

Zajištění kompatibility knihovny mnoharozměrových samočinně se nastavujících regulátorů se systémem Real – Time Workshop

Václav Zajíček

Diplomová práce
2005

Univerzita Tomáše Bati ve Zlíně
Fakulta technologická

Institut řízení procesů a aplikované informatiky

Tímto děkuji svému vedoucímu Ing. Marku Kubalčíkovi, Ph.D. za odbornou pomoc a příkladné vedení při řešení diplomové práce.

Děkuji také Ing. Petru Chalupovi, Ph.D. za vysvětlení základních aspektů programování pro zvolené technologické prostředí.

Souhlasím s tím, že s výsledky mé práce může být naloženo podle uvážení vedoucího diplomové práce, ředitele ústavu a institutu. V případě publikace budu uveden jako spoluautor.

Prohlašuji, že jsem na celé diplomové práci pracoval samostatně a použitou literaturu jsem citoval.

Ve Zlíně, 15. 06. 2005

.....

podpis diplomanta

RESUMÉ

V prostředí Matlab – Simulink je vytvořena knihovna samočinně se nastavujících regulátorů. Tato knihovna umožňuje počítačovou podporu návrhu a ověřování diskrétních samočinně se nastavujících regulátorů pro systémy se dvěma vstupy a dvěma výstupy. Implementované regulátory mohou být ověřeny jak simulačně, tak řízením reálných laboratorních modelů. Hlavním cílem práce bylo zabezpečení kompatibility knihovny se systémem Real – Time Workshop. Jednotlivé části regulátorů realizované jako s-funkce psané v kódu Matlabu bylo nutno naprogramovat v C jazyce. Takto upravenou knihovnu lze zkompilovat a spustit na libovolné cílové platformě. Součástí práce je rovněž ověření funkčnosti knihovny řízením laboratorního modelu.

SUMMARY

A self – tuning controllers' library was designed under Matlab – Simulink environment. The library enables design and verification of discrete self – tuning controllers for two inputs – two outputs systems. The implemented controllers can be verified both by simulation and real – time control of laboratory models. The main aim of the work is to make the library compatible with the Real – Time Workshop System. Particular parts of the implemented controllers are implemented as s – functions written in the programming language of the Matlab environment. It was necessary to create these s-functions in the C programming language. This modification enables generating source codes and programs to be used outside the Matlab environment. The work contains also experimental results achieved during verification of the library by control of a laboratory model.

OBSAH

ÚVOD	6
1 TEORETICKÁ ČÁST	7
1.1 POUŽITÉ ALGORITMY	7
1.1.1 Použité modely řízených procesů	7
1.1.2 Návrh algoritmu regulátoru 1DOF	9
1.1.3 Návrh algoritmu regulátoru 2DOF – A	11
1.1.4 Návrh algoritmu regulátoru 2DOF – B	13
1.1.5 Autonomní řízení s využitím kompenzátorů	14
1.1.6 Rekurzivní identifikace	18
1.1.7 Modifikace průběžné identifikace pro regulátory bez kompenzátorů	20
1.1.8 Modifikace průběžné identifikace pro regulátory s korekčními členy	21
1.2 SIMULACE V PROSTŘEDÍ MATLAB – SIMULINK	22
1.2.1 Tvorba aplikací pomocí programu Real – Time Workshop	22
1.2.2 Generování zdrojových kódů	23
1.2.3 Slučitelnost schémat Simulink	23
1.3 S-FUNKCE	24
1.3.1 Přičlenění s-funkce k modelu	24
1.3.2 Jak pracuje s-funkce	25
1.3.3 Zavádění s-funkcí	26
1.3.4 Koncepce s-funkce	27
1.4 S-FUNKCE PSANÉ V JAZYCE C	28
1.4.1 Povinné definice v souboru s-funkce	29
1.4.2 Zpětně volané metody s-funkce	30
1.4.3 Makra datové struktury SimStruct	32
1.5 PŘEHLED REGULÁTORŮ A PŘEKLÁDANÝCH SOUBORŮ	40
2 PRAKTICKÁ ČÁST	41
2.1 TECHNOLOGICKÁ KARTA DS1104	41
2.1.1 Instalace	42
2.2 LABORATORNÍ MODEL SPŘAŽENÝCH ELEKTROMOTORŮ	43
2.3 REALIZACE MODELŮ	44
2.3.1 Vytvoření a spuštění aplikace	44
2.4 PŘIPOJENÍ VNĚJŠÍCH ZAŘÍZENÍ	45
2.5 SIMULAČNÍ OVĚŘENÍ	47
2.6 ŘEŠENÍ SYSTÉMU LINEÁRNÍCH ROVNIC MATICOVOU INVERZÍ	51
2.7 ŘÍZENÍ LABORATORNÍHO MODELU V REÁLNÉM ČASE	52
2.7.1 Snímání měřených dat pomocí ControlDesku	52
2.7.2 Použití počátečních odhadů modelu bez apriorní informace	52
2.7.3 Použití ustálených parametrů z předchozích experimentů	57
ZÁVĚR	62
SEZNAM POUŽITÉ LITERATURY	63
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK	64
SEZNAM OBRÁZKŮ	65
SEZNAM TABULEK	66
SEZNAM PŘÍLOH	67

ÚVOD

Oblast adaptivního řízení procházela v uplynulých letech rozsáhlým rozvojem. Cílem tohoto přístupu bylo řešit problematiku návrhu regulátorů v případech, kdy charakteristiky řízeného procesu nejsou dostatečně známy, nebo se v jeho průběhu mění. Postupně vznikly různé směry řešení této problematiky. Jedním z úspěšných a perspektivních směrů jsou tzv. samočinně se nastavující regulátory (STC – Self-Tuning Controllers).

K usnadnění návrhů i simulací byla v prostředí Matlab – Simulink vytvořena knihovna těchto regulátorů. Všechny funkce regulátorů jsou napsány v jazyce prostředí Matlab a uloženy v tzv. m-souborech. Po úspěšných simulacích a laboratorním ověření lze uvažovat o průmyslovém začlenění regulátoru. Průmyslové procesy jsou často řízeny pomocí programovatelných logických automatů nebo průmyslových počítačů. Součástí Matlabu je program Real – Time Workshop, který umožňuje vytvářet kód spustitelný na těchto průmyslových zařízeních. Výsledný kód v tomto případě již tedy nepotřebuje ke své činnosti Matlab ani osobní počítač. Generování kódu je řízeno překladovými soubory, které popisují převod schémat programu Simulink do cílového programovacího jazyka. Různé cílové systémy vyžadují různé překladové soubory.

Všechny bloky Simulinku ale nejsou podporovány běžně dodávanými překladovými soubory. V případě knihovny regulátorů se jedná o bloky Matlab Fcn a S-Function. K zajištění kompatibility je třeba bloky Matlab Fcn nahradit s-funkcemi stejných jmen a poté je přeložit do jazyka C. U bloků S-Function postačí převést současný kód do jazyka C. Funkce regulátorů a jejich vnitřní struktura zůstává stejná, mění se jen způsob programování.

Výsledná aplikace byla úspěšně ověřena na řídicím systému dSPACE, ke kterému byl připojen laboratorní model spřažených elektromotorů.

1 TEORETICKÁ ČÁST

Základní myšlenka STC je založena na průběžné (rekurzivní) identifikaci vhodného modelu řízeného procesu s následnou syntézou regulátoru [1, 4].

Prakticky nejlépe využitelných výsledků bylo dosaženo především při řízení jednorozměrových systémů, pro které byla navržena celá řada numericky stabilních a různě složitých algoritmů. Tyto algoritmy je pak možno implementovat na řídicí počítače, které jsou vybaveny vhodnou jednotkou pro styk s technologickým prostředím. Rozšíření na vícerozměrové systémy nečiní v mnoha případech zásadnější problémy.

1.1 Použité algoritmy

Již vytvořená knihovna [3] může být rozšířena pro systémy s libovolným počtem vstupů a výstupů, primárně je však zaměřena na systémy se dvěma vstupy a dvěma výstupy. V praxi existuje celá řada procesů, které mají charakter systémů se dvěma vstupy a dvěma výstupy. Na institutu je rovněž k dispozici řada laboratorních modelů, které takové průmyslové procesy simulují (model spřažených elektromotorů, model vrtulníku, laboratorní teplovzdušný model, soustava spojených nádrží).

Všechny regulátory začleněné do knihovny jsou založeny na obecném lineárním modelu ve formě maticového zlomku. Koeficienty tohoto modelu jsou potom v rámci adaptivního řízení identifikovány ze vstupních a výstupních dat. Jedná se tedy o model chování, který nemá žádnou souvislost s fyzikálními principy řízeného procesu.

1.1.1 Použité modely řízených procesů

Dynamické chování procesu může být popsáno modelem ve formě maticového zlomku

$$\mathbf{G}(z) = \frac{\mathbf{Y}(z)}{\mathbf{U}(z)} = \mathbf{A}^{-1}(z^{-1})\mathbf{B}(z^{-1}) = \mathbf{B}_1(z^{-1})\mathbf{A}_1^{-1}(z^{-1}) \quad (1.1)$$

Kde polynomiální matice $\mathbf{A} \in R_{22}[z^{-1}]$, $\mathbf{B} \in R_{22}[z^{-1}]$ jsou levým nesoudělným rozkladem přenosové matice $\mathbf{G}(z)$ a matice $\mathbf{A}_1 \in R_{22}[z^{-1}]$, $\mathbf{B}_1 \in R_{22}[z^{-1}]$ jsou pravým nesoudělným rozkladem $\mathbf{G}(z)$. $\mathbf{U}(z)$ je vektor akčních veličin a $\mathbf{Y}(z)$ je vektor regulovaných veličin.

Všechny regulátory začleněné do knihovny jsou založeny na modelu s polynomy druhého stupně. Tento model se ukázal jako vhodný při řízení několika laboratorních procesů [4], kde regulátory založené na modelu s polynomy prvního stupně selhaly. Model s polynomy vyššího řádu než druhého vede k poměrně složitému výpočtu parametrů regulátoru a řídicího zákona. Knihovna je ale otevřený systém a je možno přidat regulátory založené na modelech různých řádů. V případě regulátorů využívajících kompenzátorů pro zajištění autonomnosti [8] byl model zjednodušen předpokladem diagonální matice $A(z^{-1})$. Kdyby byla v těchto případech matice $A(z^{-1})$ předpokládána jako nediagonální, znamenalo by to zvýšení řádu regulátoru a výpočet řídicího zákona by se stal složitějším. Model s nediagonální maticí má šestnáct parametrů.

$$A(z^{-1}) = \begin{bmatrix} 1 + a_1 z^{-1} + a_2 z^{-2} & a_3 z^{-1} + a_4 z^{-2} \\ a_5 z^{-1} + a_6 z^{-2} & 1 + a_7 z^{-1} + a_8 z^{-2} \end{bmatrix} \quad (1.2)$$

$$B(z^{-1}) = \begin{bmatrix} b_1 z^{-1} + b_2 z^{-2} & b_3 z^{-1} + b_4 z^{-2} \\ b_5 z^{-1} + b_6 z^{-2} & b_7 z^{-1} + b_8 z^{-2} \end{bmatrix} \quad (1.3)$$

Předpoklad diagonální matice $A(z^{-1})$ způsobí redukci počtu parametrů na dvanáct.

$$A(z^{-1}) = \begin{bmatrix} 0 & a_1 z^{-1} + a_2 z^{-2} \\ a_3 z^{-1} + a_4 z^{-2} & 0 \end{bmatrix} \quad (1.4)$$

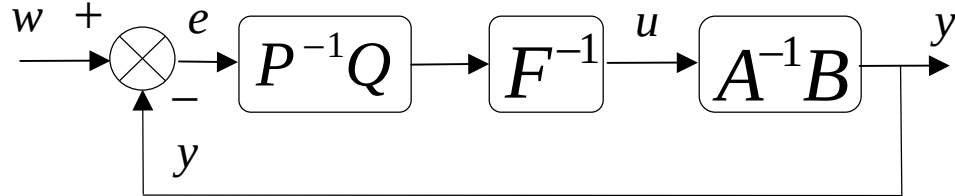
$$B(z^{-1}) = \begin{bmatrix} b_1 z^{-1} + b_2 z^{-2} & b_3 z^{-1} + b_4 z^{-2} \\ b_5 z^{-1} + b_6 z^{-2} & b_7 z^{-1} + b_8 z^{-2} \end{bmatrix} \quad (1.5)$$

Pro výpočet řídicího zákona jsou využity odhady parametrů modelu řízeného procesu. V rámci samočinně se nastavujících regulátorů je možno simulovat řízení spojitých systémů nebo diskrétních systémů se strukturou odlišnou od struktury modelu. Je také možno připojit a řídit různé laboratorní modely se dvěma vstupy a dvěma výstupy.

V následujících kapitolách bude popsán návrh algoritmů jednotlivých regulátorů začleněných do knihovny. Je to zpětnovazební regulátor s jedním stupněm volnosti (1DOF regulátor), dva regulátory se dvěma stupni volnosti (2DOF regulátory) – první z nich využívá klasickou konfiguraci se zpětnovazební a přímovazební částí, dále je v textu označován jako 2DOF – A regulátor. Druhý obsahuje dvě zpětné vazby, dále je pro něj používáno označení 2DOF – B regulátor. Dále jsou začleněny tři regulátory využívající známé kompenzátory pro zajištění autonomnosti, postupně značené jako K1, K2 a K3.

1.1.2 Návrh algoritmu regulátoru 1DOF

1DOF konfigurace je zobrazena na Obrázku 1.



Obr. 1. Schéma konfigurace 1DOF

Obsahuje jeden zpětnovazební regulátor. Stejně jako u řízeného systému, přenosová matice regulátoru může být zapsána ve formě maticového zlomku:

$$\mathbf{G}_R(z) = \mathbf{P}^{-1}(z^{-1})\mathbf{Q}(z^{-1}) = \mathbf{Q}_1(z^{-1})\mathbf{P}_1^{-1}(z^{-1}) \quad (1.6)$$

Vektor žádaných veličin $\mathbf{W}(z^{-1})$ je obecně definován jako:

$$\mathbf{W}(z) = \mathbf{F}_w^{-1}(z^{-1})\mathbf{h}(z^{-1}) \quad (1.7)$$

Referenční signály jsou předpokládány ze třídy skokových funkcí. V tomto případě je $\mathbf{h}(z^{-1})$ vektor konstant a $\mathbf{F}_w(z^{-1})$ nabývá tvaru

$$\mathbf{F}_w(z^{-1}) = \begin{bmatrix} 1 - z^{-1} & 0 \\ 0 & 1 - z^{-1} \end{bmatrix} \quad (1.8)$$

Kompenzátor $\mathbf{F}(z^{-1})$ je z regulátoru vyčleněn pouze formálně. Tento komponent musí být součástí regulátoru, aby byl splněn požadavek na asymptotické sledování referenčního signálu. Jestliže jsou referenční signály ze třídy skokových funkcí, $\mathbf{F}(z^{-1})$ je integrátor.

Z blokového schématu na Obrázku 1 vyplývá řídicí zákon (operátor z^{-1} bude kvůli zjednodušení v dalších rovnicích vynechán):

$$\mathbf{U} = \mathbf{F}^{-1}\mathbf{Q}_1\mathbf{P}_1^{-1}\mathbf{E} \quad (1.9)$$

Pro výstup systému platí maticová rovnice:

$$\mathbf{Y} = \mathbf{A}^{-1}\mathbf{B}\mathbf{F}^{-1}\mathbf{P}^{-1}\mathbf{Q}\mathbf{E} = \mathbf{A}^{-1}\mathbf{B}\mathbf{F}^{-1}\mathbf{P}^{-1}\mathbf{Q}(\mathbf{W} - \mathbf{Y}) \quad (1.10)$$

Kterou je možno s využitím pravého maticového zlomku regulátoru postupně upravit do tvaru:

$$\mathbf{Y} = \mathbf{P}_1(\mathbf{A}\mathbf{F}\mathbf{P}_1 + \mathbf{B}\mathbf{Q}_1)^{-1}\mathbf{B}\mathbf{Q}_1\mathbf{P}_1^{-1}\mathbf{W} \quad (1.11)$$

Je zřejmé, že prvky vektoru výstupní veličiny mají ve jmenovateli determinant matice $AF\mathbf{P}_1+B\mathbf{Q}_1$. Tento determinant je charakteristickým polynomem mnohazměrového regulačního obvodu. Kořeny tohoto polynomu mají rozhodující vliv na chování uzavřené smyčky. Musí být umístěny uvnitř jednotkové kružnice, aby byl systém stabilní. Podmínku stability můžeme tedy definovat pomocí následující diofantické rovnice:

$$AF\mathbf{P}_1+B\mathbf{Q}_1=\mathbf{M} \quad (1.12)$$

Kde $\mathbf{M} \in R_{22}[z^{-1}]$ je stabilní diagonální polynomiální matice:

$$\mathbf{M}(z^{-1}) = \begin{bmatrix} 1+m_1z^{-1}+m_2z^{-2}+m_3z^{-3}+m_4z^{-4} & 0 \\ 0 & 1+m_1z^{-1}+m_2z^{-2}+m_3z^{-3}+m_4z^{-4} \end{bmatrix} \quad (1.13)$$

Stupně polynomů v maticích regulátoru musí být zvoleny tak, aby regulační obvod byl vnitřně ryzí. Struktura matic $\mathbf{P}_1$ a $\mathbf{Q}_1$ byla zvolena tak, aby se počet neznámých parametrů regulátoru shodoval s počtem algebraických rovnic plynoucích z řešení diofantické rovnice metodou neurčitých koeficientů. Matice regulátoru byly tedy zvoleny ve tvaru:

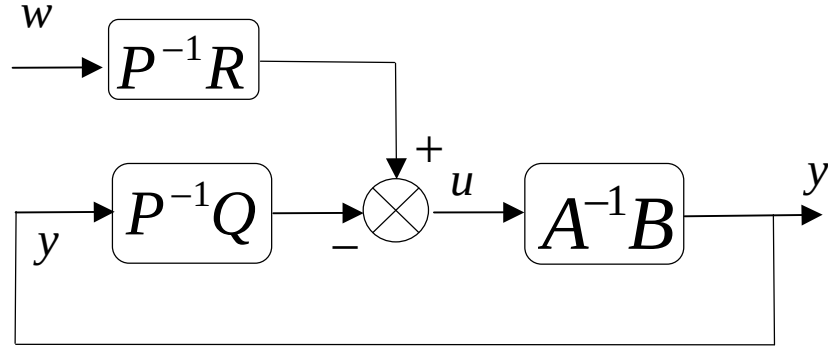
$$\mathbf{P}_1(z^{-1}) = \begin{bmatrix} 1+p_1z^{-1} & p_2z^{-1} \\ p_3z^{-1} & 1+p_4z^{-1} \end{bmatrix} \quad (1.14)$$

$$\mathbf{Q}_1(z^{-1}) = \begin{bmatrix} q_1+q_2z^{-1}+q_3z^{-2} & q_4+q_5z^{-1}+q_6z^{-2} \\ q_7+q_8z^{-1}+q_9z^{-2} & q_{10}+q_{11}z^{-1}+q_{12}z^{-2} \end{bmatrix} \quad (1.15)$$

Z řešení diofantické rovnice vyplývá šestnáct algebraických rovnic, ve kterých vystupují jako neznámé parametry regulátoru. Parametry regulátoru jsou dány řešením těchto rovnic.

1.1.3 Návrh algoritmu regulátoru 2DOF – A

2DOF – A konfigurace je zobrazena na Obrázku 2.



Obr. 2. Schéma konfigurace 2DOF - A

Na Obrázku 2 je znázorněna struktura obvodu s přímovazebním a zpětnovazebním regulátorem.

Z Obrázku 2 plyne rovnice

$$U = P^{-1}RW - P^{-1}QA^{-1}BU \quad (1.16)$$

Takže po úpravách dostáváme rovnici

$$(I + P^{-1}QA^{-1}B)U = P^{-1}RW \quad (1.17)$$

Úpravami tohoto vztahu dostáváme

$$U = A_1(PA_1 + QB_1)^{-1}RW \quad (1.18)$$

a dále potom

$$Y = A^1BU = B_1(PA_1 + QB_1)^{-1}RW \quad (1.19)$$

Uzavřená smyčka je tedy stabilní, pouze když

$$PA_1 + QB_1 = M \quad (1.20)$$

kde M je zvolená stabilní polynomiální matice

$$M(z^{-1}) = \begin{bmatrix} 1 + m_1z^{-1} + m_2z^{-2} + m_3z^{-3} + m_4z^{-4} & 0 \\ 0 & 1 + m_1z^{-1} + m_2z^{-2} + m_3z^{-3} + m_4z^{-4} \end{bmatrix} \quad (1.21)$$

Z blokového schématu vyplývá rovnice pro regulační odchylku (tato rovnice platí pouze v případě, že diagonální elementy matice $\mathbf{M}$ jsou si rovny).

$$\mathbf{E} = \mathbf{M}^{-1}(\mathbf{M} - \mathbf{B}_1 \mathbf{R})^{-1} \mathbf{F}^{-1} \mathbf{h} \quad (1.22)$$

Požadavek na asymptotické sledování referenčního signálu je splněn tehdy, jestliže existuje taková polynomiální matice $\mathbf{T}$, aby byla splněna následující diofantická rovnice.

$$\mathbf{M} - \mathbf{B}_1 \mathbf{R} = \mathbf{T} \mathbf{F} \quad (1.23)$$

Polynomiální matice $\mathbf{T}$ a $\mathbf{R}$ jsou řešením další diofantické rovnice

$$\mathbf{T} \mathbf{F} + \mathbf{B}_1 \mathbf{R} = \mathbf{M} \quad (1.24)$$

Matice $\mathbf{P}$, $\mathbf{Q}$ a $\mathbf{R}$ jsou matice regulátoru.

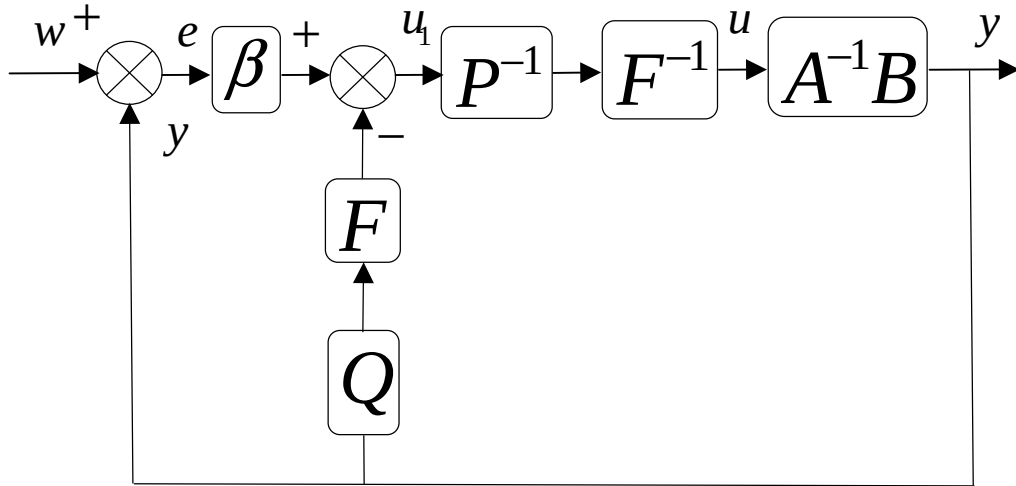
$$\mathbf{P}(z^{-1}) = \begin{bmatrix} 1 + p_1 z^{-1} + p_2 z^{-2} & p_3 z^{-1} + p_4 z^{-2} \\ p_5 z^{-1} + p_6 z^{-2} & 1 + p_7 z^{-1} + p_8 z^{-2} \end{bmatrix} \quad \mathbf{Q}(z^{-1}) = \begin{bmatrix} q_1 z^{-1} + q_2 z^{-2} & q_3 z^{-1} + q_4 z^{-2} \\ q_5 z^{-1} + q_6 z^{-2} & q_7 z^{-1} + q_8 z^{-2} \end{bmatrix} \quad (1.25)$$

$$\mathbf{R}(z^{-1}) = \begin{bmatrix} r_1 z^{-1} + r_2 z^{-2} & r_3 z^{-1} + r_4 z^{-2} \\ r_5 z^{-1} + r_6 z^{-2} & r_7 z^{-1} + r_8 z^{-2} \end{bmatrix} \quad \mathbf{T}(z^{-1}) = \begin{bmatrix} 1 + t_1 z^{-1} + t_2 z^{-2} & t_3 z^{-1} + t_4 z^{-2} \\ t_5 z^{-1} + t_6 z^{-2} & 1 + t_7 z^{-1} + t_8 z^{-2} \end{bmatrix} \quad (1.26)$$

Soustava algebraických rovnic, jejímž řešením jsou neznámé parametry regulátoru, plyne z řešení diofantických rovnic (1.20) a (1.24) metodou neurčitých koeficientů.

1.1.4 Návrh algoritmu regulátoru 2DOF – B

2DOF – B konfigurace je zobrazena na Obrázku 3.



Obr. 3. Schéma konfigurace 2DOF – B

Tato struktura obsahuje dvě zpětné vazby. Ze schématu plyne maticová rovnice pro výstup ze soustavy:

$$Y = A^{-1}BU = A^{-1}BF^{-1}P^{-1}U_1 \quad (1.27)$$

Kde

$$U_1 = \beta(W - Y) - QFY \quad (1.28)$$

Pro výstup z regulátoru platí vztah:

$$U = F^{-1}P^{-1}U_1 \quad (1.29)$$

Po dosazení za U_1 a Y :

$$U = F^{-1}P^{-1}[\beta(W - A^{-1}BU) - QFA^{-1}BU] \quad (1.30)$$

Rovnice (1.29) může být s využitím pravého maticového zlomku řízeného systému upravena do tvaru:

$$U = A_1[PA_1 + (\beta + FQ)B_1]\beta W \quad (1.31)$$

Regulační obvod je stabilní, jestliže je splněna diofantická rovnice:

$$PFA_1 + (\beta + FQ)B_1 = M \quad (1.32)$$

Matice M je definována

$$\mathbf{M}(z^{-1}) = \begin{bmatrix} 1 + m_1 z^{-1} + m_2 z^{-2} + m_3 z^{-3} + m_4 z^{-4} & 0 \\ 0 & 1 + m_5 z^{-1} + m_6 z^{-2} + m_7 z^{-3} + m_8 z^{-4} \end{bmatrix} \quad (1.33)$$

Struktura matic regulátoru byla zvolena podle stejných pravidel jako v předchozích případech následovně:

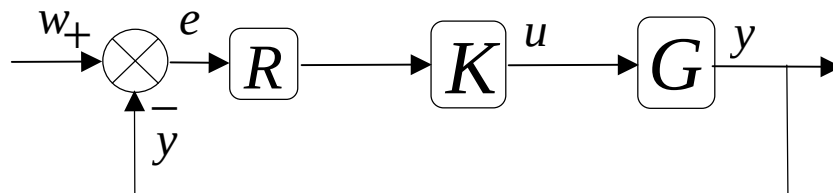
$$\mathbf{P}(z^{-1}) = \begin{bmatrix} 1 + p_1 z^{-1} & p_2 z^{-1} \\ p_3 z^{-1} & 1 + p_4 z^{-1} \end{bmatrix} \quad \mathbf{Q}(z^{-1}) = \begin{bmatrix} q_1 + q_2 z^{-1} & q_3 + q_4 z^{-1} \\ q_5 + q_6 z^{-1} & q_7 + q_8 z^{-1} \end{bmatrix} \quad (1.34)$$

$$\boldsymbol{\beta}(z^{-1}) = \begin{bmatrix} \beta_1 & \beta_2 \\ \beta_3 & \beta_4 \end{bmatrix} \quad (1.35)$$

Hodnoty parametrů regulátoru jsou výsledkem řešení diofantické rovnice (1.32).

1.1.5 Autonomní řízení s využitím kompenzátorů

Existuje několik přístupů k řízení mnohazměrových systémů s vnitřními interakcemi. Jednou z možností je zařazení kompenzátoru do série před soustavu. Cílem je potlačení nežádoucích interakcí mezi vstupními a výstupními veličinami tak, aby každá vstupní veličina ovlivňovala pouze jednu veličinu regulovanou.



Obr. 4. Schéma regulátoru s kompenzátozem

Výsledná přenosová matice $\mathbf{H}$ je potom definována jako

$$\mathbf{H} = \mathbf{K}\mathbf{G} \quad (1.36)$$

Podmínky autonomnosti jsou splněny, jestliže matice $\mathbf{H}$ je diagonální.

Matici $\mathbf{B}$ můžeme psát jako

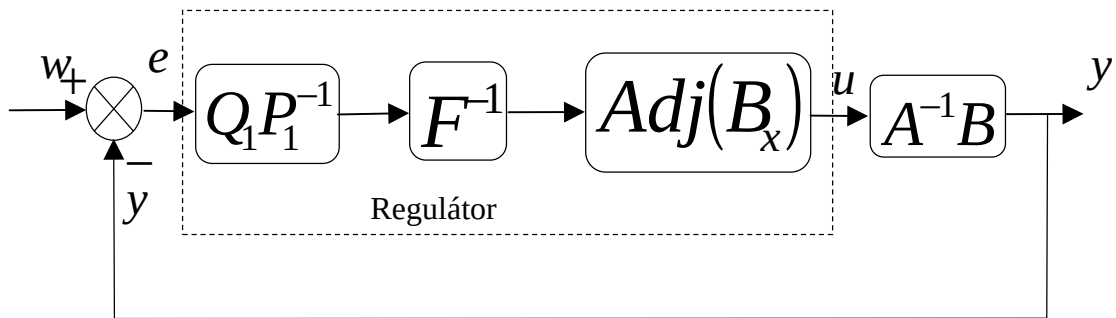
$$\mathbf{B} = z^{-1} \mathbf{B}_x = z^{-1} \begin{bmatrix} b_1 + b_2 z^{-1} & b_3 + b_4 z^{-1} \\ b_5 + b_6 z^{-1} & b_7 + b_8 z^{-1} \end{bmatrix} \quad (1.37)$$

Kompenzátor K1 je adjugovaná matice $\mathbf{B}_x$. Model byl zjednodušen předpokladem, že matice $\mathbf{A}$ je diagonální.

$$\mathbf{A}(z^{-1}) = \begin{bmatrix} 1 + a_1 z^{-1} + a_2 z^{-2} & 0 \\ 0 & 1 + a_3 z^{-1} + a_4 z^{-2} \end{bmatrix} \quad (1.38)$$

Výsledkem součinu matice $\mathbf{B}_x$ a adjugované matice $\mathbf{B}_x$ je diagonální matice $\mathbf{H}$. Determinanty matice $\mathbf{B}_x$ jsou v tomto případě diagonálními prvky. Jestliže matice $\mathbf{A}$ je předpokládána jako nediagonální, její inverze musí být zařazena před systém, aby výsledná matice $\mathbf{H}$ byla diagonální, jinak by došlo ke zvýšení řádu regulátoru a ke zvýšení složitosti uzavřeného regulačního obvodu.

Návrh algoritmu regulátoru s kompenzátozem K1



Obr. 5. Regulační obvod s kompenzátozem K1

Maticová rovnice pro výstup systému má tvar

$$\mathbf{Y} = \mathbf{P}_1(\mathbf{A}\mathbf{F}\mathbf{P}_1 + \mathbf{H}_1\mathbf{Q}_1)\mathbf{H}_1\mathbf{Q}_1\mathbf{P}_1^{-1}\mathbf{W} \quad (1.39)$$

kde

$$\mathbf{H}_1 = z^{-1}\mathbf{B}_x \text{adj}(\mathbf{B}_x) = \begin{bmatrix} \det(\mathbf{B}_x) & 0 \\ 0 & \det(\mathbf{B}_x) \end{bmatrix} \quad (1.40)$$

Pro zajištění stability uzavřeného regulačního obvodu musí být splněna následující diofantická rovnice

$$\mathbf{A}\mathbf{F}\mathbf{P}_1 + \mathbf{H}_1\mathbf{Q}_1 = \mathbf{M} \quad (1.41)$$

Polynomiální matice regulátoru jsou zvoleny následovně

$$\mathbf{P}_1(z^{-1}) = \begin{bmatrix} 1 + p_1 z^{-1} + p_2 z^{-2} & 0 \\ 0 & 1 + p_3 z^{-1} + p_4 z^{-2} \end{bmatrix} \quad (1.42)$$

$$\mathbf{Q}_1(z^{-1}) = \begin{bmatrix} q_1 + q_2 z^{-1} + q_3 z^{-2} & 0 \\ 0 & q_4 + q_5 z^{-1} + q_6 z^{-2} \end{bmatrix}$$

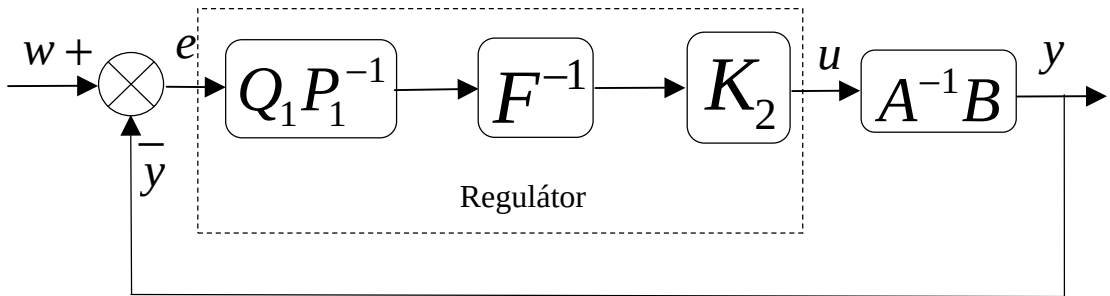
a matice $\mathbf{M}$ je definována jako

$$\mathbf{M}(z^{-1}) = \begin{bmatrix} 1 + m_1 z^{-1} + & & \\ + m_2 z^{-2} + m_3 z^{-3} + & 0 & \\ + m_4 z^{-4} + m_5 z^{-5} & & \\ & 0 & 1 + m_6 z^{-1} + \\ & & m_7 z^{-2} + m_8 z^{-3} + \\ & & + m_9 z^{-4} + m_{10} z^{-5} \end{bmatrix} \quad (1.43)$$

Diofantická rovnice definuje soustavu algebraických rovnic, jejímž řešením jsou neznámé parametry regulátoru.

Návrh algoritmu regulátoru s kompenzátorem K2

Blokové schéma uzavřeného regulačního obvodu s kompenzátorem K2 je zobrazeno na Obrázku 6.



Obr. 6. Regulační obvod s kompenzátorem K2

Kompenzátor K2 je definován následujícím vztahem

$$\mathbf{C}_1 = \frac{1}{\det(\mathbf{B}_x)} \begin{bmatrix} B_{22} & -B_{12} \\ -B_{21} & B_{11} \end{bmatrix} \quad (1.44)$$

Součin matice $\mathbf{B}_x$ a kompenzátoru je diagonální matice. V případě volby matice $\mathbf{A}$ jako diagonální platí předpoklad uvedený v předchozí kapitole. Výsledná přenosová matice $\mathbf{H}_2$ je definována rovnicí (1.45).

$$\mathbf{H}_2 = \mathbf{B}_x \mathbf{C}_1 z^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} z^{-1} \quad (1.45)$$

Rovnice výstupu řízeného systému nabývá po úpravách maticovými operacemi tvaru

$$\mathbf{Y} = \mathbf{P}_1 (\mathbf{A} \mathbf{F} \mathbf{P}_1 + \mathbf{H}_1 \mathbf{Q}_1)^{-1} \mathbf{H}_1 \mathbf{Q}_1 \mathbf{P}_1 \mathbf{W} \quad (1.46)$$

Podmínky BIBO stability jsou definovány následující diofantickou rovnicí

$$\mathbf{A} \mathbf{F} \mathbf{P}_1 + \mathbf{H}_1 \mathbf{Q}_1 = \mathbf{M} \quad (1.47)$$

Matice $\mathbf{M}$ v tomto případě nabývá tvaru

$$\mathbf{M}(z^{-1}) = \begin{bmatrix} 1 + m_1 z^{-1} + m_2 z^{-2} + & 0 \\ + m_3 z^{-3} + m_4 z^{-4} & \\ 0 & 1 + m_1 z^{-1} + m_2 z^{-2} + \\ & + m_3 z^{-3} + m_4 z^{-4} \end{bmatrix} \quad (1.48)$$

Struktura matice regulátoru byla zvolena podle stejných pravidel uvedených v předchozích kapitolách

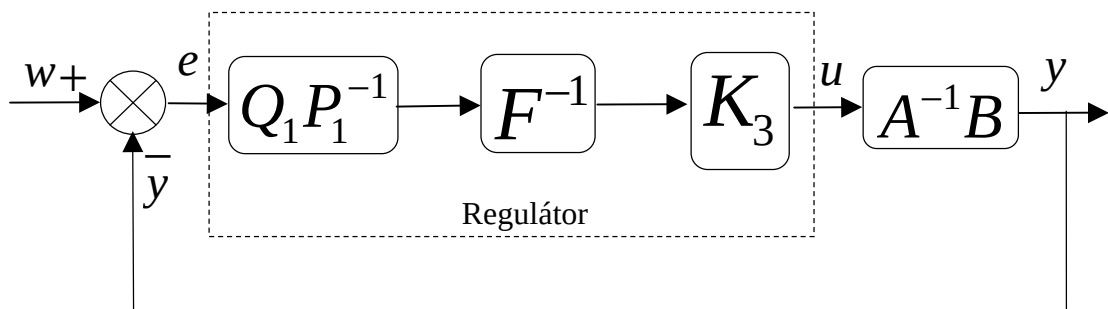
$$\mathbf{P}_1(z^{-1}) = \begin{bmatrix} 1 + p_1 z^{-1} & 0 \\ 0 & 1 + p_2 z^{-1} \end{bmatrix} \quad (1.49)$$

$$\mathbf{Q}_1(z^{-1}) = \begin{bmatrix} q_1 + q_2 z^{-1} + q_3 z^{-2} & 0 \\ 0 & q_4 + q_5 z^{-1} + q_6 z^{-2} \end{bmatrix}$$

Řešení diofantické rovnice opět vede na soustavu algebraických rovnic.

Návrh algoritmu regulátoru s kompenzátozem K3

Blokové schéma uzavřeného regulačního obvodu s kompenzátozem K3 je na Obrázku 7.



Obr. 7. Regulační obvod s kompenzátozem K3

Kompensátor K3 můžeme vyjádřit jako

$$\mathbf{H}_3 = \frac{1}{\det(\mathbf{B}_x)} \begin{bmatrix} B_{11}B_{22} & -B_{12}B_{22} \\ -B_{11}B_{21} & B_{11}B_{22} \end{bmatrix} \quad (1.50)$$

Diagonální přenosová matice je definována vztahem

$$\mathbf{H}_3 = \mathbf{B}_x \mathbf{C}_2 z^{-1} = \begin{bmatrix} B_{11} & 0 \\ 0 & B_{22} \end{bmatrix} z^{-1} \quad (1.51)$$

Rovnice výstupu řízeného systému plyne z blokového schématu Obrázek 7

$$\mathbf{Y} = \mathbf{P}_1 (\mathbf{A} \mathbf{F} \mathbf{P}_1 + \mathbf{H}_3 \mathbf{Q}_1)^{-1} \mathbf{H}_3 \mathbf{Q}_1 \mathbf{P}_1 \mathbf{W} \quad (1.52)$$

Pro zajištění stability musí být splněna diofantická rovnice

$$\mathbf{A} \mathbf{F} \mathbf{P}_1 + \mathbf{H}_3 \mathbf{Q}_1 = \mathbf{M} \quad (1.53)$$

Kde matice $\mathbf{M}$ je definována výrazem (1.48).

Struktura polynomiálních matic $\mathbf{P}_1$ a $\mathbf{Q}_1$ byla zvolena následovně

$$\begin{aligned} \mathbf{P}_1(z^{-1}) &= \begin{bmatrix} 1 + p_1 z^{-1} & 0 \\ 0 & 1 + p_2 z^{-1} \end{bmatrix} \\ \mathbf{Q}_1(z^{-1}) &= \begin{bmatrix} q_1 + q_2 z^{-1} + q_3 z^{-2} & 0 \\ 0 & q_4 + q_5 z^{-1} + q_6 z^{-2} \end{bmatrix} \end{aligned} \quad (1.54)$$

1.1.6 Rekurzivní identifikace

Pro použití samočinně se nastavujícího regulátoru není možné použít pro výpočet odhadů parametrů modelu procesu klasickou verzi ARX modelu, ale musíme použít jeho rekurzivní verzi vhodnou pro identifikaci v reálném čase. Při této modifikaci se používají nově naměřené hodnoty pouze pro opravu (korekci) původních odhadů, čímž klesá výpočetní složitost identifikačních algoritmů. Rekurzivní algoritmy umožňují sledovat změny vlastností (parametrů) procesu v reálném čase, a proto jsou základem samočinně se nastavujících regulátorů.

Nechť lineární jednorozměrový stochastický model je popsán modelem ARX. O neměřitelné náhodné složce $e_s(k)$ předpokládáme, že je posloupností vzájemně nekorelované náhodné veličiny a rovněž nekorelované se vstupem a výstupem procesu. Dále předpokládáme, že náhodná veličina má nulovou střední hodnotu a konstantní

kovarianci (rozptyl). Výhodou rekurzivní metody nejmenších čtverců je ta skutečnost, že potřebuje nejmenší objem apriorních informací o náhodné složce $e_s(k)$.

Naším úkolem je průběžně odhadovat neznámé parametry Θ modelu na základě vstupů a výstupů k časovému okamžiku k , $\{y(i), u(i), i = k, k-1, k-2, \dots, k_0\}$ (k_0 je počáteční čas identifikace). Hledáme takový vektor $\hat{\Theta}$ o rozměru $n_z = 2n$, který minimalizuje kritérium

$$J_k(\Theta) = \sum_{i=k_0}^k e_s^2(i) \quad (1.55)$$

kde

$$e_s(i) = y(i) - \Theta^T \phi(i) = \begin{bmatrix} 1 & -\Theta^T \end{bmatrix} \begin{bmatrix} y(i) \\ \phi(i) \end{bmatrix} \quad (1.56)$$

Jestliže požadujeme, aby algoritmus byl schopen sledovat pomalé změny parametrů identifikovaného procesu, můžeme toho dosáhnout technikou exponenciálního zapomínání. Potom minimalizujeme modifikované kritérium

$$J_k(\Theta) = \sum_{i=k_0}^k \varphi^{2(k-i)} e_s^2(i) \quad (1.57)$$

kde $0 < \varphi^2 \leq 1$ je faktor exponenciálního zapomínání.

V této práci je využit algoritmus směrového zapomínání, kdy se standardní exponenciální zapomínání aplikuje pouze na data přinášející novou informaci.

Vektor odhadu parametrů se aktualizuje podle rekurzivního vztahu

$$\hat{\Theta}(k) = \hat{\Theta}(k-1) + \frac{\mathbf{C}(k-1)\boldsymbol{\varphi}(k-1)}{1 + \xi(k-1)} \hat{e}(k-1) \quad (1.58)$$

kde

$$\xi(k) = \boldsymbol{\varphi}^T(k-1)\mathbf{C}(k-1)\boldsymbol{\varphi}(k-1) \quad (1.59)$$

je pomocný skalár a

$$\hat{e}(k) = y(k) - \hat{\Theta}^T(k)\boldsymbol{\varphi}(k-1) \quad (1.60)$$

je chyba predikce. Jestliže $\xi(k) > 0$, potom čtvercová kovarianční matice o rozměru n_z je aktualizována podle vztahu

$$\mathbf{C}(k) = \mathbf{C}(k-1) - \frac{\mathbf{C}(k-1)\boldsymbol{\varphi}(k-1)\boldsymbol{\varphi}^T(k-1)\mathbf{C}(k-1)}{\varepsilon^{-1}(k) + \xi(k-1)} \quad (1.61)$$

kde

$$\varepsilon(k) = \varphi(k) - \frac{1 - \varphi(k)}{\xi(k-1)} \quad (1.62)$$

Jestliže $\xi(k-1) = 0$, potom

$$\mathbf{C}(k) = \mathbf{C}(k-1) \quad (1.63)$$

Hodnota adaptivního směrového zapomínání $\varphi(k)$ je potom počítána v každé periodě vzorkování podle vztahu

$$\varphi(k) = \left\{ 1 + (1 + \rho) [\ln(1 + \xi(k-1))] + \left[\frac{(\nu(k-1) + 1)\eta(k-1)}{1 + \xi(k-1) + \eta(k-1)} - 1 \right] \frac{\xi(k-1)}{1 + \xi(k-1)} \right\}^{-1} \quad (1.64)$$

kde

$$\eta(k) = \frac{\hat{e}^2(k)}{\lambda(k)} \quad \nu(k) = \varphi(k) [(\nu(k-1) + 1)] \quad (1.65)$$

$$\lambda(k) = \varphi(k) \left[\lambda(k-1) + \frac{\hat{e}^2(k-1)}{1 + \xi(k-1)} \right] \quad (1.66)$$

jsou pomocné proměnné. Pro start algoritmu se osvědčilo vhodné zvolit následující počáteční podmínky: Prvky hlavní diagonály kovarianční matice $C_{ii}(0) = 10^3$, počáteční hodnota faktoru směrového zapomínání $\varphi(0) = 1$, $\lambda(0) = 0.001$, $\nu(0) = 10^{-6}$, $\rho = 0.99$. Volba počátečních odhadů vektoru parametrů $\hat{\Theta}(0)$ se provede na základě apriorní informace.

Výše uvedené vztahy lze přímo naprogramovat jako s-funkci v programovém systému MATLAB, aniž bychom brali zřetel na numerické aspekty, tj. bez použití numerických filtrů.

1.1.7 Modifikace průběžné identifikace pro regulátory bez kompenzátorů

V případě použití systému se dvěma vstupy a dvěma výstupy se bude identifikovat celkem šestnáct neznámých parametrů modelu ARX .

$$\begin{aligned} y_1(k) &= \Theta_1^T(k) \phi(k-1) + n_1(k) \\ y_2(k) &= \Theta_2^T(k) \phi(k-1) + n_2(k) \end{aligned} \quad (1.67)$$

Máme tedy dva vektory parametrů diskrétního modelu o rozměru $n = 8$.

$$\Theta_1^T(k) = [a_1, a_2, a_3, a_4, b_1, b_2, b_3, b_4] \quad (1.68)$$

$$\Theta_2^T(k) = [a_5, a_6, a_7, a_8, b_5, b_6, b_7, b_8] \quad (1.69)$$

Vektor dat pak plníme následujícím způsobem

$$\begin{aligned} \phi^T(k-1) = & [-y_1(k-1), -y_1(k-2), -y_2(k-1), -y_2(k-2), \\ & u_1(k-1), u_1(k-2), u_2(k-1), u_2(k-2)] \end{aligned} \quad (1.70)$$

Aktualizace odhadu vektoru parametrů pak probíhá podle vztahu

$$\Theta_{1,2}^T(k) = \Theta_{1,2}^T(k-1) + \frac{C(k-1)\phi(k)}{1 + \phi^T(k)C(k-1)\phi(k)} (y(k) - \Theta_{1,2}^T(k-1)\phi(k)) \quad (1.71)$$

1.1.8 Modifikace průběžné identifikace pro regulátory s korekčními členy

V případě použití systému se dvěma vstupy a dvěma výstupy se bude identifikovat celkem dvanáct neznámých parametrů modelu ARX .

$$\begin{aligned} y_1(k) &= \Theta_1^T(k)\phi_1(k-1) \\ y_2(k) &= \Theta_2^T(k)\phi_2(k-1) \end{aligned} \quad (1.72)$$

Máme tedy dva vektory parametrů diskrétního modelu o rozměru $n = 6$

$$\Theta_1^T(k) = [a_1, a_2, b_1, b_2, b_3, b_4] \quad (1.73)$$

$$\Theta_2^T(k) = [a_3, a_4, b_5, b_6, b_7, b_8] \quad (1.74)$$

Vektor dat pak plníme následujícím způsobem

$$\phi_1^T(k-1) = [-y_1(k-1), -y_1(k-2), u_1(k-1), u_1(k-2), u_2(k-1), u_2(k-2)] \quad (1.75)$$

$$\phi_2^T(k-1) = [-y_2(k-1), -y_2(k-2), u_1(k-1), u_1(k-2), u_2(k-1), u_2(k-2)] \quad (1.76)$$

1.2 Simulace v prostředí Matlab – Simulink

V prostředí Matlab – Simulink je vytvořena knihovna samočinně se nastavujících regulátorů [3].

Jednotlivé regulátory jsou tvořeny jako maska subsystému, tedy jako soubor bloků Simulink o určitých vstupech, výstupech a parametrech [2]. Každý regulátor používá tři vstupní veličiny (regulovanou veličinu y , její žádanou hodnotu w a akční veličinu u) a poskytuje dva výstupy (akční veličinu u a aktuální hodnoty identifikovaných parametrů modelu procesu).

Vnitřní struktura každého regulátoru je opět vytvořena jako bloky prostředí Simulink, což nabízí mimo jiné možnost jednoduché modifikace.

Z hlediska struktury se regulátor skládá ze tří základních částí:

- bloku průběžné identifikace
- bloku výpočtu parametrů regulátoru
- bloku výpočtu akčního zásahu

Z programového pohledu odpovídají jednotlivým částem regulátoru samostatné programové soubory, které jsou po spuštění simulace interpretovány prostředím Matlab. K tomuto účelu byly použity bloky MATLAB Function a S-Function. Blok MATLAB Function slouží k volání funkce, která převádí vstupní vektor na výstupní vektor. S-Function je obecnější a odpovídající funkce má možnost uchovávat i diskrétní a spojitě stavy a tím pracovat s určitou periodou vzorkování. S-funkcemi jsou realizovány algoritmy průběžné identifikace a akčního zásahu. M-funkcí je realizován výpočet parametrů regulátoru.

M-funkce nepracuje se stavy, zabývá se pouze vstupy a výstupy.

Ve vytvořené knihovně [3] jsou všechny funkce realizovány jako m-soubory (přípona .m), jsou tedy psány v programovacím jazyce prostředí Matlab.

1.2.1 Tvorba aplikací pomocí programu Real – Time Workshop

Po simulačním ověření regulátorů přichází na řadu implementace do reálného systému. Pomocí programu Real-Time Workshop, který je součástí programového prostředí Matlab – Simulink, je možné na základě schémat regulátorů v programu

Simulink generovat kód, který je použitelný pro programování programovatelných logických automatů (PLC), průmyslových počítačů (IPC) nebo speciálních počítačových karet. Velkou výhodou uvedeného přístupu je možnost zakázkové tvorby regulátorů, kdy uživatel, který potřebuje zavedení a ověření regulátoru, nemusí vlastnit relativně drahou licenci programu Matlab. Stačí, aby si tvorbu regulátoru objednal u instituce, která příslušnou licenci vlastní, a po provedení návrhu a ověření regulátoru touto institucí je uživateli předán pouze zdrojový, popř. binární kód výsledného regulátoru.

1.2.2 Generování zdrojových kódů

Proces generování zdrojových kódů je řízen překladačnými soubory, které zpracovává Target Language Compiler. Překladačové soubory (jsou označeny příponou .tlc) popisují převod schémat Simulink do cílového programovacího jazyka. Tím jsou vytvořeny zdrojové soubory, jejichž kompilací vznikne výsledná aplikace. Volbou různých překladačových souborů je možné vytvářet aplikace pro různé procesory a operační systémy. Mnozí výrobci průmyslových počítačů vytvářejí vlastní překladačové soubory, které lze integrovat do programu Simulink, takže lze převádět bloková schémata do kódu použitelného pro jejich výrobky. Prostředí Simulink je v tomto směru poměrně otevřené a i ostatní uživatelé mohou vytvářet knihovny pro převod blokových schémat do zdrojových souborů a dosáhnout kompatibility s libovolným hardwarem. Vytvoření aplikace spočívá pouze ve výběru správného překladačového souboru, případném nastavení parametrů a spuštění překladačového procesu.

1.2.3 Slučitelnost schémat Simulink

Překladačové soubory, které jsou standardně dodávány, bohužel nepodporují všechny bloky prostředí Simulink. Množina nepodporovaných bloků se pro jednotlivé překladačové soubory liší. V tomto případě se narazí na problémy u dvou bloků: MATLAB Function a S-Function. Překladačové soubory nepodporují blok MATLAB Function a blok S-Function je podporován jen pro případ, kdy je s-funkce psána v jazyce C a nepoužívá volání interních funkcí programu Matlab. K zajištění kompatibility knihovny se systémem Real – Time Workshop bude nutné naprogramovat jednotlivé části regulátorů v jazyce C.

Bloky MATLAB Function, které jsou v rámci knihovny používány k výpočtu parametrů regulátorů, budou nahrazeny s-funkcemi v jazyce C, které pouze převádějí své vstupy na výstupy – nejsou použity spojitě ani diskrétní stavy.

Bloky zajišťující průběžnou identifikaci a výpočet akčního zásahu jsou realizovány jako s-funkce, které bude třeba převést z programovacího jazyka používaného prostředím Matlab do jazyka C.

Názvy souborů zůstávají při převodu stejné, pouze místo původní přípony .m je použita přípona .c označující zdrojové kódy jazyka C.

1.3 S-funkce

Využívání programovacího jazyka prostředí Matlab přináší několik výhod [2]. Například snadnější práci s kódem, kdy není třeba vytvářet deklarace proměnných, a možnost snadného provádění maticových a vektorových operací. Rozdíl je patrný především u bloku průběžné identifikace, který používá násobení a transpozice matic, a při výpočtu parametrů regulátorů, kde je používána maticová inverze a maticové dělení. Další výhodou je snadná čitelnost a z ní plynoucí možnost studování funkčnosti jednotlivých částí regulátorů. Využívání programovacího jazyka C umožňuje přenositelnost funkcí do standardních cílových platforem pomocí programu Real – Time Workshop. Nevýhodou je naopak větší rozsah zdrojových textů, jejich horší čitelnost pro uživatele a nepřítomnost funkcí pro operace s maticemi.

Jak je uvedeno v dokumentaci k Matlabu [5], s-funkce (systémové funkce) popisují jednotlivé bloky Simulinku. Mohou být napsány přímo v Matlabu nebo v jazycích C, C++, Ada nebo Fortran. Nejsou-li napsány v Matlabu, lze je pomocí utility mex (MATLAB EXecutable) zkompileovat na soubor typu .dll, který je k Matlabu v případě vyžádání dynamicky připojen. Tvar s-funkcí je velmi obecný a může vyhovět požadavkům spojitých, diskrétních i hybridních systémů.

1.3.1 Přičlenění s-funkce k modelu

V knihovně bloků a funkcí se vybere blok s názvem S-Function a přetáhne se do modelu. Poté se ve vlastnostech příslušného bloku zapíše název s-funkce. Bude-li název odkazovat na dva stejné zdrojové soubory, z nichž jeden je mex-soubor a druhý m-soubor, použije se přednostně mex-soubor.

Automaticky se předávají čtyři parametry: t (čas), u (vstupní vektor), x (stavový vektor) a flag (celočíselná hodnota uvádějící úlohu, která má být provedena). Při zadávání

dalších parametrů se bere v úvahu jejich pořadí podle požadavků s-funkce. Parametry se oddělují čárkou. Parametrem může být konstanta, jméno proměnné nebo výraz.

S-funkce se uplatňují nejčastěji v těchto případech:

- Přidání nového obecného bloku do Simulinku
- Přidání bloku reprezentujícího ovladače technického zařízení
- Začlenění daného kódu jazyka C do simulace
- Popsání systému souborem matematických rovnic
- Použití grafických animací

Výhodou je, že se může vytvořit obecný blok, který se využije v jednom modelu mnohokrát, pokaždé s jinými parametry.

1.3.2 Jak pracuje s-funkce

K porozumění práce s-funkce je třeba pochopit způsob, jakým Simulink simuluje model a jaké jsou matematické vztahy mezi vstupy, stavy a výstupy bloků. Výstupy jsou funkcí periody vzorkování, vstupů a stavů.

Výpočet modelu probíhá postupně [5]. Nejdříve Simulink začlení bloky do modelu, rozšíří typy dat a nastaví periody vzorkování. Vypočítá parametry bloků, rozhodne o pořadí jejich vykonávání a přidělí paměť. Poté zahájí simulační smyčku. Každý průchod skrz ni je označován jako krok simulace. Během každého kroku simulace vykonává Simulink všechny bloky modelu v pořadí určeném při inicializaci. Pro každý blok vyvolává Simulink funkce, které počítají stavy, derivace a výstupy pro příslušný okamžik. Takto se pokračuje, dokud není simulace dokončena.

S-funkce obsahuje soubor zpětně volaných metod, které vykonávají úlohy požadované pro každý krok simulace. Během simulace modelu, v každém simulačním kroku, Simulink volá příslušné metody pro každý blok s-funkce v modelu. Úlohy vykonávané metodami s-funkce zahrnují:

- Počáteční nastavení parametrů – přednostně v první simulační smyčce Simulink inicializuje s-funkci. Vytváří SimStruct, strukturu obsahující informace o s-funkci. Nastavuje počet a rozměr vstupních a výstupních portů, periodu vzorkování bloku a přiděluje paměťové oblasti.

- Výpočet následujícího akčního zásahu – je-li vytvořen blok proměnné periody vzorkování, tak se vypočítá příští velikost kroku.
- Výpočet výstupů – po dokončení tohoto volání jsou všechny výstupní porty bloků platné pro současný časový krok.
- Aktualizace diskrétních stavů – všechny bloky by v tomto volání měly vykonávat činnosti prováděné pouze jedenkrát během kroku, např. aktualizaci diskrétních stavů pro další simulační smyčku.
- Integrace – týká se modelů se spojitými stavy.

1.3.3 Zavádění s-funkcí

S-funkce může být vytvořena jako m-soubor nebo mex-soubor. Každý způsob má své výhody i nevýhody [5].

S-funkce jako m-soubor

Soubor se skládá z funkce Matlabu v následujícím tvaru:

$$[\text{sys}, \text{x0}, \text{str}, \text{ts}] = \text{f}(\text{t}, \text{x}, \text{u}, \text{flag}, \text{p1}, \text{p2}, \dots)$$

kde *f* je jméno s-funkce, *t* je čas, *x* je stavový vektor, *u* je vstup, *flag* (příznak) označuje úkol, který má být vykonán a *p1*, *p2*, ... jsou parametry bloku.

Výstupní vektor obsahuje následující prvky:

- *sys* – obecný návratový parametr. Vracená hodnota závisí na hodnotě příznaku. Například pro *flag* = 3 bude obsahovat výstupy s-funkce a pro *flag* = 0 informace o funkci.
- *x0* – počáteční stavové hodnoty (prázdný vektor, pokud systém nemá stavy). *X0* je ignorováno kromě *flag* = 0.
- *str* – rezerva pro budoucí použití. V m-souboru musí být prázdnou maticí.
- *ts* – dvousloupcová matice obsahující periodu vzorkování a offsety bloku. Spojité systémy mají periodu vzorkování nastavenou na nulu.

Během simulace modelu Simulink opakovaně vyvolává *f*, používá příznak *k* označení úkolů, které mají být vykonány pro jednotlivá volání. Šablona pro vytváření m-souborů, *sfuntmpl.m*, se nachází v adresáři *matlabroot/toolbox/simulink/blocks*. Skládá se z funkcí a souboru podfunkcí, kde každá z nich odpovídá jednotlivé hodnotě příznaku.

Funkce jednoduše vyvolávají podfunkce podle hodnoty příznaku. Podfunkce jsou označovány jako zpětně volané metody s-funkce a vykonávají aktuálně požadované úkoly. Následující tabulka popisuje obsah m-souboru a odpovídající funkce mex-souboru.

Tabulka 1. Obsah m-souboru

Stupeň simulace	Funkce v mex-souboru	Flag (příznak) v m-souboru
Inicializace	mdlInitializeSizes	Flag = 0
Výpočet času dalšího vzorku	mdlGetTimeOfNextVarHit	Flag = 4
Výpočet výstupů	mdlOutputs	Flag = 3
Aktualizace diskrétních stavů	mdlUpdate	Flag = 2
Výpočet derivací	mdlDerivatives	Flag = 1
Ukončení simulace	mdlTerminate	Flag = 9

Porovnání mex-souboru a m-souboru

Výhodou m-funkce je rychlost vývoje. Překlenuje se tak časově náročný postup typický pro kompilovaný jazyk (kompilace, spojování, spouštění). Také má snadnější přístup k Matlabu a jeho nástrojům.

Hlavní výhodou mex-funkce je její všestrannost. Více zpětných volání a přístup do struktury SimStruct umožňuje mex-souboru začlenit funkčnosti nepřístupné m-souboru. Například ovládání jiných datových typů než double, komplexní vstupy, maticové vstupy a podobně.

1.3.4 Koncepce s-funkce

Pochopení následujících pojmů je důležité k správnému vytváření s-funkcí.

Direct Feedthrough

Výstup je řízen přímo hodnotou vstupního portu. S-funkce má přímý průchod signálu tehdy, když:

- Výstupní funkce (mdlOutputs nebo příznak = 3) je funkcí vstupu u. Přímý průchod signálu v tomto případě znamená, že vstup u je přístupný v mdlOutputs. Výstupy mohou být i grafického charakteru.
- Funkce mdlGetTimeOfNextVarHit (nebo příznak = 4) zpřístupňuje vstup u.

Např. systém, který vyžaduje své vstupy a má tedy přímovazební řízení, má tvar

$$y = k \times u$$

kde u je vstup, k je zesílení a y je výstup.

Dynamicky určená velikost pole

S-funkce mohou být napsány k podpoře libovolně rozměrných vstupních vektorů. V tomto případě jsou aktuální rozměry vstupu určeny dynamicky na začátku simulace na základě rozměrů vstupního vektoru řídicího s-funkci. Vstupní rozměry mohou být použity k určení počtu spojitých stavů, nespojitých stavů a výstupů.

S-funkce v m-souboru mohou mít pouze jeden vstupní port, který může přijímat pouze jednorozměrové signály (vektory). Nicméně signály mohou mít různou šířku.

V mex-souboru slouží k označení dynamicky určené šířky vstupu hodnota -1 příslušného pole ve struktuře `sizes`, která je vrácena během volání funkce `mdlInitializeSizes`. Lze také určit aktuální šířku vstupu, pokud je s-funkce volána pomocí `length(u)`. Nastaví-li se šířka na nulu, bude vstupní port z s-funkce odstraněn. S-funkce v mex-souboru může mít mnoho vstupně-výstupních portů a každý port může mít různé rozměry. Počet rozměrů a velikost každého z nich může být stanovena dynamicky.

1.4 S-funkce psané v jazyce C

Mex-soubor se stejně jako m-soubor skládá z funkcí a zpětně volaných rutin, kterými Simulink vykonává různé úkoly. Přesto mezi nimi existují významné rozdíly [5].

Mex-soubor je vytvářen v jiném programovacím jazyce: C, C++, Ada nebo Fortran. Nepoužívá příznaky a jeho funkce i rutiny jsou Simulinkem volány přímo. Z tohoto důvodu musí funkce obsažené v mex-souboru dodržovat dohodnuté způsoby pojmenovávání určené Simulinkem. Mex-funkce dále mohou využívat více zpětně volaných funkcí. Mají také přímý přístup k vnitřní datové struktuře `SimStruct`, kterou Simulink využívá k udržení informací o s-funkci. Mohou také přímo přistupovat k pracovnímu prostředí Matlabu.

Šablona pro vytváření mex-souborů v jazyce C, `sfuntmpl_basic.c`, se nachází v adresáři `matlabroot/simulink/src`. Šablona obsahuje základní implementaci všech požadovaných i volitelných zpětně volaných funkcí. Stejná šablona s bohatšími komentáři je ve stejném adresáři v souboru `sfuntmpl_doc.c`.

1.4.1 Povinné definice v souboru s-funkce

Aby byla s-funkce správně kompilovatelná, musí obsahovat na začátku a na konci povinné údaje.

Úvodní blok kódu

Jméno s-funkce:

```
#define S_FUNCTION_NAME jméno_s-funkce
```

Jméno je shodné s názvem zadaným v Simulinku do bloku S-Function.

Formát s-funkce:

```
#define S_FUNCTION_LEVEL 2
```

Určení formátu s-funkce. Formát 1 je kompatibilní se Simulinkem 3.0, doporučuje se ale používat novější formát 2.

Začlenění souboru simstruc.h:

```
#include "simstruc.h "
```

Zpřístupňuje datovou strukturu SimStruct, která obsahuje ukazatele na data použitá simulací. Zároveň vymezuje použitá makra k ukládání a získávání dat z této struktury. Při kompilaci na mex-soubor se vloží dále hlavičkové soubory tmwtypes.h, mex.h a matrix.h. Při kompilaci pro Real – Time Workshop se vloží hlavičkové soubory tmwtypes.h a rt_matrx.h.

Závěrečný blok kódu

```
1 #ifndef MATLAB_MEX_FILE
2 #include "simulink.c"
3 #else
4 #include "cg_sfun.h"
5 #endif
```

Uvedené příkazy vybírají příslušný kód pro jednotlivé aplikace:

- simulink.c je vložen, pokud je soubor kompilován na mex-soubor
- cg_sfun.h je vložen, když je soubor používán ve spojení s Real – Time Workshopem

1.4.2 Zpětně volané metody s-funkce

Všechny s-funkce psané uživatelem musí provádět sadu metod, které Simulink vyvolává během simulace. Volitelné metody jsou volány jen tehdy, pokud jsou v s-funkci definovány.

mdlCheckParameters

Kontroluje platnost parametrů s-funkce. Ověřuje nové nastavení parametrů při změně parametrů nebo jejich přehodnocení během simulace. Změny parametrů s-funkce se mohou udát kdykoli během simulační smyčky (buď na startu simulace nebo během simulačního kroku). Když dojde ke změně během simulačního kroku, Simulink volá tuto metodu dvakrát k ošetření změn parametrů. První volání během simulačního kroku slouží k ověření správnosti parametrů. Doplňkové zpracování parametrů by mělo být provedeno v rutině mdlProcessParameters.

Syntax:

```
void mdlCheckParameters(SimStruct *S)
```

Argumenty:

S – SimStruct reprezentující blok s-funkce

mdlInitializeConditions

Inicializace stavového vektoru s-funkce. Simulink vyvolává tuto volitelnou metodu na začátku simulace. Měla by inicializovat spojitě a diskrétní stavy (pokud existují). Může ale vykonávat i jiné inicializační aktivity vyžadované s-funkcí.

Syntax:

```
void mdlInitializeConditions(SimStruct *S)
```

mdlInitializeSampleTimes

Určení vzorkovacích rozsahů. Tato metoda by měla určovat periodu vzorkování pro každý vzorkovací rozsah, který je ovládán s-funkcí pomocí těchto párových maker:

```
ssSetSampleTime(S, sampleTimeIndex, sample_time)
```

```
ssSetOffsetTime(S, offsetTimeIndex, offset_time)
```

SampleTimeIndex běží od nuly do počtu period vzorkování minus jedna. Počet period vzorkování je určen v mdlInitializeSizes prostřednictvím ssSetNumSampleTimes.

Syntax:

```
void mdlInitializeSampleTimes(SimStruct *S)
```

mdlInitializeSizes

Určuje počet vstupů, výstupů, stavů a parametrů s-funkce i její další vlastnosti. Tato metoda je Simulinkem volána jako první. Měla by určovat:

- počet parametrů dané s-funkce pomocí `ssSetNumSFcnParams`
- počet stavů pomocí `ssSetNumContStates` a `ssSetNumDiscStates`
- počet vstupních portů (`ssSetNumInputPorts`)
- rozměr vstupních portů (`ssSetInputPortWidth`)
- zda má vstupní port přímovazební řízení (`ssSetInputPortDirectFeedThrough`). Port má přímé řízení, pokud je vstup použit ve funkci `mdlGetTimeOfNextVarHit` nebo `mdlOutputs`
- počet výstupních portů (`ssSetNumOutputPorts`)
- rozměr výstupních portů (`ssSetOutputPortWidth`)
- počet period vzorkování
- velikost ovlivňovaných vektorů bloku (`ssSetNumRWork`, `ssSetNumIWork`, `ssSetNumPWork`, `ssSetNumModes`, `ssSetNumNonsampledZCs`)
- vlastnosti simulace implementované blokem (`ssSetOptions`)

Syntax:

```
void mdlInitializeSizes(SimStruct *S)
```

mdlOutputs

Vypočítává výstupní signály bloku. Tato povinná metoda je volána během každého simulačního kroku. Počítá výstupy s-funkce a ukládá je do výstupních polí. Identifikační číslo úlohy určuje úlohu běžící při vyvolání `mdlOutputs`.

Syntax:

```
void mdlOutputs(SimStruct *S, int_T tid)
```

Argumenty:

tid – Identifikační číslo úlohy

mdlUpdate

Aktualizuje stavy bloku. Tato volitelná metoda je volána při každém simulačním kroku. Počítá stavy s-funkce a ukládá je do stavového vektoru. Může ale vykonávat i jiné úkoly. Je použita v případě, kdy má s-funkce diskrétní stavy nebo nemá přímé řízení. Důvodem je, že většina s-funkcí, které nemají diskrétní stavy, ale mají přímé řízení, nemají aktualizací funkci. Proto je Simulink schopen odstranit potřebu pro zvláštní volání v těchto situacích. Pokud s-funkce tuto metodu vyžaduje a nevyhovuje kterákoliv z výše uvedených situací, určují se diskrétní stavy makrem `ssSetNumDiscStates` ve funkci `mdlInitializeSizes`.

Syntax:

```
void mdlUpdate(SimStruct *S, int_T tid)
```

Argumenty:

tid – Identifikační číslo úlohy

mdlTerminate

Vykonává činnosti vyžadované při ukončení simulace. Tato metoda například uvolňuje paměť. Pomocí makra `ssSetOptions` lze určit, zda Simulink vyvolá tuto metodu. Není-li tato vlastnost nastavena, Simulink vyvolá `mdlTerminate` na konci simulace pouze pokud byla dříve vyvolána také metoda `mdlStart`. Je-li nastavena, Simulink vždy vyvolá `mdlTerminate` na konci simulace.

Syntax:

```
void mdlTerminate(SimStruct *S)
```

1.4.3 Makra datové struktury SimStruct

Soubor `matlabroot/simulink/include/simstruc.h` je hlavičkový soubor jazyka C. Určuje datovou strukturu a makra zpřístupňující `SimStruct`. Zapouzdřuje všechny údaje týkající se modelu a s-funkce včetně parametrů bloku a výstupů.

V Simulinku má každý model přidělenou jednu datovou strukturu `SimStruct`. Každá s-funkce v modelu má svoji vlastní přidruženou strukturu. Uspořádání těchto struktur se podobá stromovému adresáři. Struktura přidružená k modelu se nazývá kořenová. Struktury přidružené k s-funkcím se nazývají potomci.

ssSetNumSFcnParams

V metodě `mdlInitializeSizes` určuje počet parametrů s-funkce.

Syntax:

```
ssSetNumSFcnParams(SimStruct *S, int_T nSFcnParams)
```

Argumenty:

S – SimStruct reprezentující blok s-funkce

nSFcnParams – Počet parametrů, které S obsahuje

ssSetNumContStates

V metodě `mdlInitializeSizes` určuje počet spojitých stavů bloku.

Syntax:

```
void ssSetNumContStates(SimStruct *S, int_T n)
```

Argumenty:

n – Počet spojitých stavů

ssSetNumDiscStates

V metodě `mdlInitializeSizes` určuje počet diskrétních stavů bloku. Hodnoty diskrétních stavů jsou zaznamenány, pokud je zaškrtnuta volba States v dialogovém okně Simulation Parameters (záložka Workspace I/O).

Syntax:

```
ssSetNumDiscStates(SimStruct *S, int_T nDiscStates)
```

Argumenty:

nDiscStates – Počet diskrétních stavů

ssSetNumInputPorts

V metodě `mdlInitializeSizes` určuje počet vstupních portů bloku.

Syntax:

```
void ssSetNumInputPorts(SimStruct *S, int_T nInputPorts)
```

Argumenty:

nInputPorts – Počet vstupních portů bloku S. Musí to být nezáporné celé číslo.

ssSetInputPortWidth

V metodě `mdlInitializeSizes` určuje šířku vstupního portu.

Syntax:

```
void ssSetInputPortWidth(SimStruct *S, int_T port, int_T width)
```

Argumenty:

`port` – index vstupního portu, jehož šířka bude nastavena

`width` – šířka vstupního portu (nenulové kladné celé číslo)

ssSetInputPortDirectFeedThrough

V metodě `mdlInitializeSizes` určuje přímovazební stav portů bloku. Není-li stav určen, nastaví se implicitní hodnota 0. Nastavení hodnoty 0 pro vstupní port znamená, že odpovídající signál vstupního portu není použit v `mdlOutputs` nebo v `mdlGetTimeOfNextVarHit`. V takovém případě může nastat zpoždění jednoho simulačního kroku ve vstupním signálu. To může způsobit sporné situace a chyby kvůli neslučitelnosti.

Syntax:

```
void ssSetInputPortDirectFeedThrough(SimStruct *S, int_T port, int_T dirFeed)
```

Argumenty:

`port` – index vstupního portu, jehož přímovazebnost bude nastavena

`dirFeed` – přímovazební status bloku (0 nebo 1)

ssSetNumOutputPorts

V metodě `mdlInitializeSizes` určuje počet výstupních portů bloku.

Syntax:

```
void ssSetNumOutputPorts(SimStruct *S, int_T nOutputPorts)
```

Argumenty:

`nOutputPorts` – Počet výstupních portů bloku `S`. Musí to být nezáporné celé číslo.

ssSetOutputPortWidth

V metodě `mdlInitializeSizes` určuje šířku výstupního portu.

Syntax:

```
void ssSetOutputPortWidth(SimStruct *S, int_T port, int_T width)
```

Argumenty:

port – Index výstupního portu, jehož šířka bude nastavena

width – Šířka výstupního portu (nenulové kladné celé číslo)

ssSetNumSampleTimes

V metodě `mdlInitializeSizes` určuje počet period vzorkování bloku s-funkce.

Syntax:

```
void ssSetNumSampleTimes(SimStruct *S, int_T nSampleTimes)
```

Argumenty:

nSampleTimes – Počet period vzorkování, které S má (kladné celé číslo větší než 0)

ssSetNumRWork

V `mdlInitializeSizes` určuje velikost vektoru pracujícího s plovoucí desetinnou tečkou.

Syntax:

```
void ssSetNumRWork(SimStruct *S, int_T nRWork)
```

Argumenty:

nRWork – Počet prvků v pracovním vektoru s plovoucí desetinnou tečkou (nula, kladné celé číslo nebo dynamicky určované)

ssSetNumIWork

V metodě `mdlInitializeSizes` určuje velikost vektoru pracujícího s celými čísly.

Syntax:

```
void ssSetNumIWork(SimStruct *S, int_T nIWork)
```

Argumenty:

nIWork - Počet prvků v pracovním celočíselném vektoru (nula, kladné celé číslo nebo dynamicky určované)

ssSetNumPWork

V metodě `mdlInitializeSizes` určuje velikost vektoru pracujícího s ukazateli.

Syntax:

```
void ssSetNumPWork(SimStruct *S, int_T nPWork)
```

Argumenty:

nPWork - Počet prvků, které mají být přiděleny do vektoru pracujícího s ukazateli (nula, kladné celé číslo nebo dynamicky určované)

ssSetNumModes

V metodě mdlInitializeSizes určuje velikost vektoru pracujícího s módy. Simulink alokuje tento vektor a inicializuje jeho prvky na hodnotu 0. Pokud není standardní hodnota (nula) vhodná, lze nastavit jiné počáteční hodnoty v metodě mdlInitializeConditions.

Syntax:

```
ssSetNumModes(SimStruct *S, nModes)
```

Argumenty:

nModes – Velikost vektoru (nula, kladné celé číslo nebo dynamicky určované)

ssSetNumNonsampledZCs

V metodě mdlInitializeSizes určuje počet stavů, pro které blok zjišťuje průsečíky s horizontální osou mezi dvěma vzorkovacími body.

Syntax:

```
ssSetNumNonsampledZCs(SimStruct *S, nNonsampledZCs)
```

Argumenty:

nNonsampledZCs – Počet nevzorkovaných průsečíků s horizontální osou, které blok zjišťuje (nula, kladné celé číslo nebo dynamicky určované)

ssSetOptions

V metodě mdlInitializeSizes určuje možnosti s-funkce.

Syntax:

```
void ssSetOptions(SimStruct *S, uint_T options)
```

Argumenty:

options – Možnosti (jednotlivé volby musí být spojeny operátorem OR)

ssGetSFcnParam

Získá parametr bloku s-funkce. Lze ji použít v libovolné rutině k zpřístupnění parametru zadaného v dialogovém boxu bloku s-funkce. Hodnota indexu začíná na nule a je menší než `ssGetSFcnParamsCount(S)`.

Syntax:

```
const mxArray *ssGetSFcnParam(SimStruct *S, int_T index)
```

Argumenty:

index – Index parametru, který má být vrácen

ssGetSFcnParamsCount

Získá počet parametrů zadaných v dialogovém boxu bloku s-funkce.

Syntax:

```
ssGetSFcnParamsCount(SimStruct *S)
```

ssGetNumSFcnParams

Získá počet parametrů očekávaných blokem s-funkce.

Syntax:

```
int_T ssGetNumSFcnParams(SimStruct *S)
```

ssGetErrorStatus

Získá řetězec obsahující poslední chybovou zprávu.

Syntax:

```
const char_T *ssGetContStates(SimStruct *S)
```

ssSetSampleTime

Nastavuje periodu vzorkování.

Syntax:

```
void ssSetSampleTime(SimStruct *S, st_index, time_T period)
```

Argumenty:

st_index – Index periody vzorkování, jejíž perioda bude nastavena (st_index začíná na nule)

period – Perioda vzorkování určená pomocí st_index

ssSetOffsetTime

Nastavuje ofsetový čas bloku.

Syntax:

```
ssSetOffsetTime(SimStruct *S, st_index, offset)
```

Argumenty:

st_index – Index periody vzorkování, jejíž ofset má být nastaven (st_index začíná na nule)

offset – Ofset periody vzorkování určený pomocí st_index

ssGetRealDiscStates

Získá diskrétní stavový vektor bloku. Vracený vektor obsahuje pole prvků typu real_T o délce ssGetNumDiscStates(S). Typicky je stavový vektor inicializován v mdlInitializeConditions, aktualizován v mdlUpdate a použit v mdlOutputs. Stejně vlastnosti má také makro ssGetDiscStates.

Syntax:

```
real_T *ssGetRealDiscStates(SimStruct *S)
```

ssGetNumDiscStates

Získá počet diskrétních stavů bloku.

Syntax:

```
int_T ssGetNumDiscStates(SimStruct *S)
```

ssGetInputPortRealSignal

Získá adresu reálného signálu vstupujícího na vstupní port. Makro lze použít pouze v případě reálného vstupního signálu. Zároveň musí metoda mdlInitializesizes určit prvky vstupního signálu jako souvislé za pomoci makra ssSetInputPortRequiredContiguous.

Syntax:

```
const real_T *ssGetInputPortRealSignal(SimStruct *S, inputPortIdx)
```

Argumenty:

inputPortIdx – Index portu, jehož perioda vzorkování je vyžadována

ssSetInputPortRequiredContiguous

Určuje, že prvky signálu vstupující na daný port musí zabírat souvislé oblasti v paměti. Toto dovoluje metodě zpřístupnit prvky signálu jednoduše zvýšením ukazatele signálu vráceného makrem `ssGetInputPortSignal`. S-funkce může nastavit hodnotu této vlastnosti již v metodě `mdlInitializeSizes` a nejpozději v metodě `mdlSetWorkWidths`. Standardní nastavení pro příznak je `false`. Proto je standardním makrem pro zpřístupnění vstupního signálu `ssGetInputSignalPtrs`.

Syntax:

```
void ssSetInputPortRequiredContiguous(SimStruct *S, int_T port, flag)
```

Argumenty:

port – Index vstupního portu

flag – Příznak (true nebo false)

ssGetInputPortSignal

Získá adresu souvislého signálu vstupujícího na vstupní port. Metoda by měla používat toto makro pouze v případě, kdy metoda `mdlInitializeSizes` určila, že prvky vstupního signálu jsou souvislé (použitím makra `ssSetInputPortRequiredContiguous`).

Syntax:

```
const void* ssGetInputPortSignal(SimStruct *S, inputPortIdx)
```

Argumenty:

inputPortIdx – index portu, jehož perioda vzorkování je vyžadována

ssGetOutputPortRealSignal

Získá ukazatel na výstupní signál typu `double` (`real_T`). Makro je použitelné v libovolné rutině k zpřístupnění výstupního portu, jehož index začíná na nule a musí být menší než počet výstupních portů. Vrací souvislý `real_T` vektor o délce rovnající se šířce výstupního portu.

Syntax:

```
real_T *ssGetOutputPortRealSignal(SimStruct *S, int_T port)
```

Argumenty:

port – Index výstupního portu

1.5 Přehled regulátorů a překládaných souborů

Knihovna (Knihovna.mdl) obsahuje různé typy regulátorů realizovaných jako maska subsystému. Algoritmy jednotlivých částí regulátorů jsou uloženy v m-souborech Matlabu. Názvy souborů zůstávají po překladu do jazyka C stejné, pouze se změní přípona .m na příponu .c. Při vytvoření spustitelného mex-souboru vzniká soubor s příponou .dll.

Tabulka 2. Přehled regulátorů a překládaných souborů

Regulátor	Identifikace	Parametry regulátoru	Akční zásah	Převod parametrů
1DOF	iden	reg_1dof	akcni_1dof	id_p
2DOF - A	iden	reg_2dof	akcni_2dof	id_p
2DOF - B	iden	reg_2dofok	akcni_2dofok	id_p
K1	iden_K1	reg_K1	akcni_K1	id_p_K1
K2	iden_K23	reg_K2	akcni_K2	id_p_K23
K3	iden_K23	reg_K3	akcni_K3	id_p_K23

2 PRAKTICKÁ ČÁST

Cílem této práce je naprogramovat v jazyce C stávající knihovnu samočinně se nastavujících dvourozměrových regulátorů a tím zajistit kompatibilitu této knihovny s prostředím Real – Time Workshop, které umožňuje vytvářet regulátory samostatně spustitelné na různých průmyslových zařízeních.

Takto přeložená knihovna bude ověřena simulačně i řízením laboratorního modelu. Při řízení laboratorního modelu bude knihovna spuštěna na technologické kartě dSPACE 1104 (DS1104) a průběhy řízení budou zachyceny pomocí jejího programu ControlDesk.

2.1 Technologická karta DS1104

DS1104 je standardní počítačová karta, kterou lze umístit do PCI slotu běžného počítače typu PC [6]. Tato karta je speciálně navržena pro vývoj vysokorychlostních číslicových regulátorů pracujících s mnoha proměnnými. V různých oblastech se také využívá k simulacím v reálném čase. Je to kompletní řídicí systém pracující v reálném čase. Je založen na procesoru 603 PowerPC, který pracuje na frekvenci 250 MHz. Pomocí kabelu lze spojit 100-pinový konektor umístěný na kartě s dvěma samičími 50-pinovými konektory. Tímto způsobem může karta přijímat vnější signály nebo vysílat své vlastní.



Obr. 8. Karta DS1104

Hostitelský počítač musí být standardní PC s PCI slotem (32 bitů, 5 V, 33 MHz). Procesor počítače by měl pracovat alespoň na frekvenci 450 MHz. Doporučuje se operační paměť o velikosti 256 MB RAM.

Licenční klíč se instaluje do volného paralelního portu.

Operační systém musí být Windows 98, Windows Me, Windows NT 4.0 nebo Windows 2000. Pro práci s RTI musí být nainstalován Matlab.

Pro další komplexní úkoly, jakými je např. ovládání simulací prostřednictvím ControlDesku, je třeba splnit další požadavky na systém.

Operační systém musí být Windows NT 4.0 (Service Pack 6a nebo pozdější) nebo Windows 2000. Nejsou podporovány Windows 98 ani Windows Me.

Matlab R12.1 a Simulink 4.0 (nebo pozdější) musí být nainstalovány před instalací softwaru dSPACE.

2.1.1 Instalace

Programové vybavení

Software se instaluje z příloženého CD podobně jako jiné programy v prostředí operačního systému Windows. Během instalace musí být v disketové jednotce vložena disketa obsahující licenci. Po restartu počítače se uživatel musí přihlásit s administrátorskými právy (pod Windows NT 4.0 a Windows 2000).

K ovládání simulací slouží program ControlDesk. Při jeho spuštění je zkontrolována licence v klíči, který je umístěn v paralelním portu.

Hardware

Karta DS1104 se instaluje do volného PCI slotu běžným způsobem. Do jednoho počítače lze nainstalovat i více karet DS1104. V ControlDesku lze ověřit připravenost karty k činnosti. Záložka Platform v okně Navigator zobrazuje ikonu nainstalované karty. Dvojitým kliknutím na tuto ikonu se zobrazí vlastnosti a stav karty.

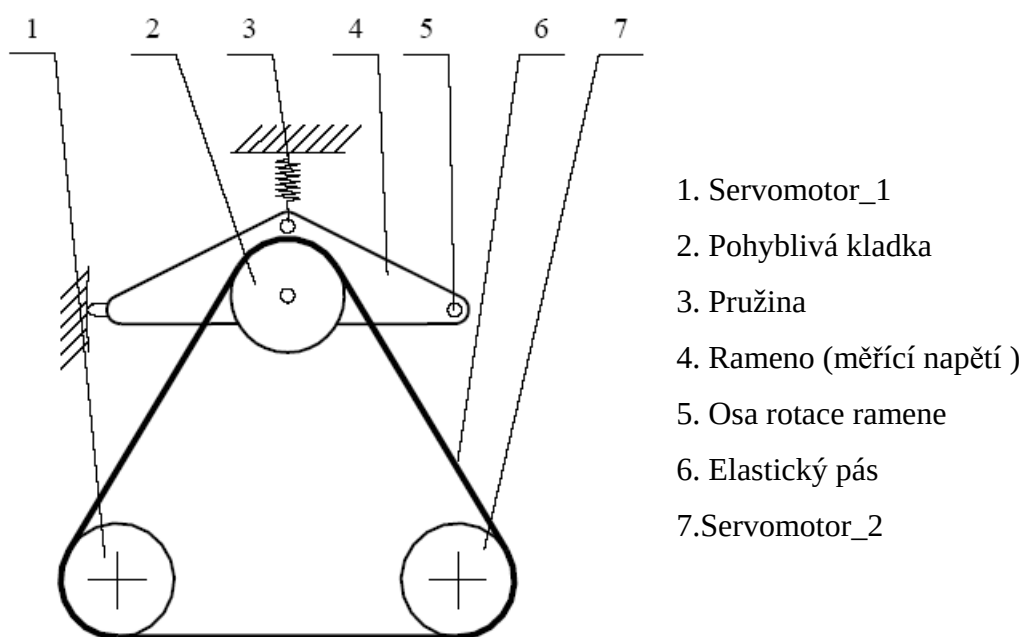
Ověření instalace

K ověření, že regulační karta DS1104 pracuje správně, je doporučeno nahrát systémovou kontrolní aplikaci `smd_1104_ch.sdf`. V ControlDesku v záložce File Selector změníme adresář na `%DSPACE_ROOT%\Demos\DS1104\Check\` a soubor popisující systém (přípona `.sdf`) uchopíme a přetáhneme na ikonu karty v okně Platform Navigator. Pokud se v záložce Log Viewer nezobrazí žádná chybová zpráva, tak je karta nainstalována správně.

Nyní je systém připraven ke spouštění aplikací na kartě DS1104.

2.2 Laboratorní model spřažených elektromotorů

Na Obrázku 9 je znázorněno schéma laboratorního výukového modelu CE 108 – coupled drives apparatus (výrobce TecQuipment Ltd., Nottingham, United Kingdom).



Obr. 9. Schéma laboratorního modelu CE – 108

Model je založen na zkušenostech se skutečnými průmyslovými aplikacemi a umožňuje zkoumat problémy souběžného řízení podélné deformace a rychlosti pásu materiálu ve spojitých technologických procesech, které se vyskytují například při výrobě papíru, drátů apod., kdy je materiál zpracováván ve formě nekonečného pásu. Ze schématu na Obrázku 9 je zřejmé, že se model skládá ze tří kladek, umístěných na vertikálním

panelu tak, aby tvořily rovnoramenný trojúhelník. Obě dolní kladky jsou nasazeny na hřídele dvou identických servomotorů, jimiž je model řízen. Třetí kladka je instalována tak, aby se mohla volně pohybovat. Modeluje tak pracovní stanici, která je vybavena senzory pro měření rychlosti a podélné deformace pružného gumového pásu, který je nasazen na kladky. Měření podélné deformace je prováděno nepřímo měřením úhlu odklonu pohyblivého ramene, na němž je umístěna třetí kladka. Rychlost a podélná deformace (úhel odklonu pohyblivého ramene) pásu jsou tedy výstupy systému a zároveň regulované veličiny. Akčními veličinami jsou potom vstupy servomotorů.

2.3 Realizace modelů

Existují dva způsoby, jak vytvářet real-time aplikace spouštěné na kartě dSPACE.

Prvním je použití Simulinku k vytváření modelů. Kód jazyka C je pak vytvářen automaticky programem Real – Time Workshop ve spolupráci s RTI karty dSPACE. RTI je rozhraní mezi Simulinkem a různými platformami dSPACE.

Druhým je algoritmus psaný ručně v jazyce C. Nezbytné kompilační nástroje nutné k vytváření objektových souborů jsou částí softwaru dSPACE.

2.3.1 Vytvoření a spuštění aplikace

Existující model (.mdl) se otevře v Matlabu. V nabídce okna modelu se zvolí možnost Tools – Real – Time Workshop – Options k zobrazení parametrů simulace. Na záložce Real – Time Workshop se ověří nastavení hodnot: System target file na rti1104.tlc, Template makefile na rti1104.tmf a Make command na make_rti. Stiskem tlačítka Build se aplikace vytvoří, přenesení se na DS1104 a automaticky se spustí. Zároveň vzniká SDF soubor (přípona .sdf), který obsahuje všechny potřebné informace o vytvořené aplikaci. K zastavení nebo opětovnému spuštění se používá ControlDesk. K sledování proměnných běžící real-time aplikace lze využít ControlDesk. Obsahuje například nástroj plotter, který po připojení k požadovaným proměnným vykresluje graf.

Již vytvořená aplikace může být nahrána do DS1104 a spuštěna opět pomocí ControlDesku použitím jejího .sdf souboru. Takto se vyhneme zdlouhavé kompilaci. Jsou-li ale v modelu Simulinku provedeny změny, je opětovná kompilace samozřejmě nutná.

Aplikace může být spuštěna nahráním do globální paměti nebo flash paměti karty. Rychlost spuštění je v obou případech stejná.

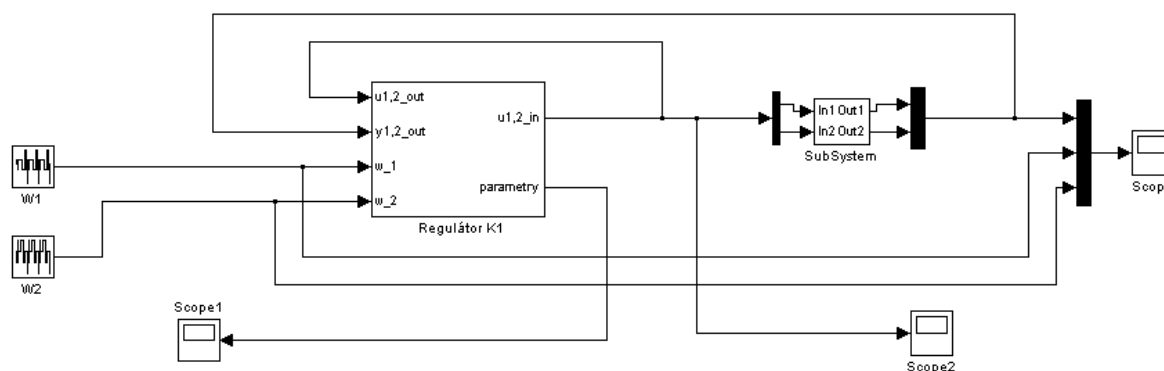
Při zapnutí nebo restartování hostitelského počítače karta ověří, není-li ve flash paměti uložena aplikace. Pokud ano, spustí ji.

Při vypnutí nebo restartování hostitelského počítače dojde k vymazání aplikace uložené v globální paměti.

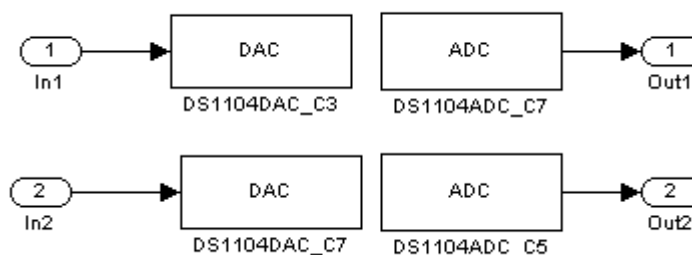
Při nahrání aplikace do flash paměti je zkopírována také do globální paměti a pak spuštěna. Dřívější aplikace v globální paměti se přepíše.

2.4 Připojení vnějších zařízení

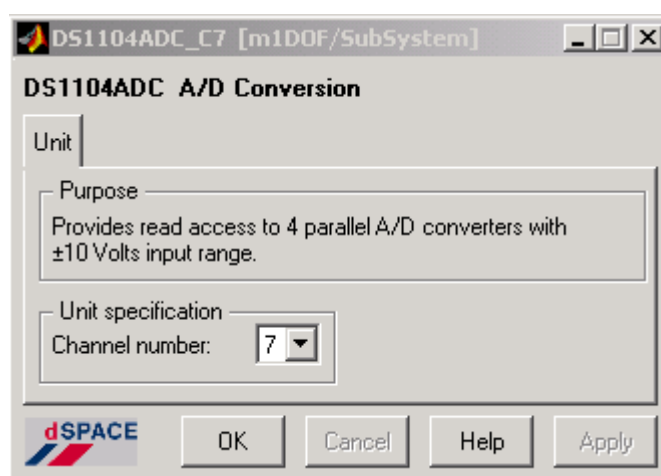
Vnější zařízení se připojují pomocí 100-pinového konektoru P1. Jeho prostřednictvím lze získat přístup k vstupním i výstupním signálům. Použitím kabelu dodávaného s kartou lze tento konektor P1 spojit s dvěma samičími 50-pinovými konektory P1A a P1B. V modelu tento přístup zajišťuje blok SubSystems (Obrázek 10). Jeho detail na Obrázku 11 obsahuje označení jednotlivých vstupních a výstupních kanálů. Propojení sledovaných a řízených veličin modelu s kartou pomocí konektoru P1A popisuje Tabulka 3. Karta obsahuje převodníky v rozsahu -10 V až $+10\text{ V}$ (Obrázek 12).



Obr. 10. Model soustavy obsahující blok pro komunikaci s laboratorním modelem



Obr. 11. Detail bloku určujícího kanály pro komunikaci s laboratorním modelem



Obr. 12. A/D převodník

Tabulka 3. Přiřazení signálů k pinům konektoru P1A

Pin	Signál	Označení kanálu	Prvek modelu	Veličina
48	ADCH7	DS1104ADC_C7	Otáčky kola	y_1
16	ADCH5	DS1104ADC_C5	Napínání pásu	y_2
14	DACH3	DS1104DAC_C3	Motor 1	u_1
29	DACH7	DS1104DAC_C7	Motor 2	u_2
1	GND		Zem	

2.5 Simulační ověření

Činnost regulátorů byla simulačně ověřena na modelu s diskrétní řízenou soustavou. Perioda vzorkování byla nastavena na 0.25 s. Parametry řízené soustavy:

$$\mathbf{A}(z^{-1}) = [0.3, 0.1, 0.1, 0.2, 0.1, 0.3, 0.3, 0.1]$$

$$\mathbf{B}(z^{-1}) = [0.1, 0.4, 0.9, 0.4, 0.6, 0.2, 0.3, 0.4]$$

Počáteční odhady parametrů modelu:

Parametry a: [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8]

Parametry b: [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8]

V případě diagonální matice A:

Parametry a: [0.1, 0.2, 0.3, 0.4]

Parametry matice $M(z^{-1})$ pro regulátor 1DOF a 2DOF - B:

$$M(z^{-1}) = [-0.9, 0.19, -0.009, -0.002, -0.9, 0.19, -0.009, 0.002]$$

Parametry matice $M(z^{-1})$ pro regulátor 2DOF - A:

$$M(z^{-1}) = [-0.4, 0.06, -0.004, 0.0001, -0.408, 0.06, 0.004, 0.0001]$$

Parametry matice $M(z^{-1})$ pro regulátor K1:

$$M(z^{-1}) = [-0.7, 0.01, -0.1, 0.05, 0.0001, -0.7, 0.01, -0.1, 0.05, 0.0001]$$

Parametry se zadávají pomocí dialogového okna regulátoru (Obrázek 13).

Block Parameters: Regulátor - 1DOF

Regulátor typu 1DOF (mask)

Parameters

Perioda vzorkování

[0.25]

Parametry matice M

[-0.9 0.19 -0.009 -0.002 -0.9 0.19 -0.009 0.002]

Odhad parametrů_a

[-0.5827 0.1745 -0.022 0.1797 0.0167 -0.0886 -0.4564 -0.0830]

Odhad parametrů_b

[-0.0035 0.0955 0.1484 0.2197 0.2783 0.3107 -0.0371 -0.3489]

Vstupní parametry Matice_C

[1e1*eye(8)]

Vstupní parametry_fi

[1 1]

Vstupní parametry_rho

[0.99 0.99]

Vstupní parametry_lambda

[0.001 0.001]

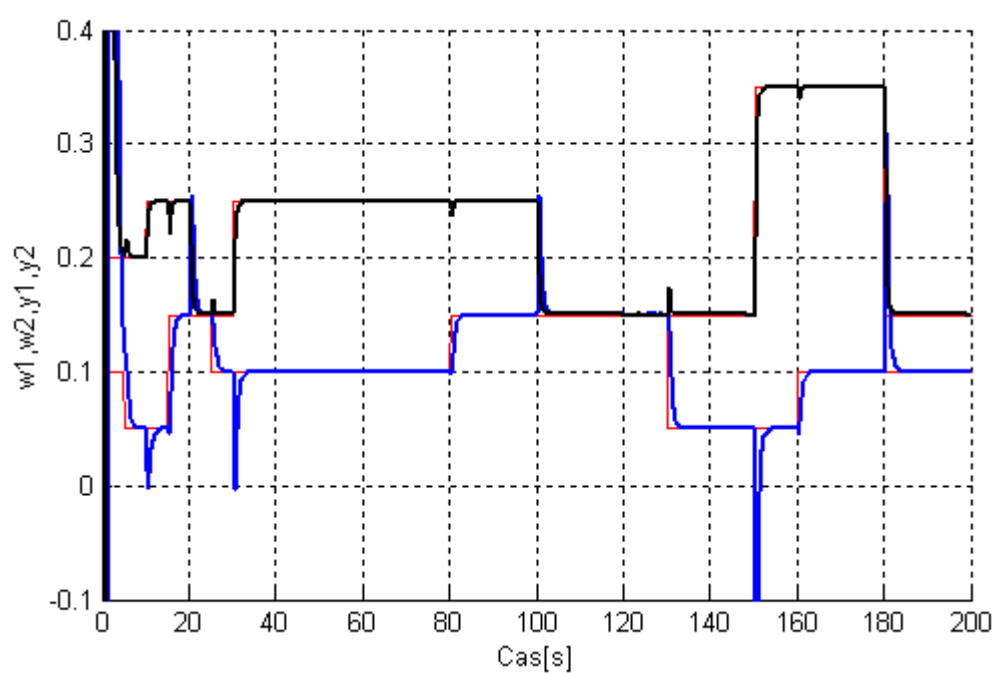
Vstupní parametry_ny0

[1e-6 1e-6]

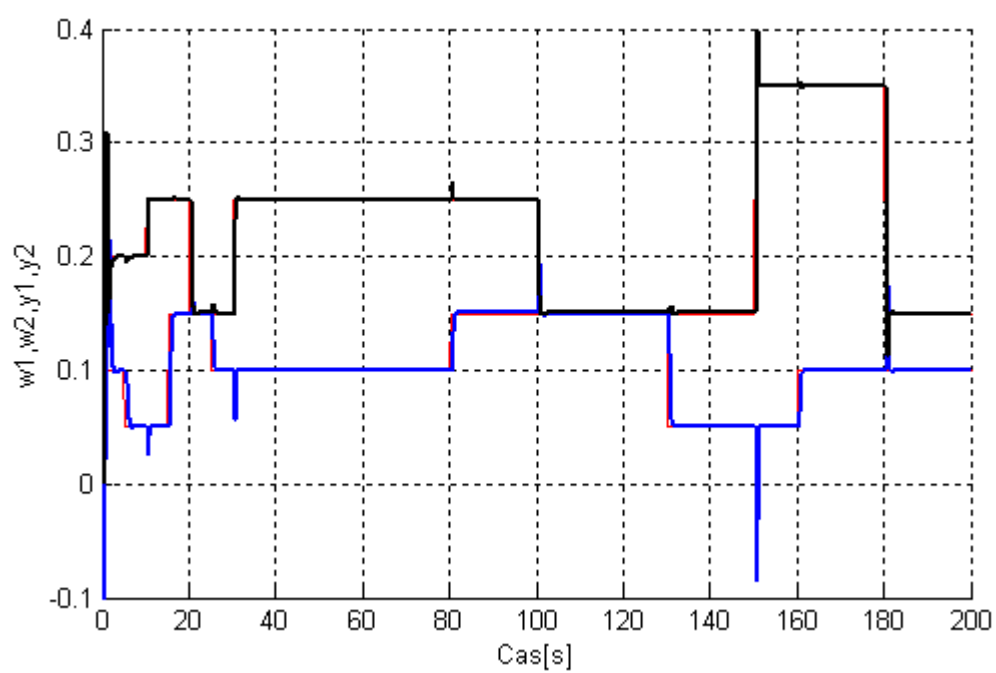
OK Cancel Help Apply

Obr. 13. Dialogové okno regulátoru pro zadávání parametrů

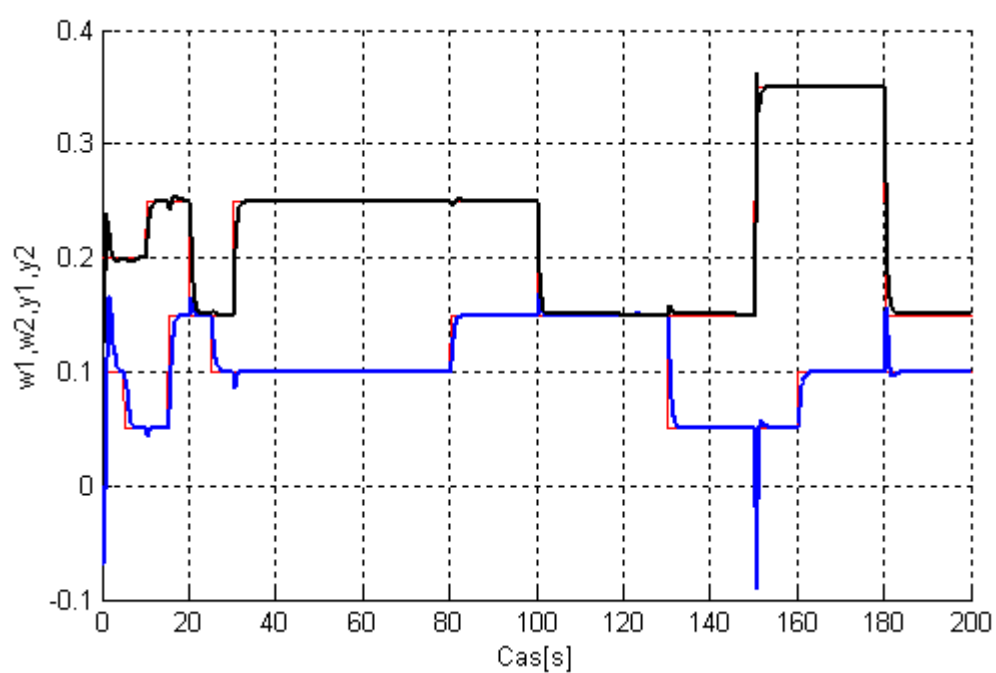
Průběhy simulací řízení jsou na následujících obrázcích (Obrázek 14 až 17).



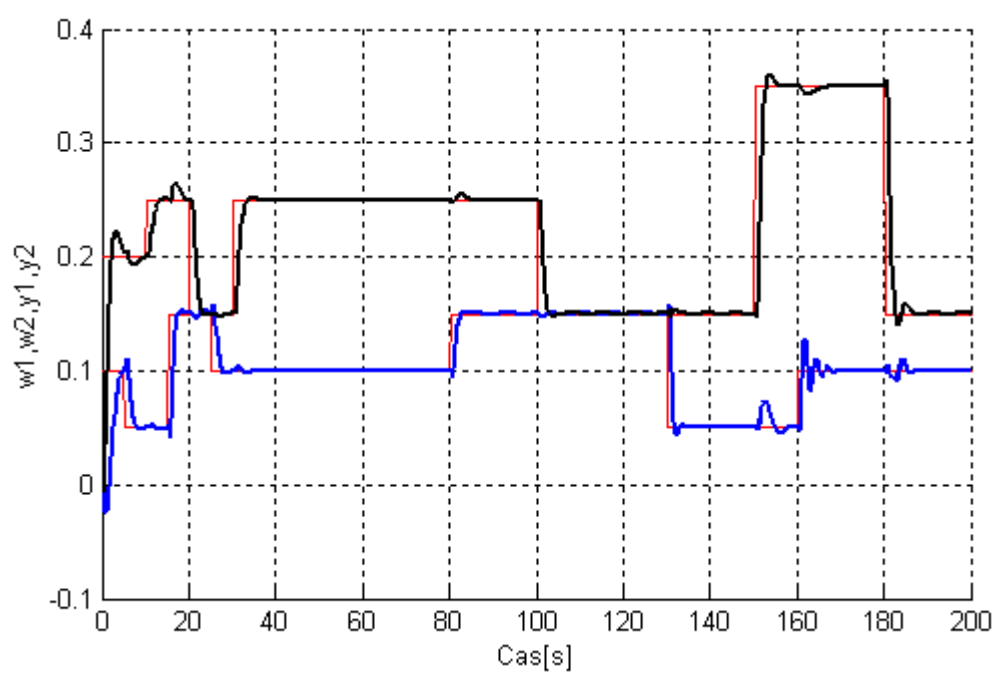
Obr. 14. Simulace řízení s regulátorem 1DOF



Obr. 15. Simulace řízení s regulátorem 2DOF - A



Obr. 16. Simulace řízení s regulátorem 2DOF - B



Obr. 17. Simulace řízení s regulátorem K1

2.6 Řešení systému lineárních rovnic maticovou inverzí

Při výpočtu parametrů regulátoru se řeší soustava lineárních rovnic. V maticovém vyjádření lze soustavu zapsat takto:

$$A \cdot X = B \quad (1.77)$$

Kde matice A je čtvercová o rozměrech 8×8 .

V Matlabu lze soustavu rovnic řešit pomocí maticové inverze. Zápis má tvar

$$X = \text{inv}(A) \cdot B \quad (1.78)$$

Kde matice X o rozměrech 8×1 obsahuje hledané řešení.

Jazyk C v prostředí Real – Time Workshop ale neobsahuje funkce pro operace s maticemi. Běžně používané algoritmy pro výpočet inverzní matice jsou snadno použitelné pouze pro matice do rozměru 3×3 . Inverze matice se ale nemusí počítat, pokud k řešení soustavy rovnic použijeme Gaussovu eliminační metodu [9]. Její princip spočívá v tom, že v rozšířené matici soustavy dostáváme nuly pod diagonálou postupně po sloupcích. Tuto metodu lze využít i pro případ dělení matice zleva. Její princip:

- 1) Najdeme ve sloupci x_1 největší číslo v absolutní hodnotě
- 2) Celý řádek, který obsahuje hlavní prvek, vydělíme hodnotou hlavního prvku
- 3) Pomocí upraveného hlavního řádku eliminujeme neznámou x_1 ze zbývajících řádků
- 4) Postup opakujeme

Zápis v jazyce C , kde proměnná n představuje počet neznámých, je v Příloze 2.

2.7 Řízení laboratorního modelu v reálném čase

Ověření knihovny v reálném čase proběhlo s využitím laboratorního modelu CE-108. Perioda vzorkování byla nastavena na hodnotu 0.25s.

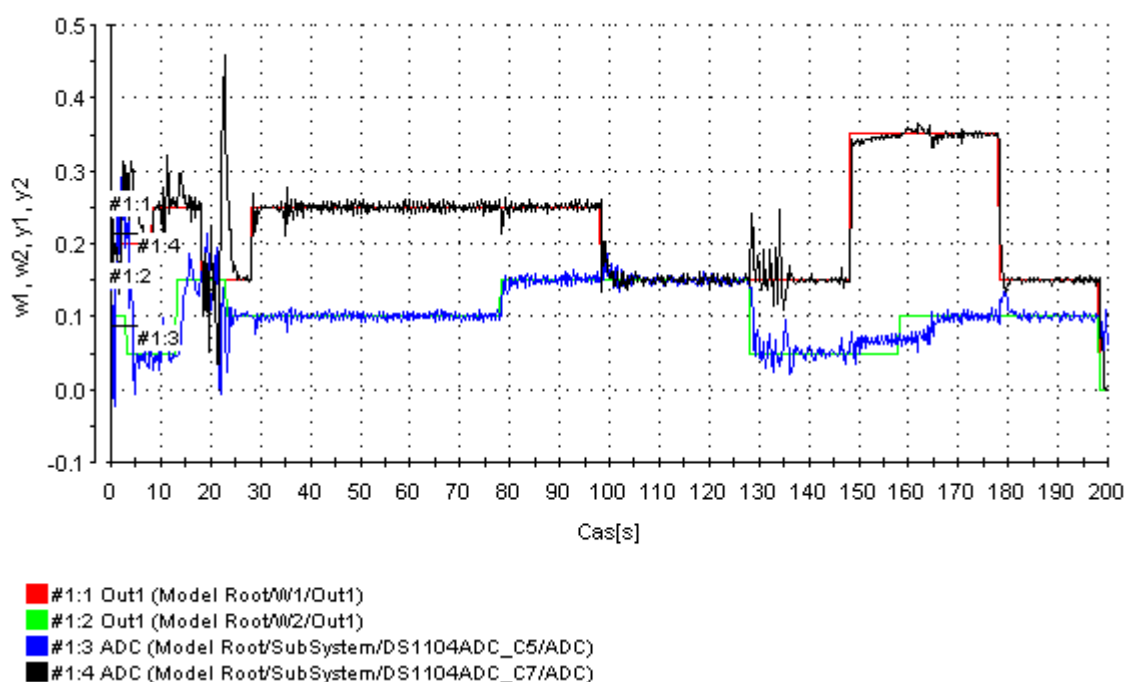
2.7.1 Snímání měřených dat pomocí ControlDesku

ControlDesk je software dodávaný s kartou dSPACE [6]. Poskytuje všechny funkce potřebné k řízení, sledování a automatizování experimentů. Všechny vstupní i výstupní signály vytvořené v Simulinku jsou po načtení .sdf souboru do karty dSPACE dostupné i v ControlDesku. Uživatel je může snadno propojit s vestavěným vizualizačním nástrojem, jakým je například nástroj Plotter, který v reálném čase vykresluje na obrazovku graf měřených veličin. Následující grafy byly zaznamenány pomocí tohoto nástroje. Naměřená data byla také pomocí okna Capture Setting Window uložena do souborů typu .csv a .mat. Soubory obsahují automaticky generovaný popis experimentu a vlastní data: čas, hodnoty regulovaných veličin, akčních veličin a žádaných veličin.

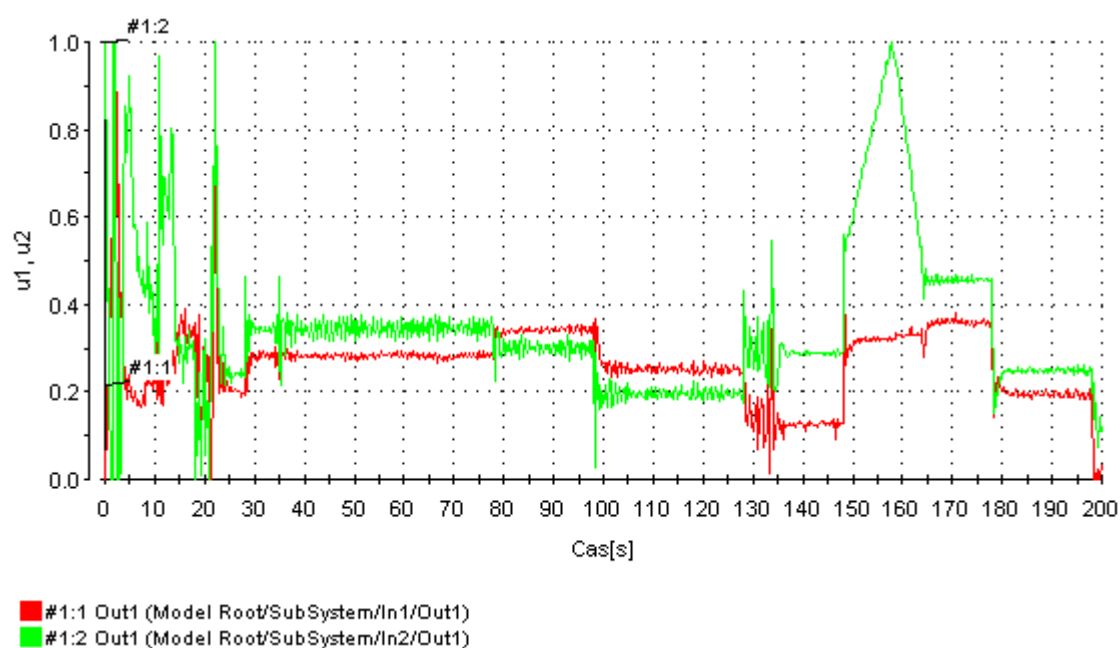
2.7.2 Použití počátečních odhadů modelu bez apriorní informace

Nastavení počátečních odhadů parametrů modelu procesu bylo stejné jako u simulací, tedy bez znalosti jakýchkoli apriorních informací o řízeném systému.

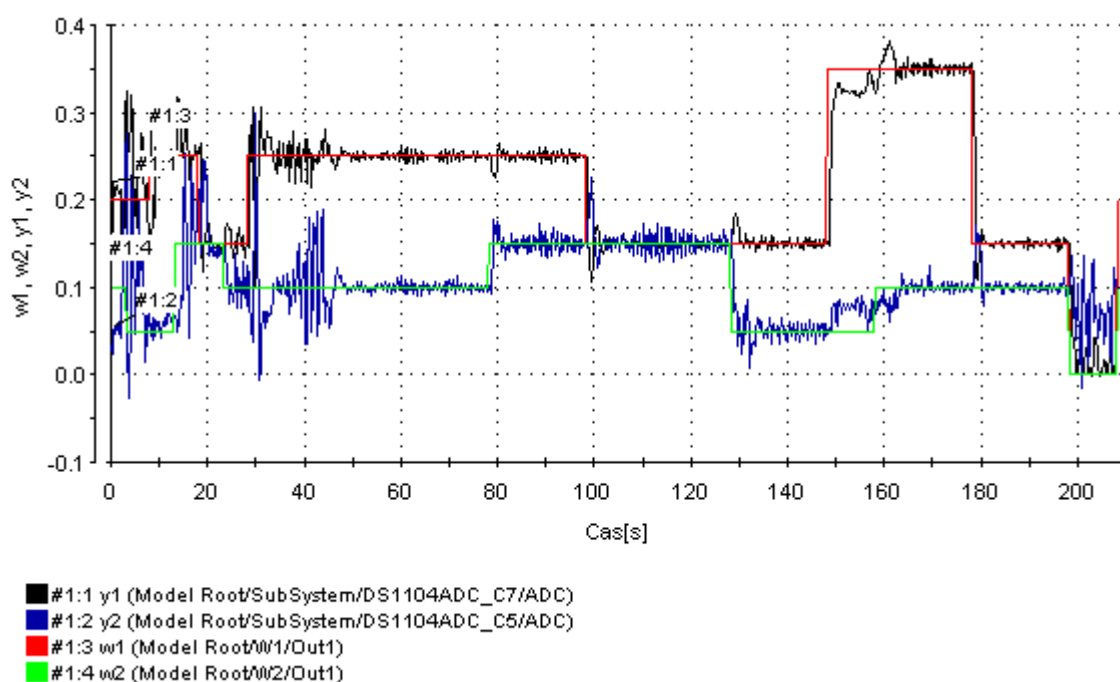
Byly zaznamenány následující průběhy řízení a akčních veličin (Obrázek 18 až 25).



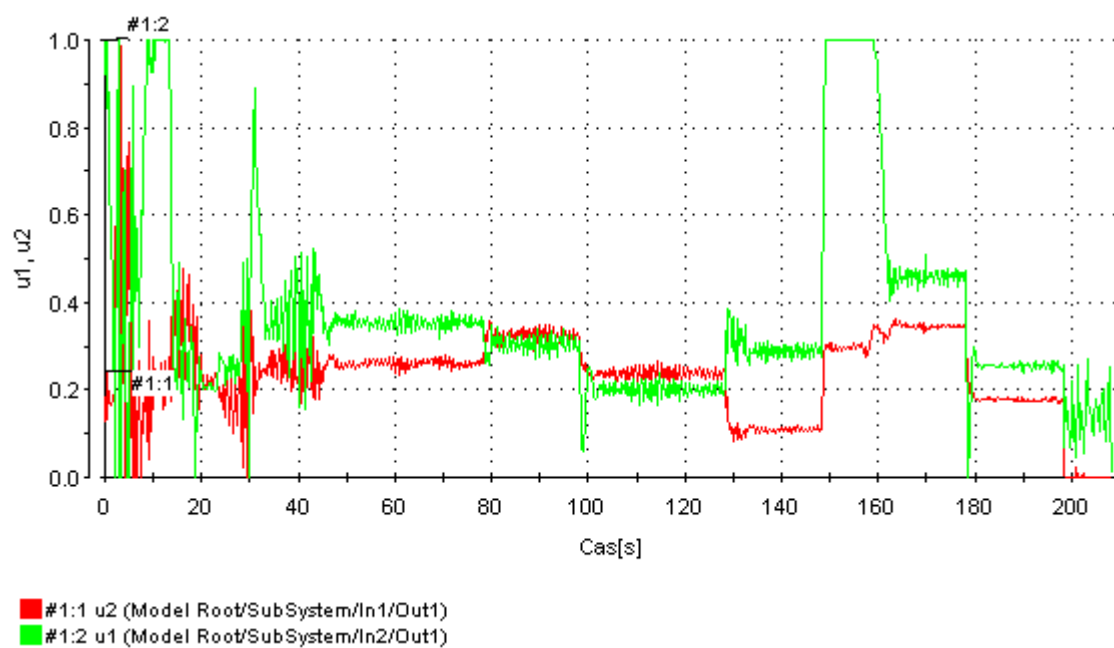
Obr. 18. Řízení laboratorního modelu pomocí regulátoru 1DOF



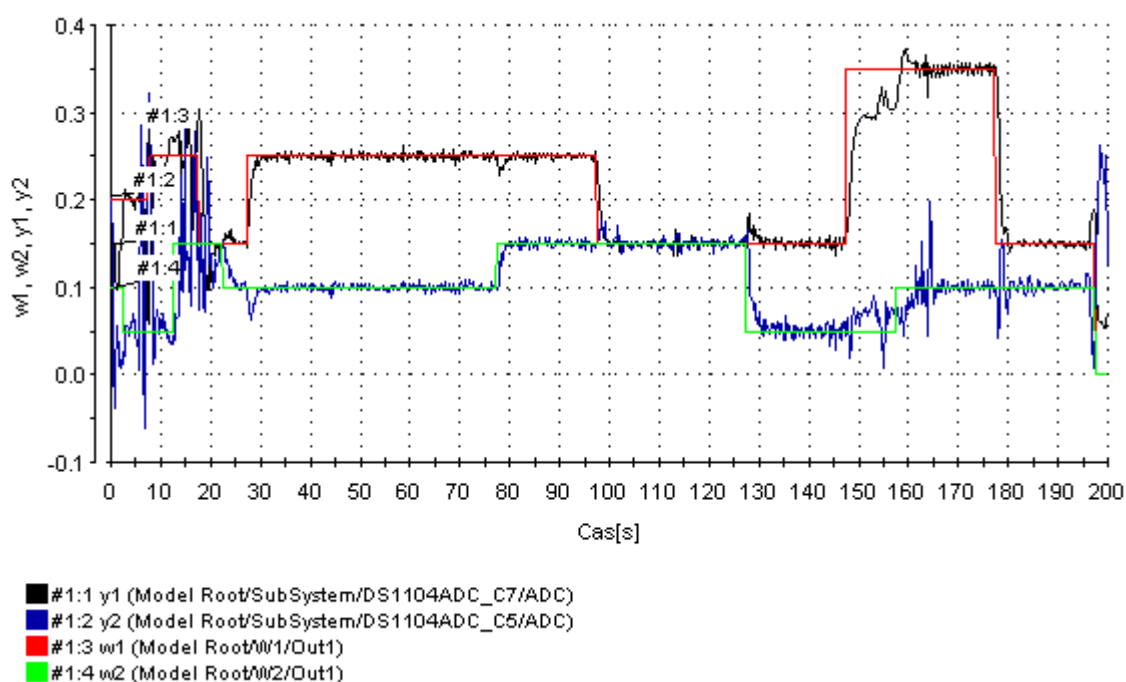
Obr. 19. Průběh akčních veličin při řízení modelu pomocí regulátoru 1DOF



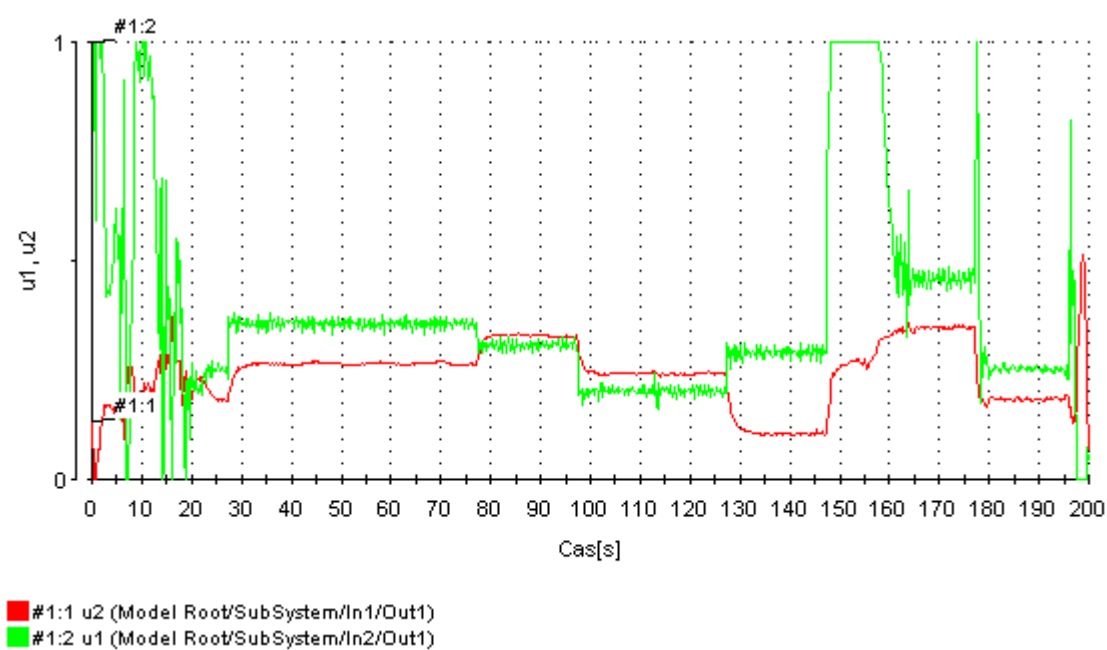
Obr. 20. Řízení laboratorního modelu pomocí regulátoru 2DOF - A



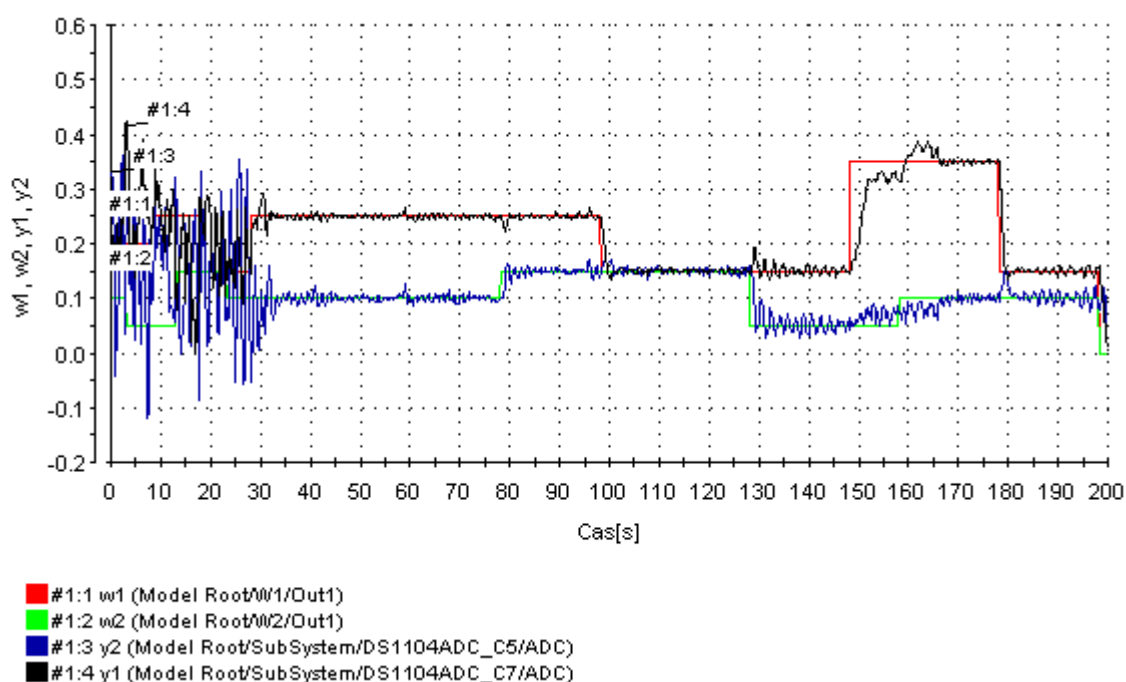
Obr. 21. Průběh akčních veličin při řízení modelu pomocí regulátoru 2DOF - A



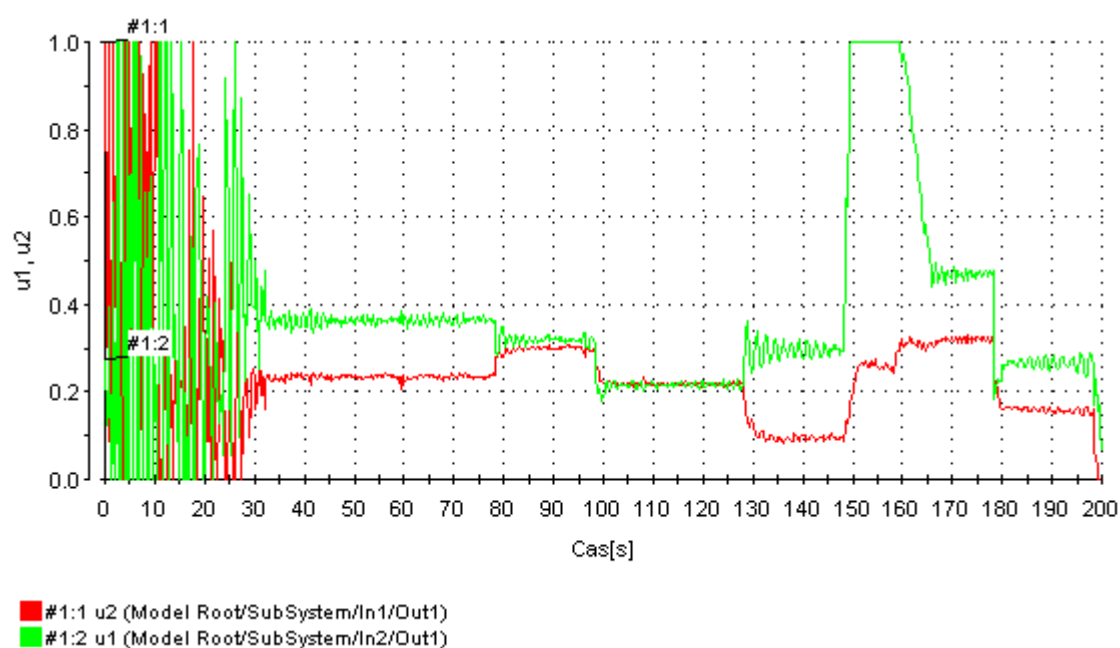
Obr. 22. Řízení laboratorního modelu pomocí regulátoru 2DOF - B



Obr. 23. Průběh akčních veličin při řízení modelu pomocí regulátoru 2DOF - B



Obr. 24. Řízení laboratorního modelu pomocí regulátoru K1



Obr. 25. Průběh akčních veličin při řízení modelu pomocí regulátoru K1

2.7.3 Použití ustálených parametrů z předchozích experimentů

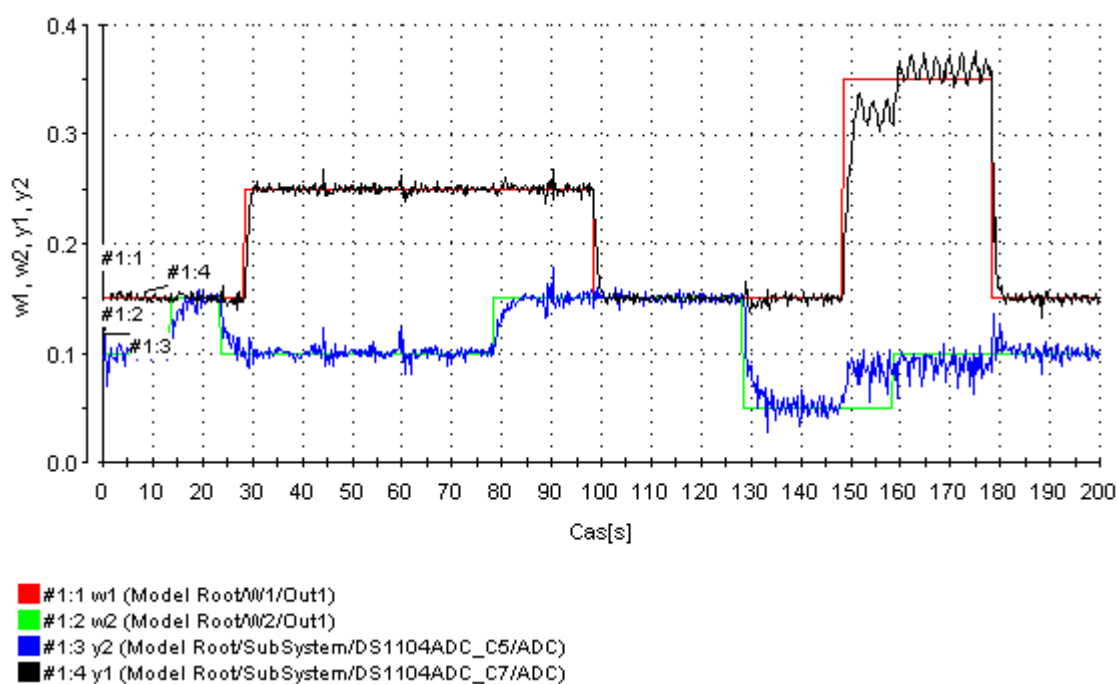
Nastavení počátečních odhadů parametrů bylo provedeno na základě ustálených parametrů z předchozího experimentu. Algoritmus průběžné identifikace byl rovněž změněn redukcí počátečních odhadů diagonálních prvků čtvercové kovarianční matice z 1000 na 10. Referenční signály byly nastaveny na stejnou hodnotu jako na konci předchozího experimentu, protože soustava je nelineární a odhady parametrů jsou platné jenom pro tyto ustálené stavy.

Počáteční odhady parametrů modelu byly změněny na:

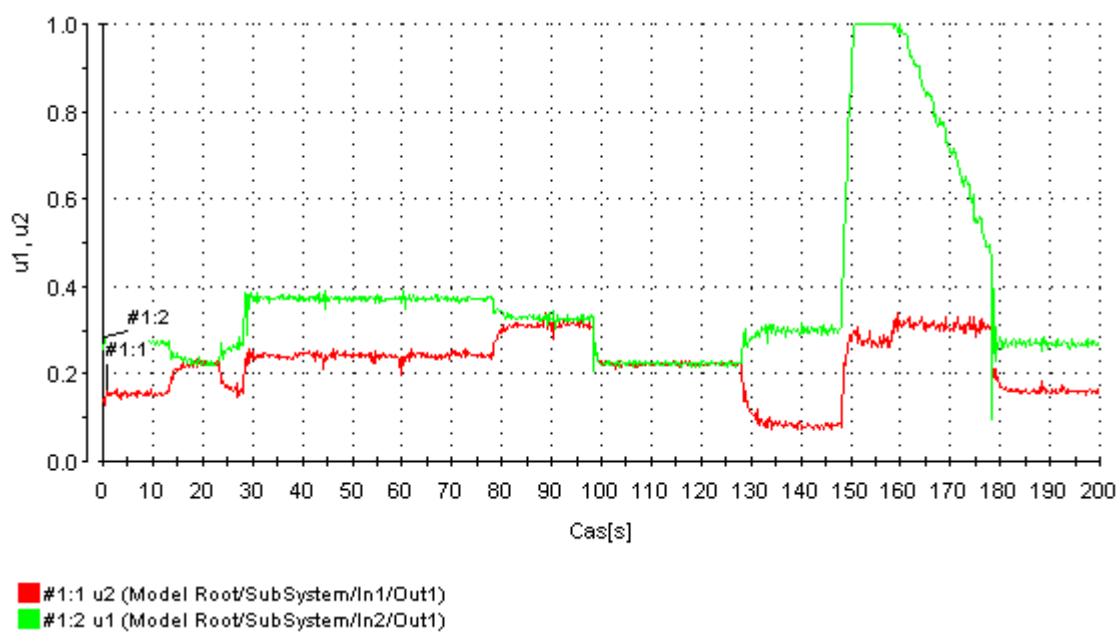
Parametry a: [-0.5827, 0.1745, -0.022, 0.1797, 0.0167, -0.0886, -0.4564, -0.0830]

Parametry b: [-0.0035, 0.0955, 0.1484, 0.2197, 0.2783, 0.3107, -0.0371, -0.3489]

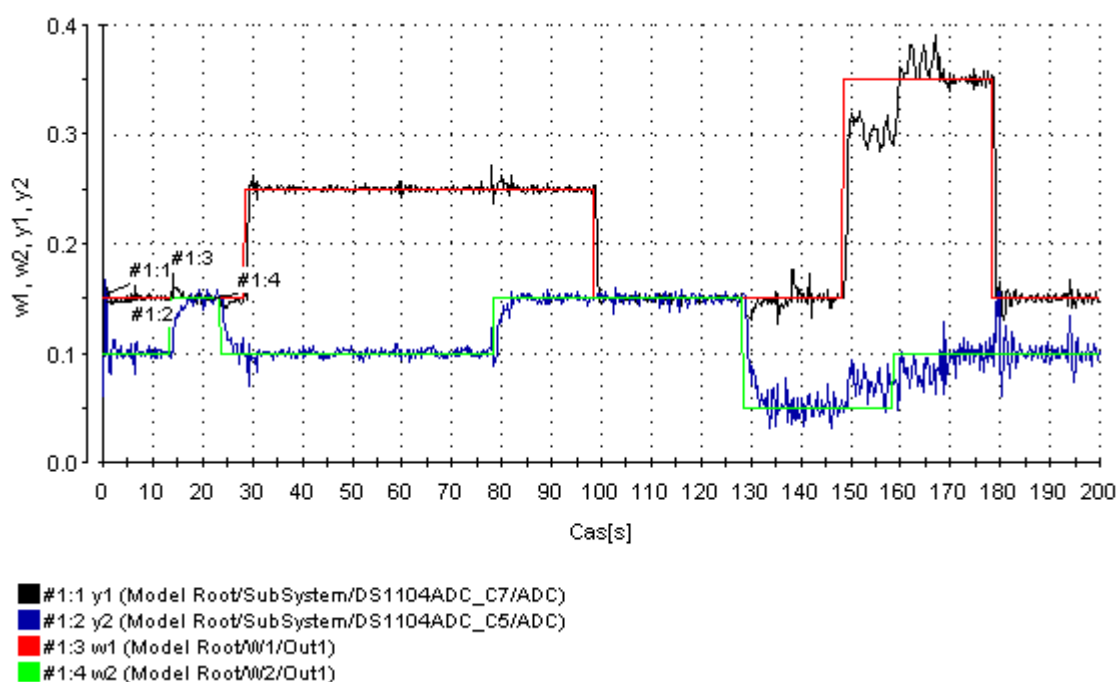
Byly zaznamenány následující průběhy řízení a akčních veličin (Obrázek 26 až 33). Dokazují zlepšení průběhů akčních veličin i regulovaných veličin.



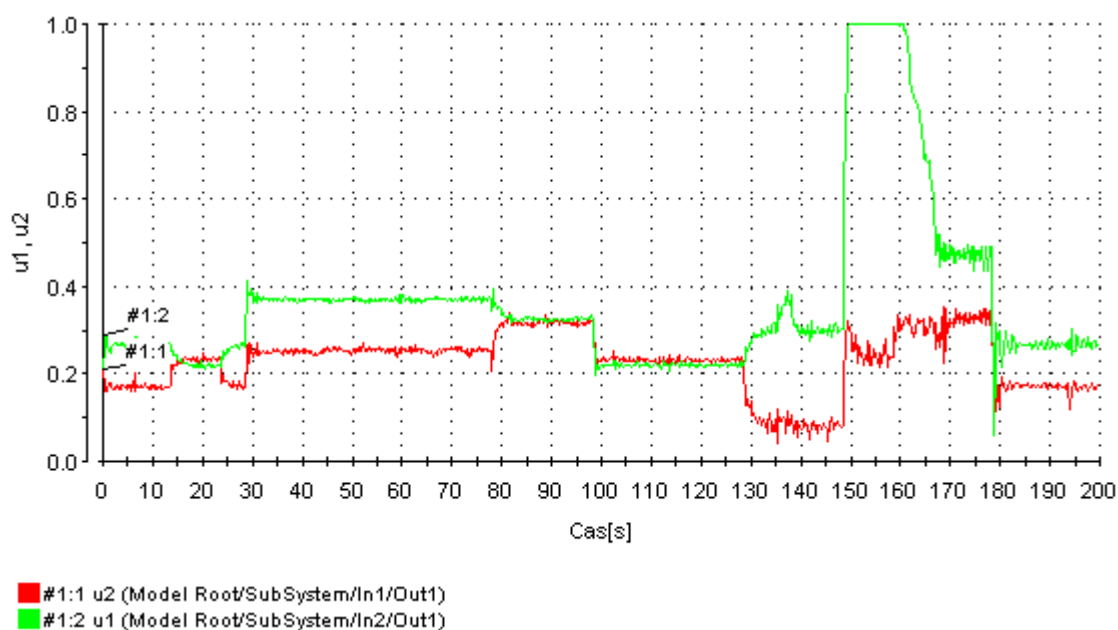
Obr. 26. Řízení laboratorního modelu pomocí regulátoru 1DOF



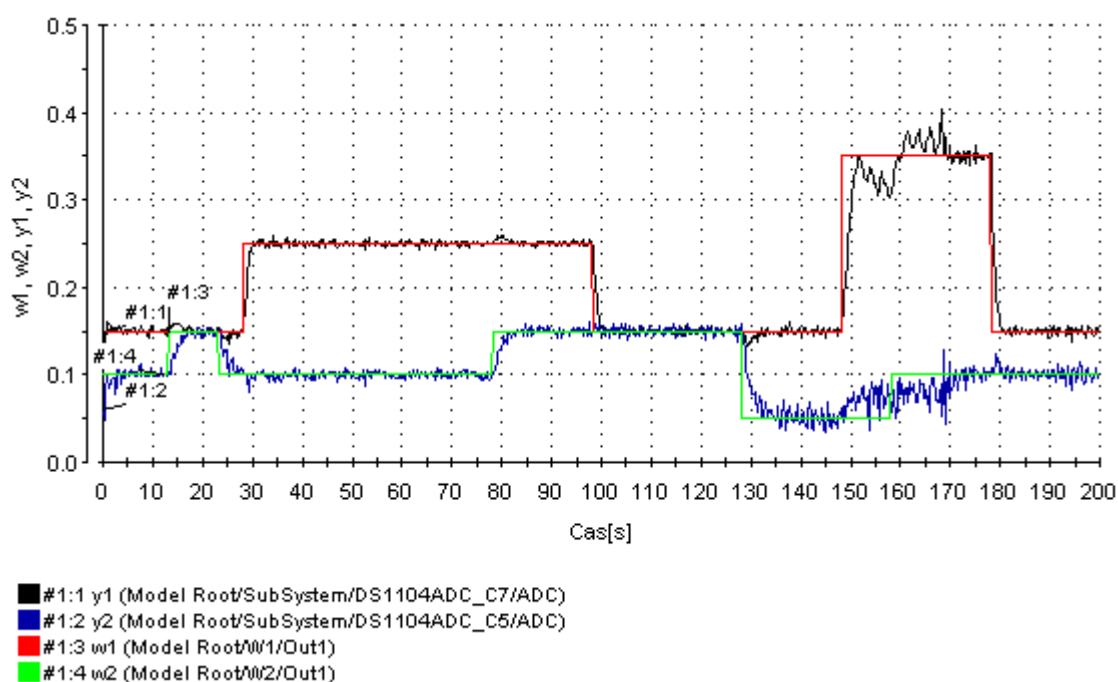
Obr. 27. Průběh akčních veličin při řízení modelu pomocí regulátoru 1DOF



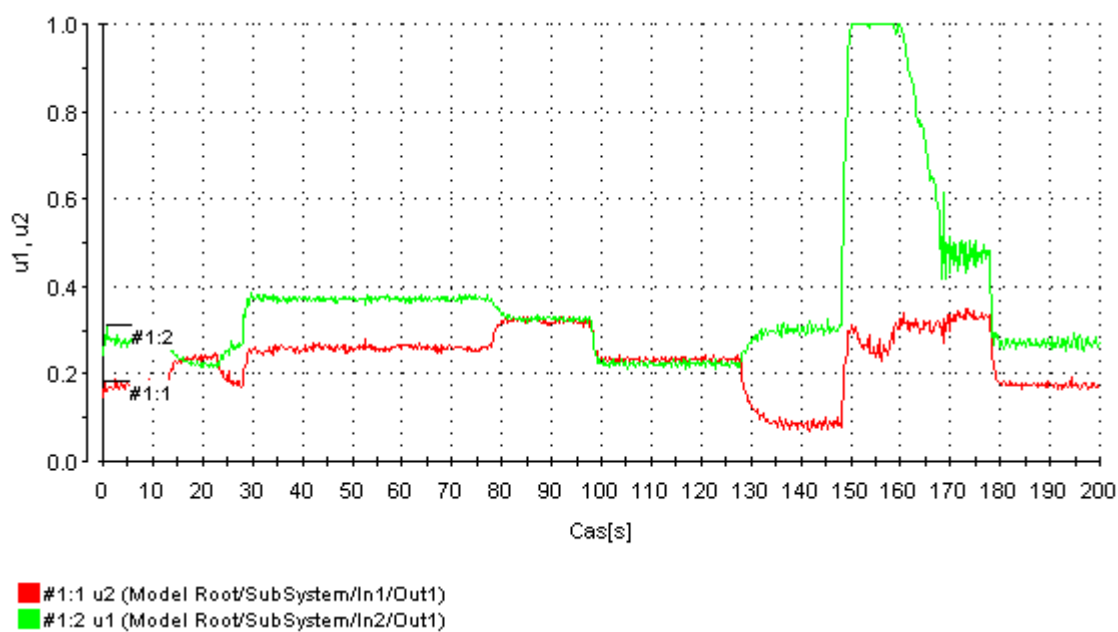
Obr. 28. Řízení laboratorního modelu pomocí regulátoru 2DOF - A



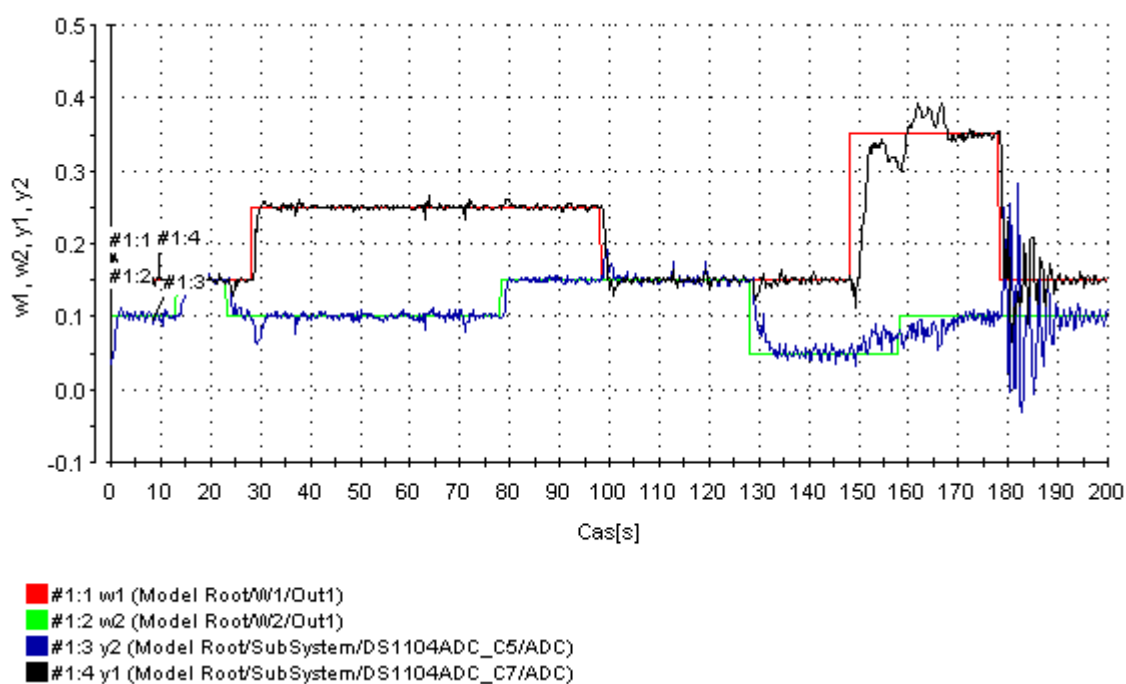
Obr. 29. Průběh akčních veličin při řízení modelu pomocí regulátoru 2DOF - A



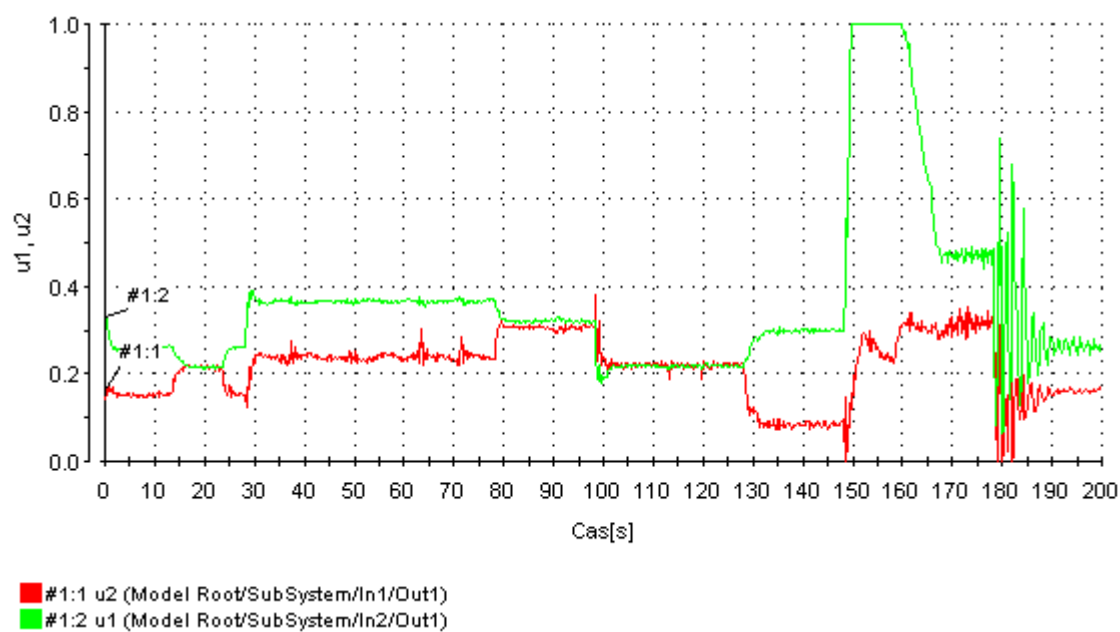
Obr. 30. Řízení laboratorního modelu pomocí regulátoru 2DOF - B



Obr. 31. Průběh akčních veličin při řízení modelu pomocí regulátoru 2DOF - B



Obr. 32. Řízení laboratorního modelu pomocí regulátoru K1



Obr. 33. Průběh akčních veličin při řízení modelu pomocí regulátoru K1

ZÁVĚR

Cílem této diplomové práce bylo zajištění kompatibility existující knihovny samočinně se nastavujících diskrétních regulátorů se systémem Real – Time Workshop. Knihovna umožňuje návrh a ověřování regulátorů pro systémy se dvěma vstupy a dvěma výstupy. Části regulátorů jsou vytvořeny jako s-funkce napsané v kódu Matlabu.

Kompatibilita byla zajištěna naprogramováním všech částí regulátorů v jazyce C. Rozsah těchto zdrojových textů je samozřejmě podstatně větší než u zdrojových textů vytvořených pomocí programovacího jazyka Matlabu, kde je možnost snadného provádění maticových a vektorových operací a není třeba vytvářet deklarace proměnných. Rozdíl je patrný především u bloku průběžné identifikace a při výpočtu parametrů regulátorů, kde se vyskytuje řada maticových operací. Tímto způsobem upravené regulátory je možné vložit do simulinkových schémat a pak generovat kód spustitelný například na průmyslových počítačích a programovatelných automatech. Právě tyto systémy často zajišťují řízení průmyslových procesů.

Jednotlivé regulátory byly spouštěny jako samostatná aplikace na technologické kartě dSPACE DS1104. V reálném čase byl řízen laboratorní model spřažených elektromotorů CE-108 obsahující dva vstupy a dva výstupy. Tímto způsobem byla potvrzena možnost vytvářet široké spektrum průmyslových aplikací.

Průběhy regulovaných veličin a akčních veličin byly měřeny a zobrazovány pomocí programu ControlDesk dodávaného s kartou dSPACE.

SEZNAM POUŽITÉ LITERATURY

- [1] BOBÁL, V., BÖHM, J., PROKOP, R. A FESSL, J. *Praktické aspekty samočinně se nastavujících regulátorů: algoritmy a implementace*. Vysoké učení technické v Brně, 1999.
- [2] BOBÁL, V. A CHALUPA, P. Knihovna samočinně se nastavujících regulátorů pro prostředí Matlab/Simulink(II). *Automatizace*, 2003, č. 9, s. 592-598.
- [3] MACA, M. *Vytvoření knihovny samočinně se nastavujících dvourozměrových regulátorů*. Diplomová práce. Univerzita Tomáše Bati ve Zlíně, 2004.
- [4] KUBALČÍK, M. *Mnohorozměrové adaptivní řízení*. Disertační práce. Vysoké učení technické v Brně, 2000.
- [5] THE MATHWORKS, INC. *Matlab Help*, 2000.
- [6] dSPACE, Inc. *Experiment Guide*, 2002.
- [7] KADLEC, V. *Učíme se programovat v jazyce C*. Computer Press, 2002.
- [8] Peng, Y. *A General Decoupling Precompensator for Linear Multivariable Systems with Application to Adaptive Control*. IEEE Trans. Aut. Control, Vol. 35, No. 3, 1990, pp. 344-348.
- [9] Rektorys, K. *Přehled užití matematiky*. Státní nakladatelství technické literatury, 1981.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

y	regulovaná veličina
u	akční veličina
e	regulační odchylka
w	žádaná veličina
T_v	perioda vzorkování
X	matice, vektor
$X(z^{-1})$	matice, vektor v diskretním popisu
U	vektor akčních veličin
W	vektor žádaných veličin
Y	vektor regulovaných veličin
G	matice přenosových funkcí
R	matice přenosových funkcí regulátoru
M	matice rozložení pólů
B_x	adjugovaná matice
deg	stupeň
det	determinant
DOF	stupně volnosti
PLC	Programable Logic Controller
IPC	Industrial Personal Computer
RAM	Random Access Memory
RTI	Real-Time Interface
SDF	System Description File
PCI	Peripheral Component Interconnect

SEZNAM OBRÁZKŮ

Obr. 1. Schéma konfigurace 1DOF	9
Obr. 2. Schéma konfigurace 2DOF - A	11
Obr. 3. Schéma konfigurace 2DOF – B.....	13
Obr. 4. Schéma regulátoru s kompenzátorem.....	14
Obr. 5. Regulační obvod s kompenzátorem K1.....	15
Obr. 6. Regulační obvod s kompenzátorem K2.....	16
Obr. 7. Regulační obvod s kompenzátorem K3.....	17
Obr. 8. Karta DS1104	41
Obr. 9. Schéma laboratorního modelu CE – 108.....	43
Obr. 10. Model soustavy obsahující blok pro komunikaci s laboratorním modelem.....	45
Obr. 11. Detail bloku určujícího kanály pro komunikaci s laboratorním modelem	46
Obr. 12. A/D převodník	46
Obr. 13. Dialogové okno regulátoru pro zadávání parametrů	48
Obr. 14. Simulace řízení s regulátorem 1DOF	49
Obr. 15. Simulace řízení s regulátorem 2DOF - A	49
Obr. 16. Simulace řízení s regulátorem 2DOF - B	50
Obr. 17. Simulace řízení s regulátorem K1	50
Obr. 18. Řízení laboratorního modelu pomocí regulátoru 1DOF.....	53
Obr. 19. Průběh akčních veličin při řízení modelu pomocí regulátoru 1DOF	53
Obr. 20. Řízení laboratorního modelu pomocí regulátoru 2DOF - A.....	54
Obr. 21. Průběh akčních veličin při řízení modelu pomocí regulátoru 2DOF - A	54
Obr. 22. Řízení laboratorního modelu pomocí regulátoru 2DOF - B.....	55
Obr. 23. Průběh akčních veličin při řízení modelu pomocí regulátoru 2DOF - B	55
Obr. 24. Řízení laboratorního modelu pomocí regulátoru K1.....	56
Obr. 25. Průběh akčních veličin při řízení modelu pomocí regulátoru K1.....	56
Obr. 26. Řízení laboratorního modelu pomocí regulátoru 1DOF.....	58
Obr. 27. Průběh akčních veličin při řízení modelu pomocí regulátoru 1DOF	58
Obr. 28. Řízení laboratorního modelu pomocí regulátoru 2DOF - A.....	59
Obr. 29. Průběh akčních veličin při řízení modelu pomocí regulátoru 2DOF - A	59
Obr. 30. Řízení laboratorního modelu pomocí regulátoru 2DOF - B.....	60
Obr. 31. Průběh akčních veličin při řízení modelu pomocí regulátoru 2DOF - B	60
Obr. 32. Řízení laboratorního modelu pomocí regulátoru K1.....	61
Obr. 33. Průběh akčních veličin při řízení modelu pomocí regulátoru K1.....	61

SEZNAM TABULEK

Tab. 1 Obsah m-souboru.....	27
Tab. 2 Přehled regulátorů a překládaných souborů.....	40
Tab. 3 Přiřazení signálů k pinům konektoru P1A.....	46

SEZNAM PŘÍLOH

- P 1 : CD-ROM obsahující knihovnu samočinně se nastavujících regulátorů pro systémy se dvěma vstupy a dvěma výstupy kompatibilní s Real – Time Workshopem. Soubory s naměřenými daty při řízení laboratorního modelu.
- P 2 : Algoritmus Gaussovy eliminační metody.
- P 3 : Ukázka kódu ze souboru iden.c.

PŘÍLOHA P 2: ALGORITMUS GAUSSOVY ELIMINAČNÍ METODY.

```
for (i=0; i<n; i++)
{
    max = fabs(A[i*n+i]);
    k = i;
    for (j=i+1; j<n; j++)
    {
        if (max < fabs(A[j*n+i]))
        {
            max = fabs(A[j*n+i]);
            k = j;
        }
    }
    //podmínka singularity matice soustavy
    if (max != 0)
    {
        for (j=i; j<n; j++)
        {
            pom = A[i*n+j];
            A[i*n+j] = A[k*n+j];
            A[k*n+j] = pom;
        }
        pom = B[i];
        B[i] = B[k];
        B[k] = pom;

        //vydělení hlavního řádku hodnotou hlavního prvku
        hlavni_prvek = A[i*n+i];
        for (j=i; j<n; j++)
        {
            A[i*n+j] = A[i*n+j]/hlavni_prvek;
        }
        B[i] = B[i]/hlavni_prvek;

        //eliminace neznámé x-i z ostatních řádků
        for (j=i+1; j<n; j++)
        {
            for (m=i+1; m<n; m++)
            {
                A[j*n+m] = A[j*n+m] - A[j*n+i] * A[i*n+m];
            }
            B[j] = B[j] - A[j*n+i] * B[i];
        }
    }
}

//zpětné dosazování
for (i=n-1; i>=0; i--)
{
    X[i] = B[i];
    for (j=i+1; j<n; j++)
    {
        X[i] = X[i] - A[i*n+j] * X[j];
    }
}
```

PŘÍLOHA P 3: UKÁZKA KÓDU ZE SOUBORU IDEN.C.

```
1 #define S_FUNCTION_NAME iden
2 #define S_FUNCTION_LEVEL 2
3 #include "simstruc.h" /* the simstruct access macros */
4
5 /*=====
6  * Number of S-function Parameters and macros to access from the
6  SimStruct *
6  =====*/
7 #define Tv_PARAM (ssGetSFcnParam(S,0)) //Tv
8 #define Theta0a_PARAM (ssGetSFcnParam(S,1))
8 // [a1,a2,a3,a4,a5,a6,a7,a8]
9 #define Theta0b_PARAM (ssGetSFcnParam(S,2))
9 // [b1,b2,b3,b4,b5,b6,b7,b8]
10 #define C0_PARAM (ssGetSFcnParam(S,3)) //C0
11 #define fi0_PARAM (ssGetSFcnParam(S,4)) //fi0
12 #define ny0_PARAM (ssGetSFcnParam(S,5)) //ny0
13 #define la0_PARAM (ssGetSFcnParam(S,6)) //la0
14 #define rho0_PARAM (ssGetSFcnParam(S,7)) //rho0
15 #define M_PARAM (ssGetSFcnParam(S,8))
16 /*=====
17  * Macros to access the S-function parameter values *
18  *=====*/
19 #define Tv ((real_T) mxGetPr(Tv_PARAM)[0])
20 #define Theta0a ((real_T*) mxGetPr(Theta0a_PARAM))
20 //vector
21 #define Theta0b ((real_T*) mxGetPr(Theta0b_PARAM))
21 //vector
22 #define C0 ((real_T*) mxGetPr(C0_PARAM))
22 //vector
23 #define fi0 ((real_T) mxGetPr(fi0_PARAM)[0])
24 #define ny0 ((real_T) mxGetPr(ny0_PARAM)[0])
25 #define la0 ((real_T) mxGetPr(la0_PARAM)[0])
26 #define rho0 ((real_T*) mxGetPr(rho0_PARAM))
26 //vector
27 #define M ((real_T*) mxGetPr(M_PARAM))
27 //vector
28 #include "tmwtypes.h"
29 #include "math.h"
30 /*=====
31  * S-function methods *
32  *=====*/
33 static void mdlInitializeSizes(SimStruct *S)
```

```

34 {
35 //params: Tv,Theta0a,Theta0b,C0,fi0,ny0,la0,rho0,M
36 ssSetNumSFcnParams(S, 9);
37 - kód vynechán
38 ssSetNumInputPorts( S, 1);
39 ssSetInputPortWidth(S, 0, 4);          // uk1 uk2 yk1 yk2
40 ssSetInputPortRequiredContiguous(S, 0, true);
41 ssSetNumOutputPorts(S, 1);
42 ssSetOutputPortWidth(S, 0, 24);        //Theta_ab1 Theta_ab2 M
43 ssSetNumDiscStates(S, 156);            //Theta_ab1 Theta_ab2 PHI C1 C2
    //la1 la2 nu1 nu2 phi1 phi2
44
45 ssSetInputPortDirectFeedThrough(S, 0, 1);
46 ssSetNumSampleTimes(S, 1);
47 ssSetNumRWork(S, 0);
48 ssSetNumIWork(S, 0);
49 ssSetNumPWork(S, 0);
50 ssSetNumModes(S, 0);
51 ssSetNumNonsampledZCs(S, 0);
52 ssSetOptions(S, 0);
53 }
54
55 static void mdlInitializeSampleTimes(SimStruct *S)
56 {
57     ssSetSampleTime(S, 0, Tv);
58     ssSetOffsetTime(S, 0, 0.0);
59 }
60
61 #define MDL_INITIALIZE_CONDITIONS
62 static void mdlInitializeConditions(SimStruct *S)
63 {
64 - kód vynechán
65 }
66 static void mdlOutputs(SimStruct *S, int_T tid)
67 {
68 //states
69 mxArray *Theta_ab1;          //[-a1, -a2, -a3, -a4, b1, b2, b3, b4]
70 mxArray *Theta_ab2;          //[-a5, -a6, -a7, -a8, b5, b6, b7, b8]
71 mxArray *PHI;                //y1(k-1), -y1(k-2), -y2(k-1), -y2(k-2), u1(k-1),
    //u1(k-2), u2(k-1), u2(k-2)
72 mxArray *C1, *C2;            //x(23:86), x(87:150)
73 mxArray *PHI_C1, *PHI_C2, *C1_PHI, *C2_PHI;

```

```

74
75 double *pTheta_ab1, *pTheta_ab2, *pPHI, *pC1, *pC2, *pM, *pPHI_C1;
76 double *pPHI_C2, *pC1_PHI, *pC2_PHI;
77
78 //inputs
79 real_T uk1, uk2, yk1, yk2;
80 //others
81 const real_T *in = (const real_T*) ssGetInputPortRealSignal(S,0);
82 real_T *out = ssGetOutputPortRealSignal(S,0);
83 const real_T *xdisc = ssGetRealDiscStates(S);
84 double la1, la2, nu1, nu2, phi1, phi2, e1, e2, xi1, xi2;
85 double Theta_ab1_PHI, Theta_ab2_PHI;
86 int i, j;
87
88 uk1 = in[0]; //u1(k-1)
89 uk2 = in[1]; //u2(k-1)
90 yk1 = in[2]; //y1(k)
91 yk2 = in[3]; //y2(k)
92
93 Theta_ab1 = mxCreateDoubleMatrix(8, 1, mxREAL);
94
95 -kód vynechán
96
97 pTheta_ab1 = mxGetPr(Theta_ab1);
98 pTheta_ab2 = mxGetPr(Theta_ab2);
99 pPHI = mxGetPr(PHI);
100 pC1 = mxGetPr(C1);
101 pC2 = mxGetPr(C2);
102 pPHI_C1 = mxGetPr(PHI_C1);
103 pPHI_C2 = mxGetPr(PHI_C2);
104 pC1_PHI = mxGetPr(C1_PHI);
105 pC2_PHI = mxGetPr(C2_PHI);
106 Theta_ab1_PHI = 0.0;
107 for (i=0; i<8; i++) //Theta_ab1_PHI = Theta_ab1' * PHI
108 {
109     Theta_ab1_PHI += pTheta_ab1[i] * pPHI[i];
110 }
111 e1 = yk1 - Theta_ab1_PHI;
112
113 Theta_ab2_PHI = 0.0;
114 for (i=0; i<8; i++) //Theta_ab2_PHI = Theta_ab2' * PHI

```

```

115 {
116     Theta_ab2_PHI += pTheta_ab2[i] * pPHI[i];
117 }
118 e2 = yk2 - Theta_ab2_PHI;
119
120 xi1 = 0.0;
121 for (i=0; i<8; i++)      //xi1 = PHI' * C1 * PHI
122 {
123     pPHI_C1[i] = 0.0;
124     for (j=0; j<8; j++)
125     {
126         pPHI_C1[i] += pPHI[j] * pC1[j+i*8]; //C:j-th row, i-th column
127     }
128 xi1 += pPHI_C1[i] * pPHI[i];
129 }
130
131 xi2 = 0.0;
132 for (i=0; i<8; i++)      //xi2 = PHI' * C2 * PHI
133 {
134     pPHI_C2[i] = 0.0;
135     for (j=0; j<8; j++)
136     {
137         pPHI_C2[i] += pPHI[j] * pC2[j+i*8];
138     }
139 xi2 += pPHI_C2[i] * pPHI[i];
140 }
141 for (i=0; i<8; i++) //Theta_ab1 = Theta_ab1 +(C1*PHI/(1+xi1) * e1);
142 {
143     pC1_PHI[i] = 0.0;
144     for (j=0; j<8; j++)
145     {
146         pC1_PHI[i] = pC1_PHI[i] + pC1[i+j*8] * pPHI[j];
147     }
148 pTheta_ab1[i] = pTheta_ab1[i] + (pC1_PHI[i]/(1+xi1) * e1);
149 }
150 for (i=0; i<8; i++)//Theta_ab2 = Theta_ab2 +(C2*PHI/(1+xi2) * e2);
151 {
152     pC2_PHI[i] = 0.0;
153     for (j=0; j<8; j++)
154     {
155         pC2_PHI[i] = pC2_PHI[i] + pC2[i+j*8] * pPHI[j];

```

```

156     }
157 pTheta_ab2[i] = pTheta_ab2[i] + (pC2_PHI[i]/(1+xi2) * e2);
158 }
159
160 //compute output
161
162 - kód vynechán
163
164 #define MDL_UPDATE
165 static void mdlUpdate(SimStruct *S, int_T tid)
166 {
167
168 - kód vynechán
169
170 xi1 = 0.0;
171 for (i=0; i<8; i++)      //xi1 = PHI' * C1 * PHI
172 {
173     for (j=0; j<8; j++)
174     {
175         xi1 += pPHI[j] * pC1[j+i*8]* pPHI[i]; //C:j-th row,i-th column
176     }
177 }
178
179 xi2 = 0.0;
180 for (i=0; i<8; i++)      //xi2 = PHI' * C2 * PHI;
181 {
182     for (j=0; j<8; j++)
183     {
184         xi2 += pPHI[j] * pC2[j+i*8] * pPHI[i];
185     }
186 }
187 if (xi1 > 0)
188 {
189     epsilon1 = phi1-(1-phi1)/xi1;
190     //C1= C1 - C1*PHI*PHI'*C1/(inv(epsilon1)+xi1)
191     for (i=0; i<8; i++)
192     {
193         pC1_PHI[i] = 0.0;
194         pPHI_C1[i] = 0.0;
195         for (j=0; j<8; j++)
196         {

```

```

197             pC1_PHI[i] += pC1[i+j*8] * pPHI[j];
198             pPHI_C1[i] += pC1[j+i*8] * pPHI[j];
199         }
200     }
201     for (i=0; i<8; i++)
202     {
203         for (j=0; j<8; j++)
204         {
205             pC1[j+8*i] -= pC1_PHI[j] * pPHI_C1[i]/((1/epsilon1)+xi1);
206         }
207     }
208 }
209
210 - kód vynechán
211
212 Theta_ab1_PHI = 0.0;
213 for (i=0; i<8; i++) //Theta_ab1_PHI = Theta_ab1' * PHI
214 {
215     Theta_ab1_PHI += pTheta_ab1[i] * pPHI[i];
216 }
217 e1 = yk1 - Theta_ab1_PHI;
218
219 Theta_ab2_PHI = 0.0;
220 for (i=0; i<8; i++) //Theta_ab2_PHI = Theta_ab2' * PHI
221 {
222     Theta_ab2_PHI += pTheta_ab2[i] * pPHI[i];
223 }
224 e2 = yk2 - Theta_ab2_PHI;
225
226 if (xi1 > 0)
227 {
228     la1 = phi1*(la1 + e1*e1/(1+xi1));
229     nu1 = phi1*(nu1+1);
230     eta1 = e1*e1/la1;
231     phi1 = 1/(1+(1+rho0[0])*(log(1+xi1)+((nu1+1)*eta1;
232     phi1 = phi1/(1+xi1+eta1)-1)*xi1/(1+xi1)));;
233 }
234 if (xi2 > 0)
235 {
236     la2 = phi2*(la2 + e2*e2/(1+xi2));
237     nu2 = phi2*(nu2+1);

```

```

238         eta2 = e2*e2/la2;
239         phi2 = 1/(1+(1+rho0[1])*(log(1+xi2)+((nu2+1)*eta2;
240         phi2 = phi2/(1+xi2+eta2)-1)*xi2/(1+xi2));
241     }
242     //ulozeni do posuvneho vektoru
243     for (i=0; i<8; i++)//Theta_ab1 = Theta_ab1 +(C1*PHI/(1+xi1) * e1);
244     {
245         pTheta_ab1[i] = pTheta_ab1[i] + (pC1_PHI[i]/(1+xi1) * e1);
246     }
247     for (i=0; i<8; i++)//Theta_ab2 = Theta_ab2 +(C2*PHI/(1+xi2) * e2);
248     {
249         pTheta_ab2[i] = pTheta_ab2[i] + (pC2_PHI[i]/(1+xi2) * e2);
250     }
251     - kód vynechán
252 }
253 static void mdlTerminate(SimStruct *S)
254 {
255 }
256 /*=====
257  * Required S-function trailer *
258  *=====*/
259 #ifdef MATLAB_MEX_FILE
260 #include "simulink.c"
261 #else
262 #include "cg_sfund.h"
263 #endif

```