frame = console(None,-1,"Coupled DC motor Control Application")
    frame.Show()
    app.MainLoop()
```