Slovak University of Technology in Bratislava
Institute of Information Engineering, Automation, and Mathematics

PROCEEDINGS

17th International Conference on Process Control 2009

Hotel Baník, Štrbské Pleso, Slovakia, June 9 – 12, 2009

ISBN 978-80-227-3081-5

http://www.kirp.chtf.stuba.sk/pc09

Editors: M. Fikar and M. Kvasnica

Jadlovská, A., Dolinský, K., Lonščák, R.: Application of Designed Program Modules in C# Language for Simulation of Models of Dynamic Systems, Editors: Fikar, M., Kvasnica, M., In *Proceedings of the 17th International Conference on Process Control '09*, Štrbské Pleso, Slovakia, 534–547, 2009.

Full paper online: http://www.kirp.chtf.stuba.sk/pc09/data/abstracts/037.html

APPLICATION OF DESIGNED PROGRAM MODULES IN C# LANGUAGE FOR SIMULATION OF MODELS OF DYNAMIC SYSTEMS

A. Jadlovská, K. Dolinský, R. Lonščák

Department of Cybernetics and Artificial Inteligence, Faculty of Electrical Engineering, Technical University of Košice, Letná 9/A, 040 01 Košice
e-mail: Anna.Jadlovská@tuke.sk, kamil@plazma.sk, Richard.Lonscak@tuke.sk

Abstract: The purpose of this paper is to give a brief illustration of possibilities of programing language C# in field of modeling and classical control theory. We describe implementation of algorithms that perform transformation of continuous linear dynamic systems into their discrete equvivalents, continuous PID control into its discrete form and employ these algorithms in discrete closed loop control. We also show how implemented numerical methods can be used to solve systems of differential equations which are used in modeling of nonlinear dynamic systems. All required related functions are implemented and integrated into program modules which are used together in two similar applications designed to simulate control of linear and nonlinear dynamic system Ball & Plate thus verifying robustness of the PID controllers.

Keywords: Velocity algorithm, digital controller, Ball & Plate, C# in modeling

1 INTRODUCTION

Fields of modeling and control enjoy substantial attention. It's required however to know and master right algorithms, programming techniques, mathematic methods and control procedures to employ correctly in developed applications. This paper is dedicated to modeling and control of dynamic systems with employment of computers. Higher programming languages provide an excellent solution for solving problems that include simulation of control, identification and likewise. Some cases though require different approach. When main aim is speed or stability of application it's best to use languages as C or C++. There were a lot of new programming languages in last past years. Quite a lot of attention has been acquired by a one called C#. There are numerous publications encompassing algorithms written in C# for computer science and artificial intelligence. Yet publications that are trying to use this language for classic theory of control are very rare. Therefore we decided to contribute into this field with content of this paper and also demonstrate possibilities of this language.

2 MODELING OF DYNAMICAL SYSTEMS WITH PROGRAM MODULES CREATED IN C#

Before we can start with control we need to define the dynamic system in control structure. There are numerous methods for but basically we use systems of differential equations in some form. For this particular task it is wise to use a system of differential equations in state space as it's easy to use in iterative algorithms. In general we can describe SISO linear dynamic system with following equations:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}u(t) \tag{1}$$

$$y(t) = \boldsymbol{C}\boldsymbol{x}(t) + \boldsymbol{D}u(t) \tag{2}$$

where:

$\boldsymbol{x}(t)$ - vector of internal states,

$u(t)$ - system input,

$y(t)$ - system output,

$\boldsymbol{A}$, $\boldsymbol{B}$ are constant matrixes describing internal dynamics of system,

C, ***D*** are constant matrixes that describe how are outputs bound with internal states of system and initial conditions.

As it is impossible for a computer to process data in every instant we need to simplify our model and transform it into a discrete equivalent.

$$\begin{aligned} \boldsymbol{x}((i+1)\Delta t) &= \boldsymbol{F}\boldsymbol{x}(i\Delta t) + \boldsymbol{G}u(i\Delta t) \\ y(i\Delta t) &= \boldsymbol{H}\boldsymbol{x}(i\Delta t) + \boldsymbol{J}u(i\Delta t) \end{aligned} \quad (3)$$

F, ***G***, ***H***, ***J*** - matrixes describing the system in discrete state space.

Matrixes ***F***, ***G***, ***H***, and ***J*** are acquired by transformation of matrixes ***A***, ***B***, ***C***, ***D***, which characterize dynamic system in continuous state space.

$\boldsymbol{x}(i\Delta t)$ - vector of internal state variables in time $i\Delta t$

$\boldsymbol{x}((i+1)\Delta t)$ - vector of internal state variables in time $(i+1)\Delta t$

$u(i\Delta t)$ - input in time $i\Delta t$

$y(i\Delta t)$ - output in time $i\Delta t$

Such transformation is handled by *Continuous to Discrete system transformation module* (*C2D*). It's appropriate to analyze the system response to various inputs before we'll apply any control at all. If the case is SISO (Single Input Single Output) linear dynamic system we can easily apply *System Analysis module* (*SA*) which comes with intuitive user interface. After it is provided with matrixes ***F***, ***G***, ***H***, ***J*** it will compute and plot system response to choosen input. To control the system we are using continuous PID algorithm transformed to discrete equivalent a PSD control. This transformation is managed by *PID to PSD transformation module* (*PID2PSD*). Closed loop control of the system can be simulated using *Closed Loop Simulation module* (*CLS*). It's wise to verify robustness of the controller. This is possible to accomplish by several ways. One of them is causing a disturbance in control. If we ponder the fact that every real dynamic system is nonlinear and than if we have a nonlinear model of the system it's better to verify the robustness of the controller by applying it to the original nonlinear model which was simplified to linear by linearization. First of all as with linear dynamic system also nonlinear dynamic system needs to be defined. Mostly it is described by system of nonlinear differential equations. These systems are usually solved by numerical methods. For example by Runge-Kutta method which is used in *Nonlinear Differential System Solving module* (*NDSS*). In following chapters our attention will be focused on descprition of individual program modules than on a demonstration of these modules by an application which combines them together. We'll sum up its results and in the conclusion asset of this paper will be evaluated.

2.1 Continuous to discreete system transformation-module C2D

Module inputs:

- matrixes ***A***, ***B***, ***C***, ***D*** that are describing continuous linear dynamic system
- sampling period Δt

Module outputs:

- matrixes ***F***, ***G***, ***H***, ***J*** that are describing discrete linear dynamic system

2.1.1 Module analysis

Derivation of discrete state space is following as it is required for acquiring necessary formulas for transformation from discrete to continuous state space.

Let's assume we have a linear dynamic system described by (1) and (2):

Let's modify eqation (1):

$$e^{-\boldsymbol{A}t}\frac{d\boldsymbol{x}(t)}{dt} - e^{-\boldsymbol{A}t}\boldsymbol{A}\boldsymbol{x}(t) = e^{-\boldsymbol{A}t}\boldsymbol{B}u(t)$$

$$d(e^{-\boldsymbol{A}t}\boldsymbol{x}(t)) = e^{-\boldsymbol{A}t}\boldsymbol{B}u(t)dt$$

$$\int_{t_0}^{t} d(e^{-\boldsymbol{A}v}\boldsymbol{x}(v)) = \int_{t_0}^{t} e^{-\boldsymbol{A}v}\boldsymbol{B}u(v)dv$$

$$e^{-\boldsymbol{A}t}\boldsymbol{x}(t) - e^{-\boldsymbol{A}t_0}\boldsymbol{x}(t_0) = \int_{t_0}^{t} e^{-\boldsymbol{A}v}\boldsymbol{B}u(v)dv$$

$$e^{-\boldsymbol{A}t}\boldsymbol{x}(t) = e^{-\boldsymbol{A}t_0}\boldsymbol{x}(t_0) + \int_{t_0}^{t} e^{-\boldsymbol{A}v}\boldsymbol{B}u(v)dv$$

$$\boldsymbol{x}(t) = e^{\boldsymbol{A}(t-t_0)}\boldsymbol{x}(t_0) + \int_{t_0}^{t} e^{\boldsymbol{A}(t-v)}\boldsymbol{B}u(v)dv\,,$$

where $\boldsymbol{x}(t_0)$ is initial value of state vector. Under the condition that sampling is ekvidistand with sampling rate Δt, for solution in interval $v \in < i\Delta t, (i+1)\Delta t >$, $(t_0 = i\Delta t)$, while input input in this interval meets the conditon $u(v) = u(t_0)$.

And if we introduce a substitution:

$$t - v = \tau\,, -dv = d\tau$$

$$v = t_0 \rightarrow \tau = t - t_0\,, v = t \rightarrow \tau = 0$$

and if we're considering only discrete values of time,

$$t_0 = i\Delta t\,, t = (i+1)\Delta t\,,$$

where $i \in N_0 = N + \{0\}$.

Then we obtain a discrete state space description:

$$\boldsymbol{x}((i+1)\Delta t) = e^{\boldsymbol{A}\Delta t}\boldsymbol{x}(i\Delta t) + \int_{0}^{\Delta t} e^{\boldsymbol{A}\tau}d\tau\boldsymbol{B}u(i\Delta t)$$

where:

$$\boldsymbol{F} = e^{\boldsymbol{A}\Delta t} \tag{4}$$

$$\boldsymbol{G} = \int_0^{\Delta t} e^{\boldsymbol{A}\tau} d\tau \boldsymbol{B} \tag{5}$$

then:

$$\boldsymbol{x}((i+1)\Delta t) = \boldsymbol{F}\boldsymbol{x}(i\Delta t) + \boldsymbol{G}u(i\Delta t) \tag{6}$$

$$y(i\Delta t) = \boldsymbol{H}\boldsymbol{x}(i\Delta t) + \boldsymbol{J}u(i\Delta t)\,. \tag{7}$$

Since:

$$\boldsymbol{H} = \boldsymbol{C},\ \boldsymbol{J} = \boldsymbol{D}$$

It is possible to unroll function $e^{\boldsymbol{A}\Delta t}$ into Taylor's array:

$$e^{\boldsymbol{A}\Delta t} = \lim_{n\to\infty}\left(\boldsymbol{I} + \frac{\boldsymbol{A}}{1!}\Delta t + \frac{\boldsymbol{A}^2}{2!}\Delta t^2 + \frac{\boldsymbol{A}^3}{3!}\Delta t^3 + ... + \frac{\boldsymbol{A}^n}{n!}\Delta t^n \right) \tag{8}$$

where $\boldsymbol{I}$ reprezents unitarily matrix which has the same dimension as matrix $\boldsymbol{A}$.

After integration (5) we obtain according to (Krokavec 2006):

$$\boldsymbol{G} = [\boldsymbol{A}^{-1}e^{\boldsymbol{A}t}]_0^{\Delta t}\boldsymbol{B} = \boldsymbol{A}^{-1}(e^{\boldsymbol{A}\Delta t} - \boldsymbol{I})\boldsymbol{B} = \boldsymbol{A}^{-1}(\boldsymbol{F} - \boldsymbol{I})\boldsymbol{B}$$

$$\boldsymbol{G} = \boldsymbol{A}^{-1}(\boldsymbol{F} - \boldsymbol{I})\boldsymbol{B} = \boldsymbol{A}^{-1}\lim_{n\to\infty}\left(\boldsymbol{I} + \frac{\boldsymbol{A}}{1!}\Delta t + ... + \frac{\boldsymbol{A}^n}{n!}\Delta t^n - \boldsymbol{I} \right)\boldsymbol{B}$$

$$\boldsymbol{G} = \lim_{n\to\infty}\left(\boldsymbol{I}\frac{\Delta t}{1!} + \frac{\boldsymbol{A}}{2!}\Delta t^2 + \frac{\boldsymbol{A}^2}{3!}\Delta t^3 + .. + \frac{\boldsymbol{A}^n}{n!}\Delta t^n \right)\boldsymbol{B} \tag{9}$$

2.1.2 Module implementation

We can begin with equations (8) a (9) which are fundamental formulas for this module. Part of the program that handles transformation of the matrixes can be described in following steps.

Transformation of matrix $\boldsymbol{A}$ into matrix $\boldsymbol{F}$:

1. Initialization of variables $\boldsymbol{A}$, Δt, *e(0)*, ε from user interface (values are up to user), $i \leftarrow 0$.
2. $\boldsymbol{F} \leftarrow \boldsymbol{I}$
3. Saving $e(i-1)$ (error from previous step).
4. $\boldsymbol{A} \leftarrow \boldsymbol{A}^2$
5. Computation of factorial.
6. $\boldsymbol{F} \leftarrow \boldsymbol{F} + \frac{\boldsymbol{A}^i}{i!}\Delta t^i$
7. Computation of $e(i)$ for matrix $\boldsymbol{F}$.
8. Continuing with step 3 and $i \leftarrow i+1$, if condition $|e(i) - e(i-1)| < \varepsilon$ is not met. Otherwise end.

Transformation of matrix $\boldsymbol{B}$ into matrix $\boldsymbol{G}$:

1. Initialization of variables $\boldsymbol{A}$, $\boldsymbol{B}$, Δt, *e(0)*, ε from user interface (values are up to user), $i \leftarrow 0$.
2. $\boldsymbol{G} \leftarrow \boldsymbol{I}\frac{\Delta t}{1!}$
3. Saving $e(i-1)$.
4. $\boldsymbol{A} \leftarrow \boldsymbol{A}^2$
5. Computation of factorial.
6. $\boldsymbol{G} \leftarrow \boldsymbol{G} + \frac{\boldsymbol{A}^i}{i!}\Delta t^i$
7. Computation of $e(i)$ for matrix $\boldsymbol{G}$.
8. Continuing with step 3 and $i \leftarrow i+1$, if condition $|e(i) - e(i-1)| < \varepsilon$ is not met. Otherwise end.

Computation of $e(i)$ for matrix $\boldsymbol{F}$:

7.1. Initialization of variables F, Y, $r \leftarrow 0$, $i \leftarrow 0$.

7.2. $\boldsymbol{Y} \leftarrow \boldsymbol{F}^T$

7.3. $\boldsymbol{Y} \leftarrow \boldsymbol{Y}\boldsymbol{F}$.

7.4. $r \leftarrow r + \boldsymbol{Y}[i,i]$

7.5. $i \leftarrow i+1$

7.6. Continuing with step 4 until count of rows in matrix $\boldsymbol{Y}$ isn't reached.

7.7. $r \leftarrow \sqrt{r}$

$e(i)$ - error in step i, ε - accuracy

r - result, Y - instrumental variable

Computation of $e(i)$ for matrix $\boldsymbol{G}$ can be done by analogy.In following we'll be using shortened notation

$$i\Delta t = k\,,$$

$$(i+1)\Delta t = k+1\,,$$

etc. to simplify notation of eqations. And T_s for sampling period.

2.2 System Analysis module SA

Module inputs:

- matrixes $\boldsymbol{F}$, $\boldsymbol{G}$, $\boldsymbol{H}$, $\boldsymbol{J}$ describing discrete linear dynamic system
- sampling period T_s
- simulation time span $\langle t_0, t_f \rangle$
- vector of intput values $\boldsymbol{u}$

Module outputs:

- vector of output values $\boldsymbol{y}$

2.2.1 Module analysis

As we know from discrete systems theory values of state and output variables can be computed for next step in an iterative computation using eqations (3). We can analyze the system from graphical reprezentation of system response.

2.2.2 Module implementation

Part of the module that is dedicated to compute *y(k)* in an iterative computation can be described in following three steps.

1. Generation of input signal.
2. Computation based on eqation (3) of state vector $\boldsymbol{x}(k+1)$ and output $y(k)$ in cycle for each value of dicrete input signal.
3. Display of the result.

2.3 PID to PSD transformation module PID2PSD

Module inputs:

- parameters of PID controller K, T_I, T_d, μ
- sampling period T_s
- control input $u(k-1)$
- control constraint $u_{\max}$ ($u_{\min}$)
- control error $e(k)$ and $e(k-1)$

Module outputs:

- control input $u(k)$

2.3.1 Module analysis

We can start with eqation describing ideal PID controler:

$$u(t) = K[e(t) + \frac{1}{T_I}\int_0^t e(\tau)d\tau + T_d\frac{de(t)}{dt}] \qquad (10)$$

where:

K - proportional gain,

T_I - integral time constant,

T_d - derivational time constant,

$e(t)$ - control error,

$u(t)$ - control input.

Equation (10) can be transformed into discrete equivalent according to (Harsány 1998):

$$u(k) = K[e(k) + \frac{T_s}{T_I}(\frac{e(0)+e(k)}{2} + \sum_{i=1}^{k-1} e(k)) + \frac{T_d}{T_s}(e(k)-e(k-1))] \qquad (11)$$

where integration was substitued by trapezoid approximation and derivation by first difference. If we want (11) to copy (10) with satisfying precision we have to choose appropriate sampling period. Equation (11) represents so called positional algorithm for PSD controller. From efficiency point of view it's better to use velocity algorithm where increase of control intervention meets condition:

$$\Delta u(k) = u(k) - u(k-1) = q_0 e(k) + q_1 e(k-1) + q_2 e(k-2) \qquad (12)$$

where

$$q_0 = K(1 + \frac{T_s}{2T_I} + \frac{T_D}{T_s}) \qquad (13)$$

$$q_1 = -K(1 + 2\frac{T_D}{T_S} - \frac{T_S}{2T_I}) \qquad (14)$$

$$q_2 = K\frac{T_D}{T_s}. \qquad (15)$$

Hence control intervention can be computed using iterative formula:

$$u(k) = u(k-1) + \Delta u(k) \qquad (16)$$

so the velocity algorithm for PSD controller is:

$$u(k) = u(k-1) + q_0 e(k) + q_1 e(k-1) + q_2 e(k-2)$$

character of PID controler remains preserved if following conditions will be met:

$$q_0 > 0,\ q_1 < -q_0,\ -(q_0+q_1) < q_2 < q_0.$$

While we are stating only equations for PSD control. We can rewrite (16) into following form

$$u(k) = u_p(k) + u_i(k) + u_d(k), \qquad (17)$$

where

$$u_p(k) = (q_0 - q_2)e(k),$$

$$u_i(k) = u_i(k-1) + (q_0 + q_1 + q_2)e(k-1),$$

$$u_d(k) = q_2(e(k) - e(k-1)).$$

To prevent "wind-up effect", which occurs in control when control inputs are bounded to interval $< u_{\min}, u_{\max} >$, we will modify velocity algorithm in following way.

If $u(k) >= u_{\max}$ (or $u(k) >= u_{\min}$),

then

$$u_i(k) = u_i(k-1)$$

and $u(k) = u_{\max}$ (or $u(k) = u_{\min}$).

To prevent oscilation and sudden changes in control signal $u(k)$ we can use "derivative filter" which is

used in (Melichar 2008) and it is demonstrated on Fig. 1.

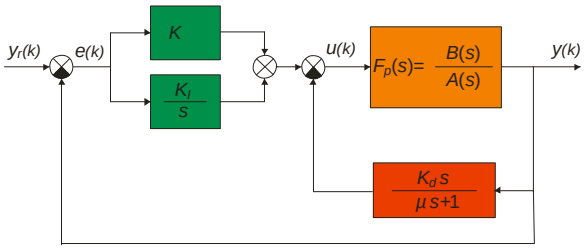

Fig. 1: Schema of derivative filter.

Where

$\mu \cong T_d/(3 \div 20)$,

K_I - integral gain,

K_D - derivational gain,

$F_p(s)$ - transfer function of controlled system.

Control rule for PID controller with derivative filter :

$$U(s) = \left(K + \frac{K_I}{s}\right)E(s) - \frac{K_D s}{\mu s + 1}Y(s).$$

After substitution $s = \frac{2(z-1)}{T_s(z+1)}$ we obtain control rule for PSD algorithm:

$$U(z) = \frac{\left[d_{0e} + d_{1e}z^{-1} + d_{2e}z^{-2}\right]E(z) + \left[d_{0y} + d_{1y}z^{-1} + d_{2y}z^{-2}\right]Y(z)}{c_0 + c_1 z^{-1} + c_2 z^{-2}}$$

in Z – transformation. This after backward transformation gives following recursive control rule:

$$\begin{aligned} u(k) = \frac{1}{c_0}(&-c_1 u(k-1) - c_2 u(k-2) + d_{0e}e(k) \\ &+ d_{1e}e(k-1) + d_{2e}e(k-2) + d_{0y}y(k) \\ &+ d_{1y}y(k-1) + d_{2y}y(k-2)) \end{aligned} \quad (18)$$

where

$$\begin{aligned} &c_0 = 4\mu + 2T_s,\ c_1 = -8\mu,\ c_2 = 4\mu - 2T_s, \\ &d_{0e} = 4K\mu + 2KT_s + 2K_I T_s \mu + K_I T_s^2, \\ &d_{1e} = -8K\mu + 2K_I T_s^2, \\ &d_{2e} = 4K\mu - 2KT_s - 2K_I T_s \mu + K_I T_s^2, \\ &d_{0y} = -4K_D,\ d_{1y} = 8K_D,\ d_{2y} = -4K_D \end{aligned} \quad (19)$$

are parameters of (18).

2.3.2 Module implementation

We can implement an algorithm which can be employed in discretization of PID controller to PSD form. Part of the module that handles this specific task can be summarized into next steps:

1. Acquisition of PID controler parameters K, T_I, T_D, sampling period T_s (and filter parameter μ).
2. Parameters q_0, q_1, q_2 computation based on (13), (14), (15) or computation of parameters for PSD with derivative filter based on (19) .

And using (17) we can compute control input $u(k)$. This part of module can be described in following steps:

1. Acquisition of control input $u(k-1)$ and control constraint u_{max} (u_{min}) and control error $e(k)$ and $e(k-1)$ from another module.
2. Computation of control input $u(k)$ based on (17).
3. If $u(k) >= u_{max}$ (or $u(k) >= u_{min}$),

   then

   $u_i(k) = u_i(k-1)$ and $u(k) = u_{max}$ (or $u(k) = u_{min}$).

Or if we are using PSD with derivative filter we can compute $u(k)$ using (18) with following implementation:

1. Acquisition of control inputs $u(k-1), u(k-2)$, control errors $e(k), e(k-1), e(k-2)$ and outputs $y(k), y(k-1), y(k-2)$ from another module.
2. Computation of control input $u(k)$ based on (18).

If we want to implement discretization of P, PI, PD controllers we can do it by analogy deriving from implementation of discretization of PID controler.

2.4 Closed Loop Simulation module CLS

Module inputs:

- matrixes ***A***, ***B***, ***C***, ***D*** that describes continuos linear dynamic system and sampling period T_s
- controller parameters K, T_I, T_d, μ
- vector of reference values $\boldsymbol{y}_r$
- simulation time span $\langle t_0, t_f \rangle$
- control input $u(k)$

Module outputs:

- vector of control inputs ***u***
- vector of control errors ***e***
- vector of output values ***y***

2.4.1 Module analysis

As we know from automatic control theory, some dynamic systems can be controlled by PID control algorithms. For successful PID control we need to obtain a set of parameters for these algorithms by

using convenient method of synthesis. The main elements of the control algorithm are represented by (6) and (7). Using these formulas we can compute values of state and output variables for each step in an iterative computation. Similarly using (17) or (18) we can obtain control intervention of the controller for each step *k*.

Schema of closed loop control structure is demonstrated on Fig. 2.

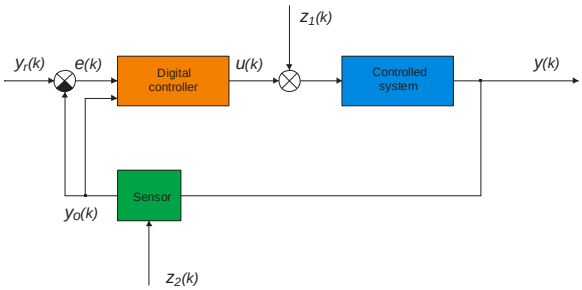

Fig. 2: Schema of closed loop control

$y_r(k)$ - desired value

$e(k)$ - control error

$u(k)$ - control input aquired using (17) or (18)

$z_1(k)$ - perturbation caused by outside world

$y(k)$ - system output which can't be measured it's aquired using (3)

$y_o(k)$ - observed system output

$z_2(k)$ - noise that is affecting the sensor

In this module we will employ *C2D* and *PID2PSD* which we described earlier. We can for example use Naslin or Graham-Lanthrop method for controler synthesis stated in (Madarász 2007), (Mikleš 1986).

2.4.2 Module implementation

Implementation of velocity algorithm, which is using selected methods of synthesis (Naslin and Graham-Lanthrop) and was verifyied on models of dynamic systems (simple mechanic oscillator and vagon set in (Dolinský 2008) in closed loop control using language C#, can be described in following steps.

1. Acquisition of system model parameters (matrixes ***A***, ***B***, ***C***, ***D*** and sampling period T_s).
2. Transformation of matrixes ***A***, ***B***, ***C***, ***D*** to their discrete equivalents ***F***, ***G***, ***H***, ***J***.
3. Acquisition of controller parameters (controller type and subsistent parameters).
4. Acquisition of reference, perturbation and noise signal parameters (description in user manual (Dolinský 2008).
5. Reference signal generation.
6. Perturbation signal generation (if was defined).
7. Noise signal generation (if was defined).
8. Recurent computation of response of controlled system to reference signal.

Implementation of recurent computation of response of controlled system to reference signal (step number 8) is described in following steps.

8.1. $k \leftarrow 0$

8.2. $e(k) \leftarrow y_r(k) - y_o(k)$

8.3. Control intervention computation according to (17) (or (18) if we are using derivative filter) using *PID2PSD* module.

8.4. Suming pertubation and control intervention (if perturbation was defined).

8.5. Computation of response of controlled system to control intervention with pertubation according to (3).

8.6. Observed controlled system output computation (value of controlled system output summed with noise).

8.7. Saving $e(k-1)$ (and if we are using derivative filter saving $u(k-1)$ and $y(k-1)$).

8.8. Saving $e(k)$ and $u(k)$ (and if we are using derivative filter $y(k)$).

8.9. If number of samples of reference signal is not exceeded continue with step 2 and $k \leftarrow k+1$. Otherwise end.

2.5 Nonlinear Differential System Solving module NDSS

Module inputs:

- starting conditions $\boldsymbol{f}(t_0, \boldsymbol{x}_0)$,
- vector of nonlinear functions $\boldsymbol{f}(t, \boldsymbol{x})$,
- sime span $\langle t_0, t_f \rangle$,
- initial step h_{min} and minimal step h.

Module outputs:

- vector of final values $\boldsymbol{f}(t_f)$

2.5.1 Module analysis

This module enables us to simulate nonlinear model of dynamic system described by systems of nonlinear differential equations.

$$\frac{dx_1}{dt} = f_1(t, x_1, ..., x_n)$$
$$\frac{dx_2}{dt} = f_2(t, x_1, ..., x_n)$$
$$\vdots$$
$$\frac{dx_n}{dt} = f_n(t, x_1, ..., x_n)$$

Module implements numerical method Runge– Kutta of 4th order. According to (Buša et al. 2006) this method is based on approximation expressed in following form.

$$x_{i,j+1} = x_{i,j} + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6} \qquad (20)$$

where:

$f_i(t)$ - i-th nonlinear function ($i = 1, 2, ..., n$)

$f_i(t_0)$ - initial value of i-th function

$f_i(t_f)$ - final value of i-th function

$x_{i,j}$ - value of i-th state variable in step j,

$x_{i,j+1}$ - value of i-th state variable in step $j+1$.

Values k_k are:

$$k_1 = h.f(t_i, x_{i,j})$$
$$k_2 = h.f(t_j + h/2, x_{i,j} + k_1/2) \qquad (21)$$
$$k_3 = h.f(t_j + h/2, x_{i,j} + k_2/2)$$
$$k_4 = h.f(t_j + h, x_{i,j} + k_3)$$

To check if the step is small enough we can use following.

$$|(k_2 - k_3)/(k_1 - k_2)| \qquad (22)$$

If value (22) is approximately equal to 0.05 we will preserve last used step. If (22) is considerable greater than 0.05 we will reduce step h .

2.5.2 Module implementation

1. Initialization of variables (initial time, final time, step, minimal step, initial contditons, collection of diferential equations).
2. If step doesn't meet the precision conditions and is greater than minimal step than we'll reduce step by half.
3. Computation of parameters k_k according to (21).
4. If step doesn't meet the precision conditions and is greater than minimal step than we'll continue with step 2 otherwise we'll continue with step 5.
5. Actual time is increased by step h .
6. Computation of values $x_{i,j+1}$ according to (20).
7. If actual time is lesser than final time we'll continue with step 2 otherwise end.

3 APPLICATION OF DESIGNED PROGRAM MODULES TO PHYSICAL SYSTEM BALL & PLATE

3.1 Application analysis

While implementing our application we have to respect that the real model is nonlinear and that it's to our benefit to simulate both linear and nonlinear model behaviour. Therefore it's required to create two applications that will integrate and control implemented program modules as different modules will be used for simulation and control in each case.

3.2 Analysis of dynamic system Ball & Plate

This model is represented by a ball rolling on a plate. This plate is controlled by a couple of step motors. Position of the ball is scaned by a camera. Images are analysed by a computer which will determine the ball location and consecutively will determine and send out appropriate voltage to step motors. These motors will lean the plate to desired angle and force the ball to move in appropriate direction so the desired results would be achieved.

Schema from (Humusoft: *CE151 Ball and Plate Apparatus – Educational Manual* 1996 - 2004) is shown on Fig. 3, block schema is on Fig. 4.

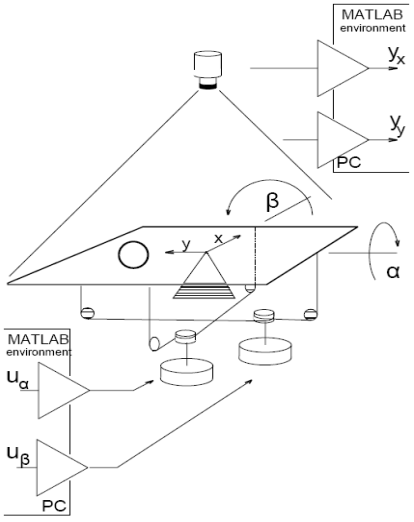

Fig. 3: Schema of Ball & Plate and apparatus

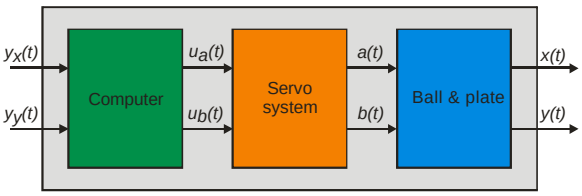

Fig. 4: Block schema of Ball & Plate

Variables description:

$y_x(t)$, $y_y(t)$ - ball location on plate determined from acquired image [m, m]

$u_a(t)$, $u_b(t)$ - voltages controlling individual steping motors [V, V]

$a(t)$, $b(t)$ - angles representing inclination of plate [rad, rad]

$x(t)$, $y(t)$ - ball real position [m, m]

As there is no feedback between ball position and plate inclination it's possible to divide the model into two separate parts servomechanism and ball freely rolling on the plate. Matematical model without servomechanism was derived in (Humusoft: *CE151 Ball and Plate Apparatus – Educational Manual* 1996 - 2004) from basic Euler - Lagrange equation.

$$\frac{d}{dt}\frac{\partial T}{\partial \dot{q}_i} - \frac{\partial T}{\partial q_i} + \frac{\partial V}{\partial q_i} = Q_i$$

where

q_i - the i-th generalized coordinate

$\dot{q}_i$ - the first derivation of the i-th generalized coordinate by time

T - kinetic energy of the system

V - potential energy of the system

Q_i - the i-th generalized force

Detailed derivation is in (Humusoft: *CE151 Ball and Plate Apparatus – Educational Manual* 1996 - 2004).

Finally we get a system of four differential equations of the second grade.

$$x:\ (m+\frac{I_b}{r_b^2})\ddot{x} - m(\dot{\alpha}\dot{\beta}y + \dot{\alpha}^2 x) + mg\sin\alpha = 0 \qquad (23)$$

$$y:\ (m+\frac{I_b}{r_b^2})\ddot{y} - m(\dot{\alpha}\dot{\beta}x + \dot{\beta}^2 y) + mg\sin\beta = 0 \qquad (24)$$

$$(I_p + I_b + mx^2)\ddot{\alpha} + m(\ddot{\beta}xy + \dot{\beta}\dot{x}y + \dot{\beta}x\dot{y} + 2\dot{\alpha}\dot{x}x) + mgx\cos\alpha = F_\alpha d\cos\alpha \qquad (25)$$

$$(I_p + I_b + mx^2)\ddot{\beta} + m(\ddot{\alpha}xy + \dot{\alpha}\dot{x}y + \dot{\alpha}x\dot{y} + 2\dot{\beta}\dot{x}x) + mgx\cos\beta = F_\beta d\cos\beta \qquad (26)$$

where

x, y - ball coordinates on the plate [m]

r_b - ball radius [m]

ω - vector of ball angular velocity [rad/s]

α, β - angles of plate inclination [rad]

I_b - ball inertia [kg.m^2]

I_p - plate inertia [kg.m^2]

m - ball mass [kg]

g - gravitional acceleration [ms^{-2}]

F_α - force influencing the plate in the direction of axis x [N]

F_β -force influencing the plate in the direction of axis y [N]

Ball motion is described by (23) and (24) which describe dependence of ball acceleration from angle and angular velocity of the plate inclination. Eqations (25) and (26) are describing how the plate inclination dynamics is influenced by the external driving force and the position and speed of the ball. According to the B&P manual it's possible to simplify the dynamics of B&P. Using the assumptions from manual we'll finally obtain following model.

$$\frac{d^2x}{dt^2} = \frac{5}{7}g\sin\alpha \approx K_b\alpha$$

$$\frac{d^2y}{dt^2} = \frac{5}{7}g\sin\beta \approx K_b\beta$$

3.2.1 *Servo system*

Block schema of the servo system from (Humusoft: *CE151 Ball and Plate Apparatus – Educational Manual* 1996 - 2004) is on Fig. 5. Due to limitations of the system this part contains several nonlinear components.

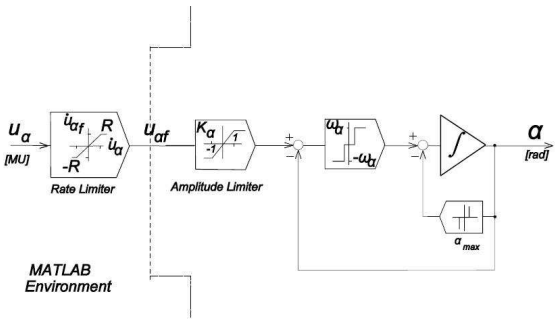

Fig. 5: Block schema of servo system

The first nonlinear component is a filter called rate limiter which is restricting the scope of speed of change (derivation) of input voltage. This filter solves the problem of software driver. The problem dwells in following. Stack in which we store wanted value can be actualized only after reaching the desired position. In other words we cannot exceed nominal speed of stepping motors which is determined by frequency of impulses which are supplyied by driving card of

stepping motors. We can expess it matematically in following statements.

$$rate = \frac{u_\alpha(k) - u_\alpha(k-1)}{T_s}$$

$$u_{\alpha f}(k) = +T_s R + u_{\alpha f}(k-1) \qquad \text{rate} < +R$$

$$u_{\alpha f}(k) = -T_s R + u_{\alpha f}(k-1) \qquad \text{rate} < -R$$

$$u_{\alpha f}(k) = u_\alpha \qquad \text{otherwise.}$$

where

u_α - filter input

$u_{\alpha f}$ - filter output

k - step

Δt - sampling period

R - rate treshold

Additional nonlinear dynamics are caused by a saturation filter called amplitude limiter which ensures that maximum slope of the plate cannot be exceeded. Input signal is limited into interval <-1,1> so after multiplying it by static gain it fits into plate inclination limits. Finally we have to respect the fact that the stepping motors have constant speed of stepping. Hence we need to add an element that will be insensitive to certain range of values and beyond this interval output will be a positive or negative constant value thus modeling the motion of the motors upwards or downwards.

System parameters

Parameters of dynamic system can be measured directly or are known from (Humusoft: *CE151 Ball and Plate Apparatus – Educational Manual* 1996 - 2004).

Normalized parameters are following.

K	overall system gain	[s $^{-2}$]
ω	nominal speed of servo system	[s $^{-1}$]
T_m	time constant of servosystem	[s]
K_α	static gain	[rad/MU]
K_b	B&P system gain	[ms^{-2}/rad]
K_x	ball position sensor constant	[MU/ m]

Linear model

State vector:

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y_x \\ \dot{y}_x \\ \bar{\alpha} \end{bmatrix} \quad \begin{bmatrix} ball's\ position & [-] \\ ball's\ velocity & [s^{-1}] \\ plate's\ inclination & [-] \end{bmatrix}$$

Input u_α ... desired plate angle sent out from Matlab $u_\alpha \in <-1,+1>$

Output y_x ... ball position read to Matlab $y_x \in <-1,+1>$

State equations

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\ \boldsymbol{x} + \boldsymbol{B}\ u_\alpha$$

$$\dot{\boldsymbol{x}} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & K \\ 0 & 0 & -\frac{1}{T_m} \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 0 \\ 0 \\ \frac{1}{T_m} \end{bmatrix} u_\alpha \qquad (27)$$

$$y_x = \boldsymbol{C}\ \boldsymbol{x} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \boldsymbol{x} \qquad (28)$$

Linear model transfer function:

$$F(s) = \frac{Y_x}{U_\alpha} = \frac{K_\alpha K_\beta K_x}{s^2(T_m s + 1)} = \frac{K}{s^2(T_m s + 1)} \quad . \qquad (29)$$

System characteristics

The B&P model is a 3rd order system with the 2nd order of astatism

Characteristic nonlinearities:

- rate limiter

- saturation

- time constant dependent on the magnitude and frequency of an input signal

unmodelled properties :

- friction

- defects in the ball and/or plate surface

3.3 Linear Ball & Plate Control Simulation module LBPC

Module inputs:

- PID parameters K, T_d, T_i, μ
- sampling period T_s ,
- simulation time span $\langle t_0, t_f \rangle$,
- noise and perturbation parameters.

Module outputs:

- vector of control input for x and y axis $\boldsymbol{u}_x$, $\boldsymbol{u}_y$,
- vector of position for x and y axis $\boldsymbol{y}_x$, $\boldsymbol{y}_y$,
- vector of observed position for x and y axis $\boldsymbol{y}_{ox}$, $\boldsymbol{y}_{oy}$,
- vector of control error for x and y axis $\boldsymbol{e}_x$, $\boldsymbol{e}_y$.

3.3.1 Module analysis

When we are assuming linear model of Ball & Plate we can compute system states and outputs using (27) and (28). Although Ball & Plate is a MIMO (Multi Input Multi Output) dynamic system it's possible to divide it into two SISO dynamic systems (ball motion on the plate in direction of axis x and motion in direc-

tion of axis *y*). Therefore we'll conduct analysis only for one coordinate (rules for the second cooridate are the same).

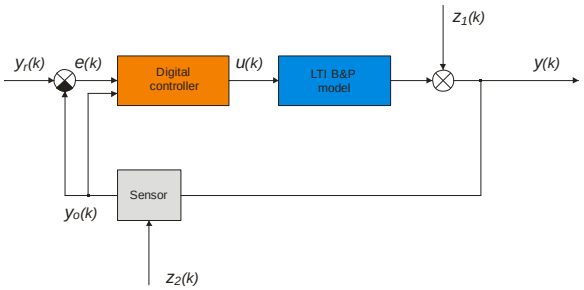

Fig. 6: Cloosed loop control of linear Ball & Plate model

$y_r(k)$ - control input(reference trajectory for one axis)

$e(k)$ - control error

$u(k)$ - control input which is computed from (17) or (18)(voltage supplied to stepping motor which controls inclination of plate in direction of one axis)

$z_1(k)$ - perturbation that affects ball position

$y(k)$ - output (ball position) which is computed from (3)

$y_o(k)$ - observed output (ball position influenced by noise)

$z_2(k)$ - noise affecting the sensor

It might appear that perturbation and noise are the same and can be considered as one signal but there is difference in character of those signals. Character of perturbation signal is deterministic while character of noise is pseudorandom although its amplitude is limited. We can use perturbation to simulate jumping and sliding of the ball on the plate as those phenomena are present in real model. Noise can be used to simulate behaviour of camera and imperfections of image recognition algorithm. Synthesis of control is based on transfer function (29).

PID controller designed by Naslin method has following parameters $K = 0.125$, $T_d = 4$, $T_i = 8$.

Corresponding parameters of PSD controller when sampling period is $T_s = 0.1$ are:

$$q_0 = 5.1257,\ q_1 = -10.124,\ q_2 = 5\,.$$

Control rule for PSD controller is following:

$$u(k) = u(k-1) + 5.1257e(k) - 10.124e(k-1) + 5e(k-2)\,.$$

PID controller designed by Graham-Lanthrop method has following parameters $K = 1.743$, $T_d = 0.494$, $T_i = 1.098$.

Corresponding parameters of PSD controller when sampling period is $T_s = 0.04$ are:

$$q_0 = 23.3,\ q_1 = -44.76,\ q_2 = 21.52.$$

Control rule for PSD controller is following:

$$u(k) = u(k-1) + 23.3e(k) - 44.76e(k-1) + 21.52e(k-2)\,.$$

3.4 *Module implementation*

To simulate closed loop circuit for both axes we can use module *CLS* which was described earlier. But we have to modify the implementation of recursive computation of response of controlled system to following form.

7.1. $k \leftarrow 0$

7.2. $e(k) \leftarrow y_r(k) - y_o(k)$

7.3. Control intervention computation according to (17) (or (18) if we are using derivative filter) using *PID2PSD* module.

7.4. Computation of state vector according to the first equation of (3),

7.5. Summing perturbation and ball position (if perturbation was defined).

7.6. Limitation of the ball position.

7.7. Computation of response of the system according to second equation in (3).

7.8. Computation of observed output (value of output summed with noise),

7.9. Saving $e(k-1)$ (and if we are using derivative filter saving $u(k-1)$ and $y(k-1)$).

7.10. Saving $e(k)$ and $u(k)$ (and if we are using derivative filter $y_o(k)$).

7.11. If number of samples of reference signal is not exceeded continue with step 2 and $k \leftarrow k+1$. Otherwise end.

3.5 *Nonlinear B&P Control Simulation Module NBPCSM*

Module inputs:

- PID parameters K, T_d, T_i, μ,
- Sampling period T_s ,
- Simulation time span $\langle t_0, t_f \rangle$,
- noise and perturbation parameters

Module outputs:

- vector of control input for *x* and *y* axis $\boldsymbol{u}_x$, $\boldsymbol{u}_y$,
- vector of position for *x* and *y* axis $\boldsymbol{y}_x$, $\boldsymbol{y}_y$,
- vector of observed position for *x* and *y* axis $\boldsymbol{y}_{ox}$, $\boldsymbol{y}_{oy}$,
- vector of control error for *x* and *y* axis $\boldsymbol{e}_x$, $\boldsymbol{e}_y$.

3.5.1 *Module analysis*

When we are assuming nonlinear model of Ball & Plate we have to compute system states and outputs using (23) a (24) which are describing behaviour of the nonlinear dynamic system.

As not the forces F_α and F_β but directly the angles α and β are system inputs. This is due to the fact that the frequency of a stepper is below the acceleration limit. No steps can be lost and the magnitude of load moment cannot affect the motor position. This assumption results in omitting the equations (25) and (26) as stated in (Humusoft: *CE151 Ball and Plate Apparatus – Educational Manual* 1996 - 2004). But of course we still have to respect limits of dynamic system.

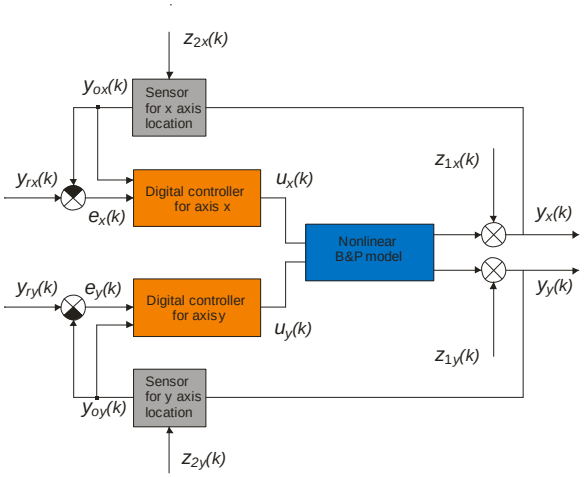

Fig. 7: Cloosed loop control of nonliear Ball & Plate

$y_{rx}(k)$ - control input (reference trajectory for axis x)

$e_x(k)$ - control error for axis x

$u_x(k)$ - action input which is computed from (17) or (18) (voltage supplied to stepping motor which controls inclination of plate in direction of x axis)

$z_{1x}(k)$ - perturbation that affects ball position

$y_x(k)$ - output for axis x (x coordinate of ball position) which is computed from (3)

$y_{ox}(k)$ - observed output for axis x (x coordinate of ball position influenced by noise)

$z_{2x}(k)$ - noise affecting the sensor in coordinate x

Description for y axis variables can be done by analogy. To control nonlinear dynamic system Ball & Plate we used controllers designed using transfer function of linear model of Ball & Plate.

3.5.2 *Module implementation*

1. Acquisition of controller parameters (controller type and subsistent parameters).
2. Acquisition of reference, perturbation and noise signal parameters and sample time.
3. Reference signal generation.
4. Perturbation signal generation (if was defined).
5. Noise signal generation (if was defined).
6. Recurent computation of response of controlled system to reference signal.

Implementation of recurent computation of response of controlled system to reference input (step number 6) is described in following steps.

6.1. $k \leftarrow 0$

6.2. $e_x(k) \leftarrow y_{rx}(k) - y_{ox}(k)$
$e_y(k) \leftarrow y_{ry}(k) - y_{oy}(k)$

6.3. Computation and rate and amplitude limitation of voltages for each motor.

6.4. Computation of plate inclination angles, their limitation, computation of difference between present and last value and its filtration (based on sensitivity).

6.5. Numerical integration of angles.

6.6. Computation of ball position and derivation of ball position.

6.7. Summing perturbation and ball position (if perturbation was defined).

6.8. Computation of ball position affected with noise and limitation of ball location.

6.9. Saving $e_x(k-1)$, $e_y(k-1)$, $e_x(k)$, $e_y(k)$, $u_x(k)$, $u_y(k)$ (if we are using derivative filter saving also $y_{ox}(k)$, $y_{oy}(k)$, $y_{ox}(k-1)$, $y_{oy}(k-1)$, $u_x(k-1)$, $u_y(k-1)$) and plate inclination ($\alpha(k)$, $\beta(k)$).

6.10. If number of samples of reference input is not exceeded continue with step 2 and $k \leftarrow k+1$. Otherwise end.

3.6 *Application results*

For simulation of control of the dynamic system Ball & Plate we used following signals. As a controlled input we used position of the ball and as a control input we used ball desired positon or desired trajectory. Results depend on employed controller, desired trajectory, time span provided to cover the trajectory, sampling period, perturbations and noise. As a perturbation signal we use a signal that affects ball positon. Thus we can approximate real conditions where ball in certain moments looses contact with the plate or is sliding. Noise in this system is considerable as image processing by camera determining the location

of the ball is affected by light conditions and ball color.

In general results of control of linear model are better. Following pictures illustrate results of linear and nonlinear model. Simulation time was 60 seconds for sqare and circle trajectory, 120 seconds for star and helix and 30 seconds for position. For control of linear model we used controller designed by Graham-Lanthrop method (K = 1.743, T_d = 0.494, T_i = 1.098) with derivative filter ($\mu = \frac{T_d}{10}$), as we acquired better results. For control of nonlinear model we used controller designed by Naslin method (K = 0.125, T_d = 4, T_i = 8) without derivative filter.

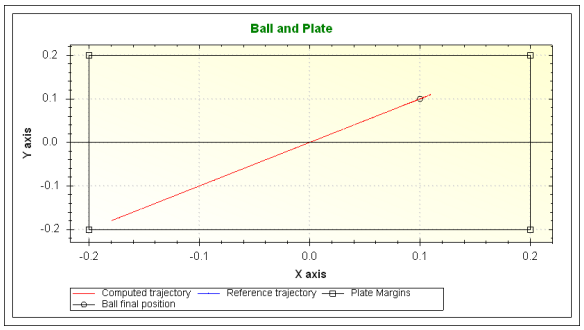

Fig. 8 : Control to desired position (linear model)

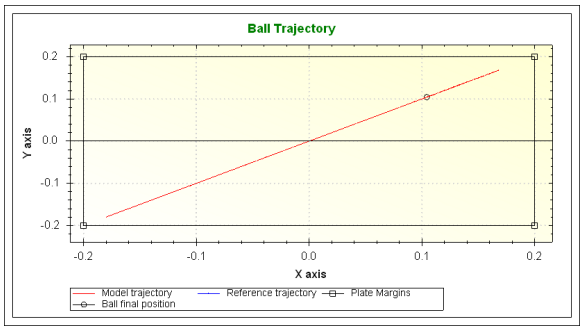

Fig. 9: Control to desired position (nonlinear model)

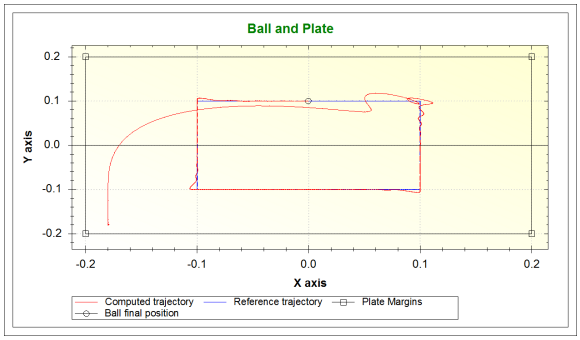

Fig. 10: Tracking of sqare trajectory (linear model)

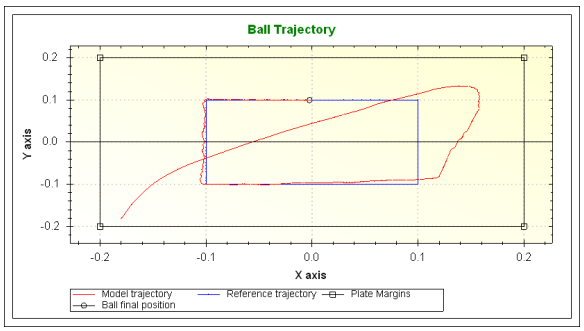

Fig. 11: Tracking of sqare trajectory (nonlinear model)

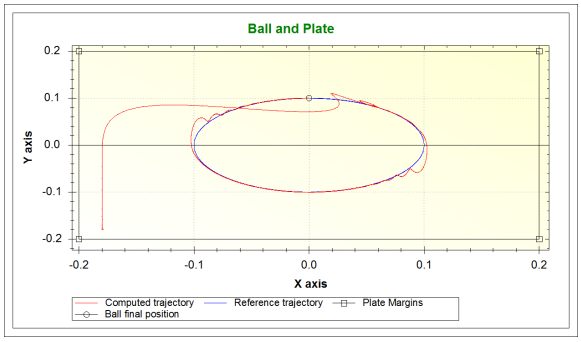

Fig. 12: Tracking of circle trajectory (linear model)

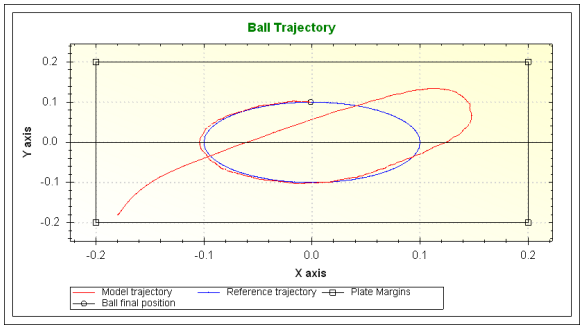

Fig. 13: Tracking of circle trajectory (nonlinear model)

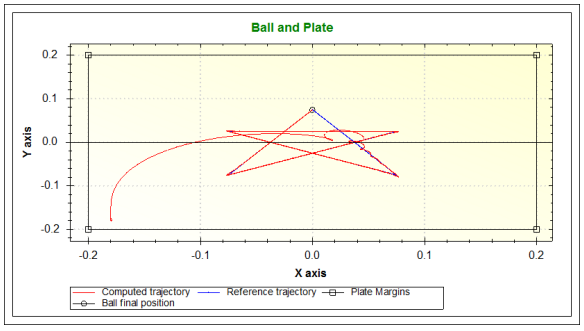

Fig. 14: Tracking of star trajectory (linear model)

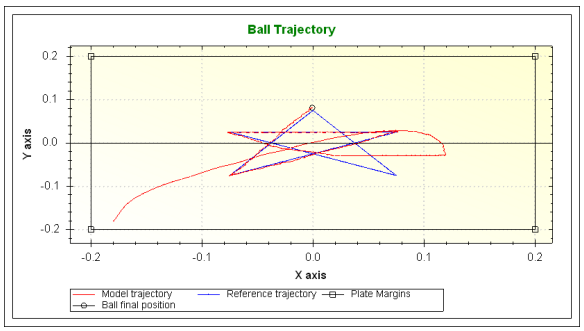

Fig. 15: Tracking of star trajectory (nonlinear model)

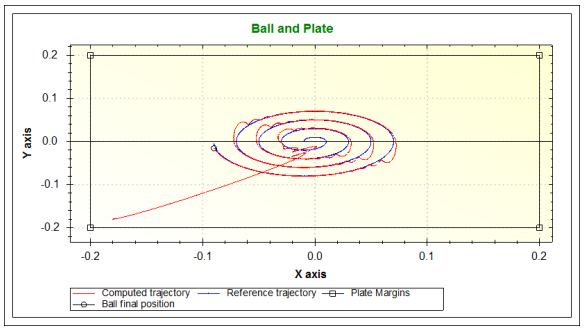

Fig. 16: Tracking of helix trajectory (linear model)

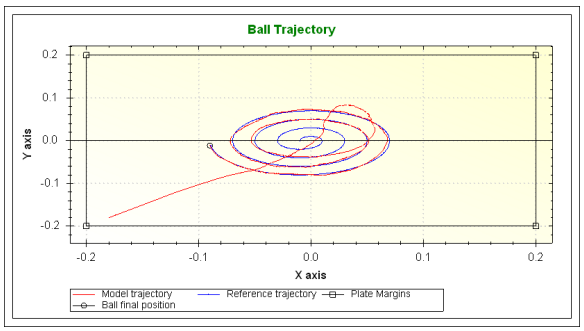

Fig. 17: Tracking of helix trajectory (nonlinear model)

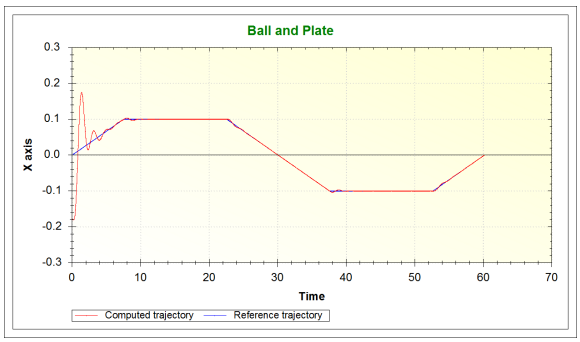

Fig. 18: Tracking of sqare trajectory (axis x) without filter.

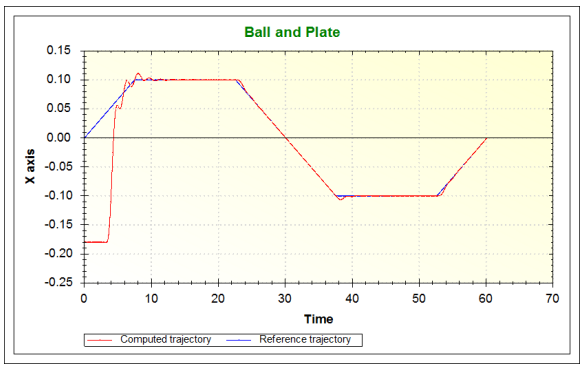

Fig. 19: Tracking of sqare trajectory (axis x) with filter.

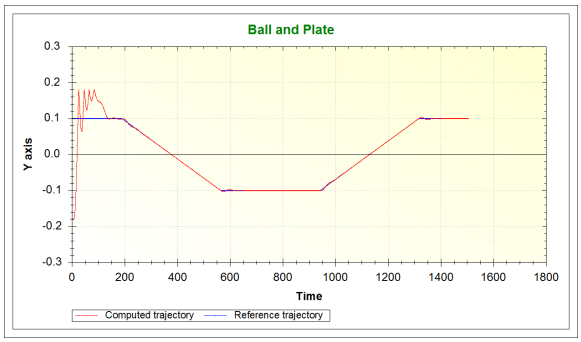

Fig. 20: Tracking of sqare trajectory (axis y) without filter.

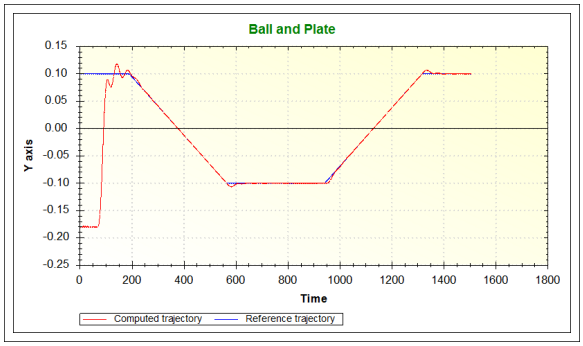

Fig. 21: Tracking of sqare trajectory (axis y) with filter.

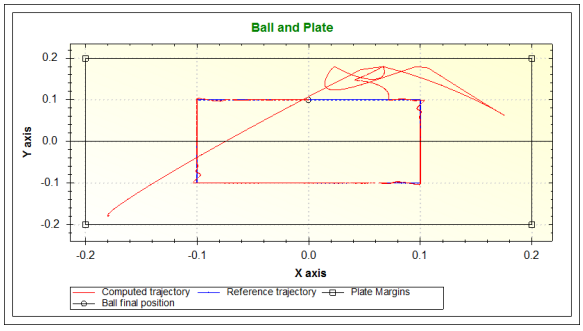

Fig. 22: Tracking with sqare trajectory without filter.

Fig. 10 can be compared with Fig. 22 to demonstrate effect of derivative filter when reference trajectory is changing very steeply.

4 CONCLUSION

We created and verified required modules that allow us to transform continuous linear dynamic system to their discrete equivalent and analyze behaviour of these dynamic systems. Then we continued with transformation of PID algorithm to its discrete form. We used PSD algorithm with anti-windup and derivative filter in simulation of control in closed loop circuit. As we wanted to evaluate robustness of control we created a program module that enables us to solve nonlinear systems of differential equations. All these modules were used in an application that simulates behaviour of real physical model Ball and Plate. We created two applications one for simulation of control of linear model and second for nonlinear model of dynamic system Ball and Plate. Thus we verified the robustness of designed control. Also we showed that while C# doesn't directly provide functions or procedures for modeling or control of dynamic system, we can create them by ourselves. Considering the fact that syntax of C# is very easy and thorough we can quickly implement required functions into program modules which can be used in an application in desired way. Also we are not bound to higher programming languages as Matlab. Moreover it's possible to use rich possibilities that platform .NET provides.

5 ACKNOWLEDGMENTS

This research has been supported by the Scientific Grant Agency of Slovak republic under project Vega No. 1/0617/08 Multiagent Network Control Systems with Automatic Reconfiguration. This support is very gratefully acknowledged.

6 REFERENCES

J. Buša, V. Pirč, Š. Schrotter. (2006). *Numerické metódy, pravdepodobnosť a matematická štatistika*. Košice: TU-FEI,. 166 s. ISBN 80-8073-632-4

K Dolinský. (2008). *Design and Realization of Program Modules for Models of Dynamical Systems*. Bachelor Thesis. (supervisors: assos. prof. Ing. A. Jadlovská, PhD, Ing. R. Lonščák), FEI TU, Košice, (in Slovak).

L. Harsány, J. Murgaš, D. Rosinová, A. Kozáková. (1998). *Teória automatického riadenia*. Bratislava: Slovenská Technická Univerzita Bratislava, Fakulta elektrotechniky a informatiky. 216s. ISBN 80-227-1098-9

Humusoft: *CE151 Ball and Plate Apparatus – Educational Manual*. (1996-2004).

D. Krokavec, A. Filasová. (2006). *Diskrétne systémy*. Košice: Elfa. 302 s. ISBN 80-8086-028-9

L. Madarász, M. Bučko, L. Fozo. (2007). *Základy automatického riadenia* - 1. Elfa. Košice. ISBN 978-80-8086-042-4

J. Mikleš, V. Hutla. (1986) *Teória automatického riadenia*. Alfa. Bratislava. ISBN 63-576-86

J.Melichar. (2008) *Lineární Systémy 2 (Učební text)*. ZČU Plzeň (available on inernet).