TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

IMPLEMENTACE STAVOVÉHO REGULÁTORU

Diplomová práce

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 3902T005 – Automatické řízení a inženýrská informatika

Autor práce: **Bc. Jan Beran**
Vedoucí práce: Ing. Pavel Herajn

Liberec 2015

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

IMPLEMENTATION OF STATE CONTROL

Diploma thesis

Study programme: N2612 – Electrical Engineering and Informatics
Study branch: 3902T005 – Automatic Control and Applied Computer Science

Author: **Bc. Jan Beran**
Supervisor: Ing. Pavel Herajn

Liberec 2015

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2014/2015**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Jan Beran**
Osobní číslo:	**M12000243**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Automatické řízení a inženýrská informatika**
Název tématu:	**Implementace stavového regulátoru**
Zadávající katedra:	**Ústav mechatroniky a technické informatiky**

Zásady pro vypracování:

1. Upravte stávající software mikrořadiče tak, aby plnil funkci stavového regulátoru.
2. Navrhněte a implementujte komunikační protokol mezi mikrořadičem a PC.
3. V prostředí Matlab vytvořte grafické prostředí umožňující kompletní nastavení parametrů stavového regulátoru a zobrazování časových průběhů stavových veličin.
4. Proveďte identifikaci dané fyzikální úlohy.
5. Navrhněte stavový regulátor, který bude danou soustavu optimálně regulovat dle zvoleného kritéria.

Rozsah grafických prací: **dle potřeby dokumentace**

Rozsah pracovní zprávy: **40–50 stran**

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

[1] **HEROUT, Pavel. Učebnice jazyka C. 6. vyd. KOPP, 2009. ISBN 978-80-7232-383-8.**

[2] **Stránky Atmel. [online]. [cit. 2013-10-10]. Dostupné z: http://www.atmel.com/**

[3] **Stránky Matlab. [online]. [cit. 2013-10-10]. Dostupné z: http://www.mathworks.com/**

[4] **BALÁTĚ, Jaroslav. Automatické řízení. BEN-Technická literatura, 2004. ISBN 978-80-7300-1.**

Vedoucí diplomové práce: **Ing. Pavel Herajn**
Ústav mechatroniky a technické informatiky

Konzultant diplomové práce: **Ing. Petr Mrázek, Ph.D.**
Ústav mechatroniky a technické informatiky

Datum zadání diplomové práce: **10. října 2014**

Termín odevzdání diplomové práce: **15. května 2015**

prof. Ing. Václav Kopecký, CSc.
děkan

L.S.

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci dne 10. října 2014

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 15. 5. 2015

Podpis: Jan Beran

Poděkování

Rád bych poděkoval vedoucímu mé diplomové práce Ing. Pavlu Herajnovi především za jeho pevné nervy a trpělivost se mnou. Také bych mu rád poděkoval za podnětné a hodnotné rady, čas strávený při konzultacích a odborné vedení během mé práce.

Abstrakt

Cílem této diplomové práce je úprava programu pro řízení fyzikálního modelu a vytvoření grafické aplikace v programu Matlab pro jeho řízení. Úprava spočívá v nahrazení stávajícího regulátoru regulátorem stavovým. Předmětem zprávy je úvod do základní znalosti stavového řízení a pokročilejšího programování v jazyce C a v programu Matlab. Tato práce se konkrétně zabývá úpravou programu pro řízení fyzikálního modelu a návrhu grafické aplikace pro úpravu a sběr dat z fyzikálního modelu.

Úvod práce se zabývá uvedením do problematiky stavového popisu, zjednodušeným návodem pro tvorbu grafického uživatelského rozhraní v programu Matlab a popisem fyzikálního modelu.

Dále je popsán upravený kód mikroprocesoru se stavovým regulátorem a postup, jak ho nahrát pomocí programátoru do mikroprocesoru. V této části zprávy je také zmíněn kód grafické aplikace pro řízení modelu.

Další část zprávy se zabývá identifikací modelu. Na základě provedené identifikace jsou zde zpracovány návrhy na možnosti řízení pomocí stavového regulátoru.

Závěr zprávy se věnuje ověření a vyhodnocení funkce navrženého regulátoru, a to formou měření.

Klíčová slova

Matlab, Simulink, identifikace, stavový regulátor, stavový popis, GUI, PRESTO, fyzikální model, GUIDE, stavová regulace

Abstract

The aim of this Diploma thesis is modification of program for controlling physical model and creating graphical user interface in Matlab for it's controlling. The modification involves replacing the current regulator by state controller. The subject of the report is an introduction to basic knowledge of state feedback controlling and advanced programming in C code and Matlab. This work deals specifically with modifying the program for the controlling of the physical model and design graphical user interface for editing and collecting data from the physical model.

Introduction of Diploma thesis deals with the introduction to the issue of state space modelling, simplified instructions for creating graphical user interface in Matlab and description of the physical model.

It also describes the modified code of microprocessor with state controller, and how to upload it using the programmer to the microprocessor. In this part of the report is also mentioned code of graphical user interface for controlling physical model.

Further part of the report deals with the identification of the model. On the basis of identification are prepared designs to possibilities for controlling using state controller.

Conclusion of the report is devoted to the verification and assessment functions of the controller designed in the form of measurement.

Keywords

Matlab, Simulink, identification, state controller, state space modelling, GUI, PRESTO, physical model, GUIDE, state space controlling

Obsah

Seznam ilustrací

Seznam grafů

Seznam tabulek

Seznam zdrojových kódů

Seznam použitých termínů a zkratek

$w(t)$	Žádaná hodnota
$u(t)$	Akční zásah
$y(t)$	Regulovaná veličina ve spojitém čase
$e(t)$	Regulační odchylka
A	Matice soustavy
B	Matice vstupu
C	Matice výstupu
D	Váhová matice
M	Matice soustavy v nespojitém čase
N	Matice vstupu v nespojitém čase
R	Matice regulátoru
L	Matice estimátoru
$F(s)$	Přenosová funkce
$\acute{x}(t)$	Derivace vektoru stavových proměnných
$x(t)$	Vektor stavových proměnných
$x(k+1)$	Derivace vektoru stavových proměnných v nespojitém čase
$x(k)$	Vektor stavových proměnných v nespojitém čase
$\hat{x}(k+1)$	Estimace derivace vektoru stavových proměnných v nespojitém čase
$\hat{x}(k)$	Estimace vektoru stavových proměnných v nespojitém čase
$y(k)$	Regulovaná veličina v nespojitém čase
$\hat{y}(k)$	Estimace regulované veličiny v nespojitém čase

Úvod

Úkolem této diplomové práce je implementovat do mikroprocesoru fyzikálního modelu stavový regulátor, který bude model řídit. Stavová regulace je efektivní a začíná být více používaná. To nejen proto, že umožňuje řídit systémy se složitou vnitřní strukturou nebo systémy s více vstupy / výstupy, ale především proto, že návrh regulátoru a estimátoru je relativně jednoduchý, pokud je znám ápřenosová funkce soustavy.

V úvodu práce se budu věnovat problematice stavového řízení, jako je například vytvoření stavového popisu, návrhu estimátoru a stavového regulátoru. Dále zde představím vývojové prostředí GUIDE, jenž je součástí programu Matlab, ve kterém se bude programovat grafické rozhraní pro ovládání modelu.

Další kapitoly práce budou věnovány popisu fyzikálního modelu. Zejména jeho hardwarovému řešení a technickým parametrům. Zároveň v této části práce bude popsáno odvození matematického popisu fyzikálního modelu.

V následující kapitole se budu zabývat identifikaci přenosové funkce modelu na základě naměřených dat a zjištěného matematického popisu. Po provedení identifikace se budu věnovat návrhu estimátoru a stavového regulátoru.

Dále se budu zabývat programováním mikriprocesoru v programu Atmel Studio. Zde bude zapotřebí nejen vyřešit příjem a odesílání dat, ale také především řídicí algoritmus stavové regulace.

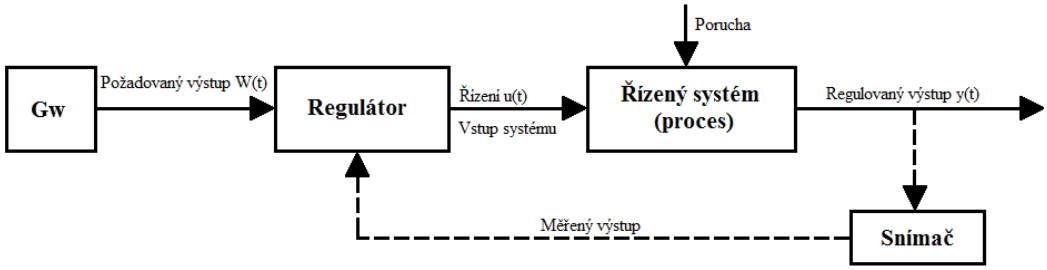
Další kapitola se bude věnovat návrhu a tvorbě grafického rozhraní pro ovládání modelu. Předpokladem je, aby aplikace umožňovala měnit parametry stavového řízení v mikroprocesoru, sledovat aktuální hodnoty a ty také ukládat pro možnost budoucího zpracování.

V závěru práce se budu zabývat vyhodnocením kvality regulace na základě provedených měření.

1 Stavový popis dynamických systémů

Primární funkcí prakticky každého řídicího systému (regulátoru) je regulovat chování jedné či více proměnných (regulovaných výstupů) na reálném řízeném systému či procesu tak, aby bylo dosaženo nějakých předem daných požadavků a to při současném respektování specifikovaných omezení a realizovatelnosti řídicího systému.

V případě, že je k dispozici přesný matematický model řízeného systému a jeho okolí a vylučujeme výskyt jakékoliv neurčitosti, lze požadovaného chování docílit výpočtem přímovazebního řízení, které je charakteristické tím, že regulátor nevyužívá zpětnou informaci o skutečném průběhu regulovaných veličin a vstupem regulátoru je pouze referenční signál. V reálných situacích je však nějaká forma neurčitosti vždy přítomna (matematický model řízeného systému není přesným modelem chování reálného systému, výskyt náhodných poruch, změny parametrů atd.), a proto dáváme přednost návrhu zpětnovazebního řízení, kdy regulátor kromě referenčního signálu využívá také informaci o skutečném průběhu regulovaných veličin a může tak reagovat na nežádoucí změny způsobené neurčitostí.



Obrázek 1.1: Porovnání přímovazebního a zpětnovazebního řízení

Návrh přímovazebního nebo zpětnovazebního řízení tedy předpokládá důkladnou znalost funkce a chování reálného řízeného systému, které musí být podchyceny vytvořením adekvátního matematického modelu reálného řízeného systému.

Použití matematicko-fyzikálního modelování vede obvykle na matematický model vnějšího popisu, který popisuje účinek vybrané vstupní veličiny na reálném systému $u(t)$na dynamické chování vybrané výstupní veličiny $y(t)$.

Tento matematický model lze popsat lineární diferenciální rovnicí n-té ho řádu s konstantními parametry.

$$y^{(n)}(t) + a_{n-1}y^{(n-1)}(t) + .. + a_1\dot{y}(t) + a_0y(t) = b_mu^{(m)}(t) + b_{m-1}u^{(m-1)}(t) + .. + b_0u(t); m \leq n \quad (1.1)$$

Zavedením stavových proměnných $x_1 \equiv y$, $x_2 \equiv \dot{y}$, … $x_n \equiv y^{(n)}$ můžeme tuto rovnici vyjádřit ve formě soustavy n lineárních diferenciálních rovnic 1. řádu, tzv. stavovými rovnicemi:

$$\acute{x_1} = x_2 \tag{1.2}$$

$$\acute{x_2} = x_3 \tag{1.3}$$

$$\vdots$$

$$\acute{x_{n-1}} = x_n \tag{1.4}$$

$$\acute{x_n} = -\frac{a_0}{a_n}x_1 - \frac{a_1}{a_n}x_2 - .. - \frac{a_{n-1}}{a_n}x_n + \frac{1}{a_n}u \tag{1.5}$$

Výstupem soustavy je (výstupem můžeme zvolit libovolnou stavovou proměnnou)

$$y = x_1 \tag{1.6}$$

Tuto soustavu stavových rovnic spolu s rovnicí výstupu můžeme vyjádřit v maticovém tvaru:

- spojitý popis:

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{1.7}$$

$$y(t) = Cx(t) + Du(t) \tag{1.8}$$

$$\begin{bmatrix} \acute{x_1}(t) \\ \vdots \\ \acute{x_n}(t) \end{bmatrix} = \begin{bmatrix} \quad A \quad \end{bmatrix} \begin{bmatrix} x_1(t) \\ \vdots \\ x_n(t) \end{bmatrix} + \begin{bmatrix} B \end{bmatrix} u(t) \tag{1.9}$$

$$y(t) = [\quad C \quad] \begin{bmatrix} x_1(t) \\ \vdots \\ x_n(t) \end{bmatrix} + Du(t) \tag{1.10}$$

- Diskrétní popis:

$$x(k+1) = Mx(k) + Nu(k) \tag{1.11}$$

$$y(k) = Cx(k) + Du(k) \tag{1.12}$$

$$\begin{bmatrix} x_1(k+1) \\ \vdots \\ x_n(k+1) \end{bmatrix} = \begin{bmatrix} \quad M \quad \end{bmatrix} \begin{bmatrix} x_1(k) \\ \vdots \\ x_n(k) \end{bmatrix} + \begin{bmatrix} N \end{bmatrix} u(k) \tag{1.13}$$

$$y(k) = [\quad C \quad] \begin{bmatrix} x_1(k) \\ \vdots \\ x_n(k) \end{bmatrix} + Du(k) \tag{1.14}$$

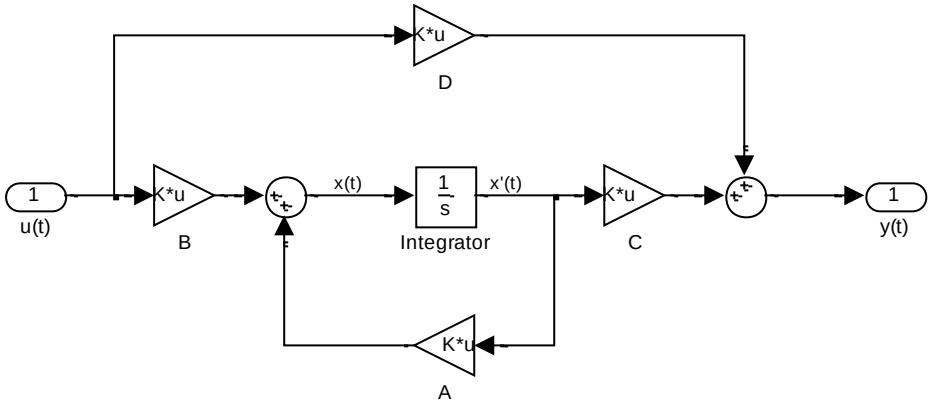
kde A, M je matice soustavy rozměru (m x n)

B, M je matice vstupu rozměru (n x m)

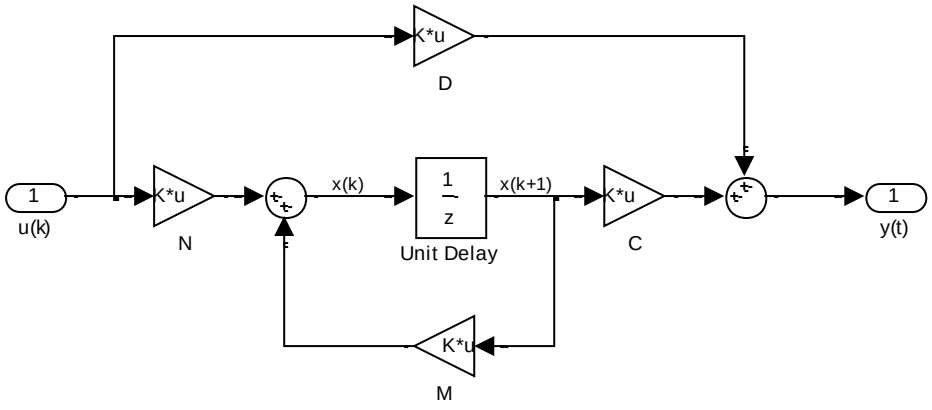
C je matice výstupu rozměru (r x n)

D je váhová matice rozměru (r x m)

Pro potřeby simulace dynamického systému zapsaného v maticovém tvaru můžeme použít obecné stavové schéma systému (viz Obrázek 1.2 a Obrázek 1.3).



Obrázek 1.2: Obecné stavové schéma spojitého systému



Obrázek 1.3: Obecné stavové schéma diskrétního systému

1.1 Normální forma řiditelnosti

Spojitý lineární dynamický systém je řiditelný, jestliže existuje takové řízení („cesta“) $u(t)$ na konečném časovém intervalu $t \in \langle t_0, t_1 \rangle$, které způsobí změnu počátečního stavu systému $x(t_0)$ v koncový stav $x(t_1)$.

$$A = \begin{bmatrix} 0 & 1 & 0 & \vdots & 0 \\ 0 & 0 & 1 & \vdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \vdots & 1 \\ -a_0 & -a_1 & -a_2 & \vdots & -a_{n-1} \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

$$C = [b_0 - b_n a_0, \quad b_1 - b_n a_1, \quad \cdots, \quad b_{n-1} - b_n a_{n-1}] \quad D = [b_n]$$

Matice řiditelnosti:

$$G = [B, \quad AB, \quad A^2B, \quad \cdots, \quad A^{n-1}B] \tag{1.15}$$

1.1 Normální forma pozorovatelnosti

Spojitý lineární dynamický systém je pozorovatelný, pokud pozorováním vstupu $u(t)$ a výstupu $y(t)$ na konečném časovém intervalu $t \in \langle t_0, t_1 \rangle$ lze určit počáteční stav systému $x(t_0)$.

$$A = \begin{bmatrix} -a_{n-1} & 1 & 0 & \vdots & 0 \\ -a_{n-2} & 0 & 1 & \vdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ -a_1 & 0 & 0 & \vdots & 1 \\ -a_0 & 0 & 0 & \vdots & 0 \end{bmatrix} \quad B = \begin{bmatrix} b_{n-1} - b_n a_{n-1} \\ b_{n-2} - b_n a_{n-2} \\ \vdots \\ b_1 - b_n a_1 \\ b_0 - b_n a_0 \end{bmatrix} \quad C = [1 \quad 0 \quad \cdots \quad 0]$$

$$D = [b_n]$$

Matice pozorovatelnosti:

$$R = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix} \tag{1.16}$$

1.2 Převod stavového popisu na obrazový přenos

Předpokládejme stavovou reprezentaci dle (1.7) a (1.8), matici D = 0 a soustavu s jedním vstupem a jedním výstupem. Počáteční podmínky jsou nulové, $x(0) = 0$. Použitím Laplaceovy transformace dostaneme rovnost

$$sX(s) = A \cdot X(s) + B \cdot U(s) \tag{1.17}$$

Řešením algebraické rovnice (1.17) určíme $\mathscr{L}$– obraz vektoru stavu

$$[sI - A] \cdot X(s) = B \cdot U(s) \rightarrow X(s) = [sI - A]^{-1} \cdot B \cdot U(s) \tag{1.18}$$

kde I je jednotková matice. Po dosazení rovnice (1.18) do (1.8) získáme $\mathscr{L}$– obraz výstupu, který je roven

$$Y(s) = C \cdot X(s) = C \cdot [sI - A]^{-1} \cdot B \cdot U(s) = F(s) \cdot U(s) \tag{1.19}$$

Použijeme-li vztah pro výpočet inverzní matice

$$[sI - A]^{-1} = \frac{adj[sI - A]}{det[sI - A]} \tag{1.20}$$

a dosadíme do rovnice (1.19), získáme $\mathscr{L}$– obraz výstupu, který je roven

$$Y(s) = C \cdot \frac{adj[sI - A]}{det[sI - A]} \cdot B \cdot U(s) = F(s) \cdot U(s). \tag{1.21}$$

Vyjádřením $F(s)$ z rovnice (1.21) získáme výsledný obrazový přenos

$$\begin{aligned} F(s) = \frac{Y(s)}{U(s)} &= C \cdot [sI - A]^{-1} \cdot B \\ &= \frac{1}{det[sI - A]} \cdot C \cdot adj[sI - A] \cdot B. \end{aligned} \tag{1.22}$$

1.3 Návrh diskrétního estimátoru úplného řádu (deterministický Luenbergerův estimátor)

Estimátor slouží k odhadnutí neměřitelných složek systému. Výsledkem je dynamický odhad neměřitelných, někdy i měřitelných, složek stavového prostoru, které jsou využívány k výpočtu akčního zásahu.

Jestliže měřená výstupní veličina $\mathrm{y}(k)$ neobsahuje žádný aditivní šumový signál, mluvíme o deterministické estimaci.

Deterministické i stochastické přístupy vedou ke stejné struktuře estimátorů. Jejich parametry jsou však optimalizovány buď vzhledem k dynamice estimačního procesun(deterministické estimátory) nebo vzhledem k pravděpodobnostním momentům předpokládaných náhodných poruchových signálů (stochastické estimátory).

Deterministický estimátor vychází ze znalosti stavových matic A, B, C, D, případně matic M, N, C, D a ze znalosti vstupu$u(t)$ $(u(k))$, a výstupu $\mathrm{y}(t)$ $(\mathrm{y}(k))$ dynamického systému.

Při návrhu estimátoru uvažujeme popis dynamického systému podle rovnic (1.11) a (1.12))

$$x(k+1) = Mx(k) + Nu(k)$$

$$y(k) = Cx(k) + Du(k)$$

Estimátor:

$$\hat{x}(k+1) = M_E\hat{x}(k) + N_E u(k) + Ly(k) \tag{1.23}$$

$$\hat{y}(k) = C\hat{x}(k) + Du(k) \tag{1.24}$$

Podmínka autonomnosti estimace:

$$\left.\begin{matrix} M - M_E - LC \\ N - N_E - LD \end{matrix}\right\} \Delta x(k+1) = M_E \Delta x(k) \tag{1.25}$$

Podmínka stability estimace:

$$M_E \text{ je stabilní pokud } det(\lambda E - M_E) = det(\lambda E - M + LC) = \tag{1.26}$$

$$(\lambda - \lambda_1)(\lambda - \lambda_2) \ldots$$

$$\lambda_i : |\lambda_i| < 1, i = 1,2,3 \ldots$$

Návrh estimátoru spočívá ve dvou krocích. V prvním kroku se zvolí matice L tak, aby λ_i matice M_E byly uvnitř jednotkového kruhu. Ve speciálním případě můžeme volit $\lambda_i = 0$, což povede na konečný a minimální počet kroků estimace. V druhém kroku se spočítají matice estimátoru M_E a N_E podle rovnice (1.25).

Dosazením výše vypočítaných matic M_E a N_E do rovnice (1.23), získáme rovnici pro výpočet stavových veličin

$$\hat{x}(k+1) = M\hat{x}(k) + Nu(k) + L\big(y(k) - \hat{y}(k)\big) \tag{1.27}$$

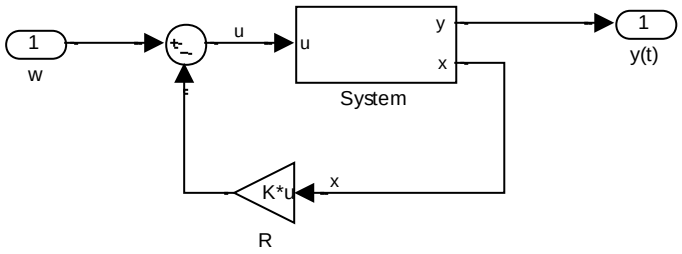
1.4 Návrh stavového regulátoru

Dynamické vlastnosti systému jsou dány vlastními čísly matice A (případně maticí M), která tvoří póly soustavy v komplexní rovině. Podle jejich polohy vzhledem k osám lze posuzovat stabilitu, vlastní frekvenci a tlumení.

Princip funkce stavového regulátoru spočívá v tom, že s jeho pomocí můžeme změnit polohu těchto pólů tak, abychom dosáhli požadovaného dynamického chování. Prvky matice stavového regulátoru jsou konstanty.

S ohledem na komplikovanou strukturu se stavové regulátory nejčastěji realizují diskrétní časové oblasti.

Obecné schéma se stavovým regulátorem:



Obrázek 1.4: Obecné schéma se stavovým regulátorem

Při návrhu stavového regulátoru se pracuje s diskrétním popisem soustavy (rovnice (1.11) a (1.12))

$$x(k+1) = Mx(k) + Nu(k)$$

$$y(k) = Cx(k) + Du(k)$$

Výstup z regulátoru $z(k)$má následující tvar:

$$z(k) = Rx(k) \tag{1.28}$$

Akční zásah $u(k)$ je roven

$$u(k) = w(k) - z(k) \tag{1.29}$$

Po dosazení a úpravě výše uvedených rovnic dostaneme diskrétní popis soustavy obsahující stavový regulátor

$$x(k+1) = (M - NR)x(k) + Nw(k) \tag{1.30}$$

$$y(k) = (C - DR)x(k) + Dw(k) \tag{1.31}$$

1.4.1 Návrh stavového regulátoru v konečném počtu kroků

Pokud je regulátor navržen touto metodou, regulační pochody končí v konečném čase. Počet kroků regulace je obvykle úměrný řádu soustavy. Výhodou tohoto návrhu je rychlost a jednoduchost regulace. Naopak nevýhodou jsou vysoké akční zásahy.

Předpoklady:

$$x(0) \neq 0$$

$$w(k) = 0 \ \rightarrow u(k) = -Rx(k)$$

Návrh:

$$x(k+1) = Mx(k) + Nu(k) = Mx(k) - NRx(k) = (M - NR)x(k)$$

$$\begin{aligned} x(k+2) &= Mx(k+1) + Nu(k+1) = \ Mx(k+1) - NRx(k+1) \\ &= M(M - NR)x(k) - NR(M - NR)x(k) \\ &= (M^2 - MNR - NRM + (NR)^2)x(k) \end{aligned}$$

$$x(k+3) = Mx(k+2) + Nu(k+2) = M^3x(k) + M^2Nu(k) + MNu(k+1) + Nu(k+2)$$

$$\vdots$$

$$\begin{aligned} x(k+n) = M^n x(k) + M^{n-1}Nu(k) + M^{n-2}Nu(k+1) + \cdots + MNu(k+n-2) \\ + Nu(k+n-1) \end{aligned}$$

$$x(k+n) = M^n x(k) + [M^{n-1}N, M^{n-1}N, \cdots, MN, N] \begin{bmatrix} u(k+n-1) \\ u(k+n-2) \\ \vdots \\ u(k) \end{bmatrix} \tag{1.32}$$

Matice $[M^{n-1}N, M^{n-1}N, \cdots, MN, N]$ se nazývá maticí řiditelnosti a označuje se písmenem G. Hledáme-li regulátor s řízením v konečném počtu kroků pro SISO systém, musí se rovnice (1.27) rovna nule.

$$M^n x(k) + G \begin{bmatrix} u(k+n-1) \\ u(k+n-2) \\ \vdots \\ u(k) \end{bmatrix} = 0 \tag{1.33}$$

Úpravou předchozí rovnice získáme

$$\begin{bmatrix} u(k+n-1) \\ u(k+n-2) \\ \vdots \\ u(k) \end{bmatrix} = G^{-1}M^n x(k) \tag{1.34}$$

Vyjádříme z rovnice (1.29) $u(k)$

$$u(k) = -[0 \quad 0 \quad \cdots \quad 1]G^{-1}M^n x(k) \tag{1.35}$$

Z rovnice (1.30) nyní můžeme vyjádřit matici regulátoru R, která se rovná

$$R = -[0 \quad 0 \quad \cdots \quad 1]G^{-1}M^n \tag{1.36}$$

Pro SISO systémy tuto matici R nazýváme Ackermanova formule

$$R = -[0 \quad 0 \quad \cdots \quad 1]G^{-1}\Delta(M) \tag{1.37}$$

kde $\Delta(M)$ je maticový polynom roven

$$\Delta(M) = (M - \lambda_1 E)(M - \lambda_2 E)\cdots(M - \lambda_n E) \tag{1.38}$$

Výpočtem této matice R zajistíme konečný a minimální počet kroků regulace soustavy.

1.4.2 Návrh stavového regulátoru podle kvadratického kritéria

Tato návrhová metoda využívá Riccatiho rovnici. Cílem je minimalizovat kvadratické kritérium a tím najít posloupnost regulačních (akčních) zásahů, pomocí kterých se z libovolného stavu stavového prostoru dostaneme d počátku v minimálním počtu kroků.

Kvadratické kritérium:

$$J = \sum_{j=0}^{N-1} [x^T(j) \cdot Q \cdot x(j) + u^T(j) \cdot P \cdot u(j)] + x^T(N) \cdot S \cdot x(N) \tag{1.39}$$

Matice Q, P a S jsou symetrické a pozitivně definitní (mají kladná vlastní čísla). Pro SISO systémy je matice P skalárem (je pouze číslem). Pomocí těchto matic lze ovlivnit rychlost a kvalitu regulačního pochodu.

Touto metodou lze navrhnout tvrdý (regulátor bez penalizace u) a měkký (regulátor s penalizací u) regulátor.

Matice Q a S můžeme volit takto:

$$Q = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad S = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 100 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Volba vysokého čísla v matici S povede k rychlému zklidnění regulačního pochodu. Rozměr matic závisí na řádu systému.

Pro tvrdý regulátor se matice P = 0 a kvadratické kritérium může vypadat například takto

$$J = \sum_{j=1}^{N-1} [x_3^2(j) + 100x_2^2(N)] \tag{1.40}$$

Pro měkký regulátor se matice P = 1 a kvadratické kritérium může vypadat například takto

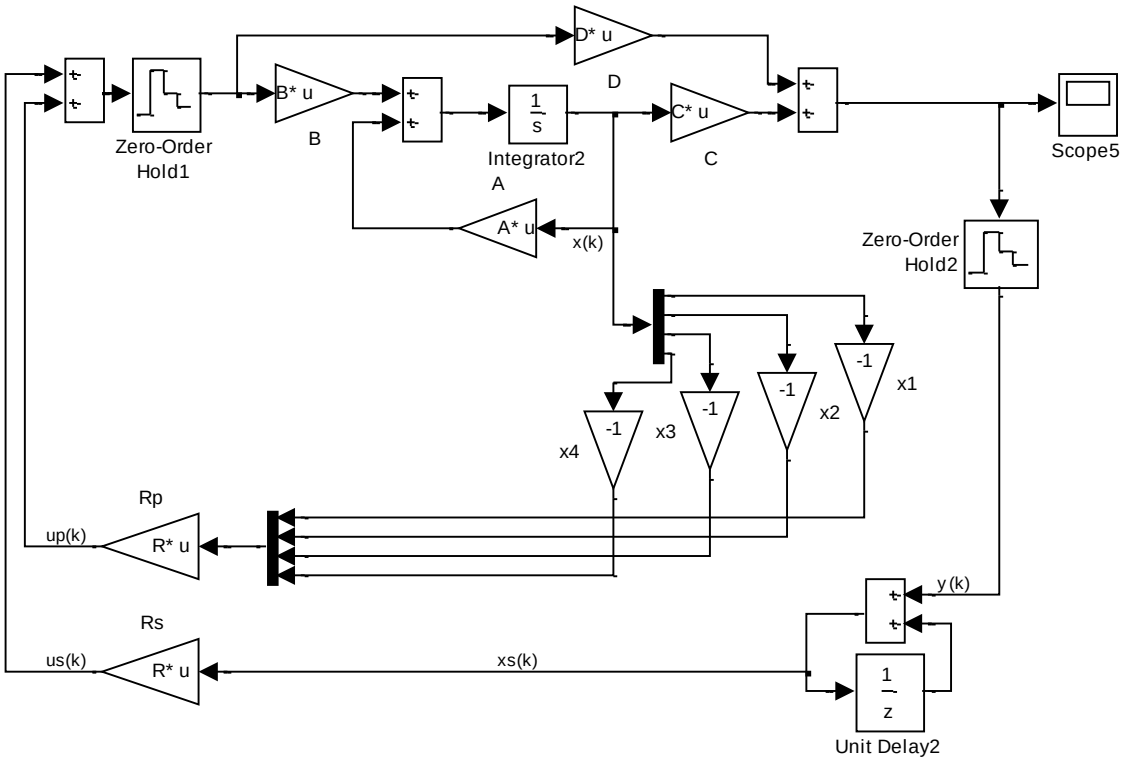
$$J = \sum_{j=1}^{N-1} [x_3^2(j) + u^2(j) + 100x_2^2(N)] \tag{1.41}$$

Pozn.: Výše uvedená kritéria se vážou ke stejné řízené soustavě a slouží pouze pro ilustrační účel. Pro jiné soustavy se kvadratické kritérium může lišit.

1.5 Diskrétní regulační obvod s astatickou složkou

Tento přístup využijeme, pokud budeme chtít, aby se stavový regulátor svým chováním blížil PI regulátoru.

Principem je přidání druhého regulátoru R_s, jehož vstupem je měřená stavová proměnná x_s. Schéma regulačního obvodu je na Obrázku 1.5.



Obrázek 1.5: Diskrétní regulační obvod s astatickou složkou

Uvažujeme popis dynamického systému podle rovnic (1.11) a (1.12) s předpokladem, že matice $D = 0$

$$x(k+1) = Mx(k) + Nu(k)$$

$$y(k) = Cx(k)$$

Popis astatické části je následující:

$$x_s(k) = x_s(k+1) + y(k) \tag{1.42}$$

$$\begin{aligned} x_s(k+1) = x_s(k) + y(k+1) &= x_s(k) + C \cdot \big(Mx(k) + Nu(k)\big) \\ &= x_s(k) + CMx(k) + CNu(k) \end{aligned} \tag{1.43}$$

Výsledný akční zásah je roven:

$$u(k) = u_p(k) + u_s(k) = R_p \cdot x(k) + R_s \cdot x_s(k) = -R_R \cdot \begin{bmatrix} x(k) \\ x_s(k) \end{bmatrix} \tag{1.44}$$

Sloučením rovnic (1.11) a (1.43) a rozšířením rovnice (1.12) získáme:

$$\begin{bmatrix} x(k+1) \\ x_s(k+1) \end{bmatrix} = \begin{bmatrix} M & 0 \\ CM & E \end{bmatrix} \cdot \begin{bmatrix} x(k) \\ x_s(k) \end{bmatrix} + \begin{bmatrix} N \\ CN \end{bmatrix} \cdot u(k) \tag{1.45}$$

$$y(k) = [C \quad 0] \cdot \begin{bmatrix} x(k) \\ x_s(k) \end{bmatrix} \tag{1.46}$$

kde

$$\begin{bmatrix} M & 0 \\ CM & E \end{bmatrix} = M_R$$

$$\begin{bmatrix} N \\ CN \end{bmatrix} = N_R$$

$$[C \quad 0] = C_R$$

Matici regulátoru R navrhujeme na výše uvedený systém (rovnice (1.45) a (1.46)) popsaný maticemi M_R, N_Ra C_R. Pro výpočet můžeme využít jednu z výše popsaných metod (kapitoly 1.5.1 a 1.5.2).

$$R_R = [R_P \quad R_S] = -[0 \quad 0 \quad \cdots \quad 1]{G_R}^{-1}{M_R}^{n+1} \tag{1.47}$$

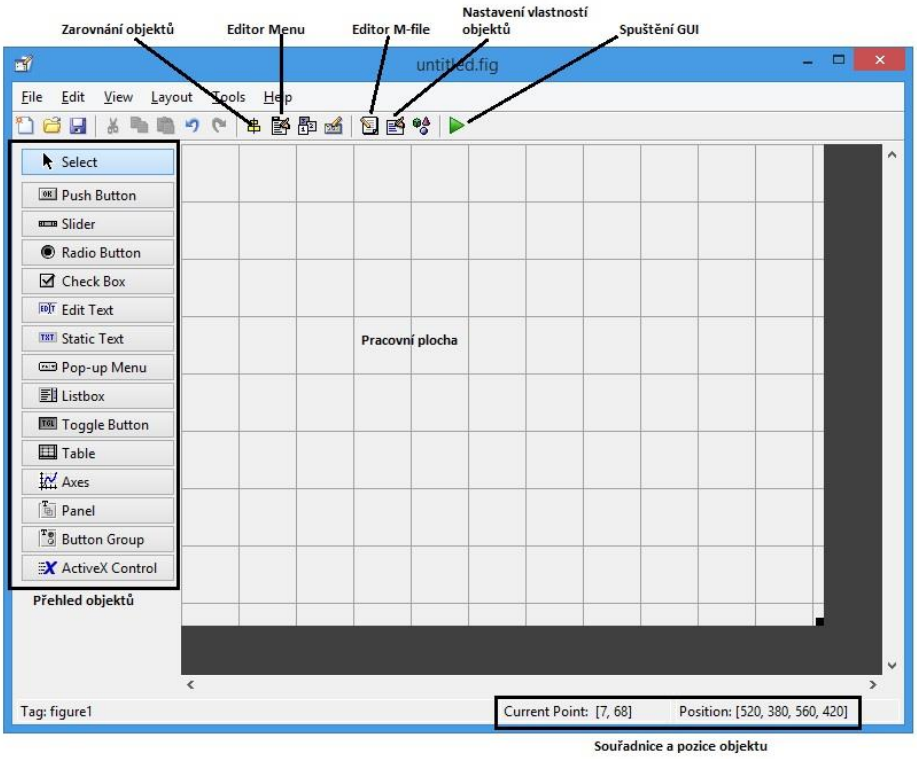
2 Tvorba grafického uživatelského rozhraní

Grafické uživatelské rozhraní (zkráceně GUI – Graphical User Interface) je rozhraní sestavené z grafických komponent jako jsou tlačítka, posuvníky, pop-up menu, textová pole apod. Hlavní funkcí GUI je nabídnout uživateli rychlejší, jednodušší a přehlednější editaci parametrů podřízeného kódu a zároveň ulehčit práci s výsledky jako například jejich ukládání, editaci nebo zobrazení.

Pro tvorbu GUI v programu MATLAB slouží vývojové prostředí GUIDE (Graphical User Interface Development Environment). Toto vývojové prostředí obsahuje soustavu nástrojů, které celý proces návrhu a programování GUI značně ulehčují. GUIDE také generuje výsledný M-file, který obsahuje výsledný kód pro ovládání, inicializaci a spouštění GUI. Základem každého takového M-filu je inicializační část a dále už obsahuje jen tzv. callback funkce (funkce, které se vykonají po aktivaci objektu v GUI). Obsah těchto funkcí je uživatelem volně programovatelný.

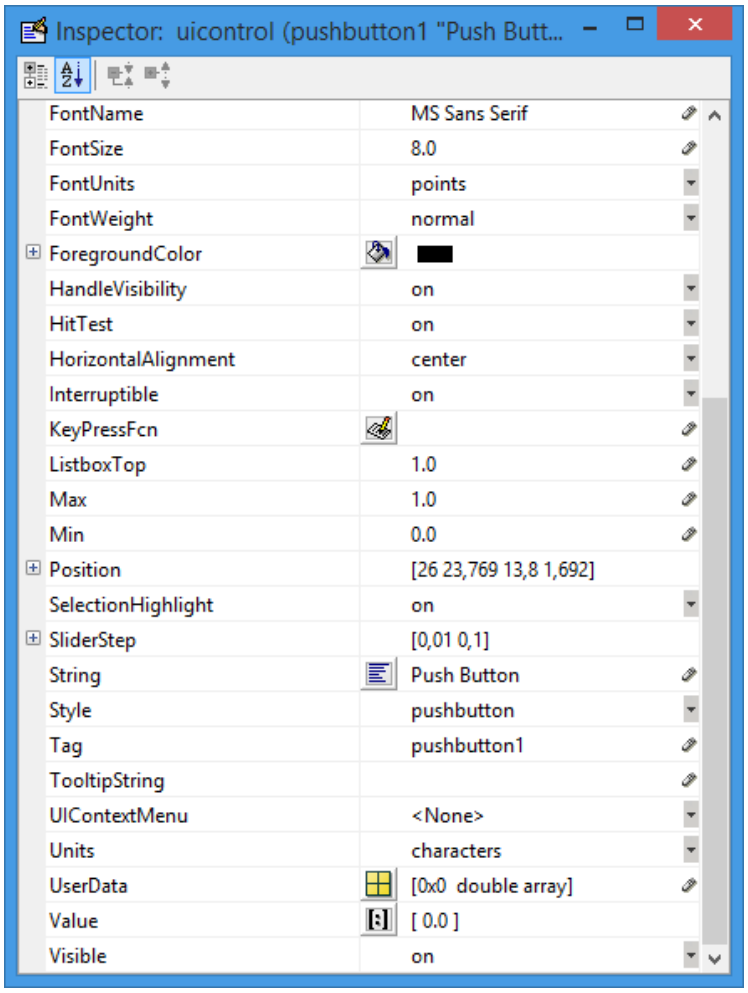
2.1 Matlab GUIDE

Pro spuštění tohoto vývojového prostředí použijeme v příkazové řádce programu MATLAB příkaz „guide“. V nově otevřeném okně lze vybrat, jestli chceme vytvořit GUI prázdné nebo z předem připravené šablony. Po potvrzení vybrané volby se otevře tzv. Layout editor (Obrázek 2.1), ve kterém probíhá samotný grafický návrh GUI.



Obrázek 2.1: Layout editor

V tomto editoru lze jednoduše vytvářet grafickou podobu GUI přetažením objektů, jako jsou například pop-up menu, tlačítka, posuvníky, grafy atd., do pracovní plochy. Po umístění objektu na pracovní plochu lze pomocí tzv. Property Inspectoru (Obrázek 2.2) měnit vlastnosti objektu, jako jsou například texty, velikosti, datové typy, limity atd.



Obrázek 2.2: Property Inspector

Každý objekt obsahuje určité callback funkce, jako je například vytvoření objektu, jeho smazání nebo stisk tlačítka. Přehled těchto funkcí lze vidět, pokud klikneme pravým tlačítkem myši na objekt na pracovní ploše pod položkou „View Callbacks“.

Programovat obsah callback funkcí lze v M-filu. Princip programování je jednoduchý a jeho princip spočívá ve dvou krocích. Prvním krokem je získání nebo nastavení vlastností vybraného nebo jiného prvku. K tomuto účelu se používají funkce „set“ a „get“. Použití těchto funkcí záleží na tom, jestli chceme pracovat s vlastnostmi vybraného nebo jiného objektu. Například jestli chceme po stisku tlačítka načíst / změnit hodnotu jiného objektu, nebo jestli chceme po stisku tlačítka měnit / číst jeho vlastnosti. Pokud bychom uvažovali

první příklad, využívá se klíčové slovo „handles“, které funguje jako ukazatel na jiný objekt. Funkce by vypadala následovně:

```
hodnota = get(handles.jmeno_objektu,'Value'); %přečtení hodnoty
set(handles.jmeno_objektu,'Vlastnost','hodnota'); %nastaveni hodnoty
```

Zdrojový kód 2.1: Matlab GUI – příklad použití ukazatele na jiný objekt

V druhém případě by se použilo klíčové slovo „hObject“, které říká, že budeme pracovat s vybraným objektem a funkce by vypadala takto:

```
hodnota = get(hObject,'Value'); %přečtení hodnoty
set(hObject,'Vlastnost','hodnota'); %nastaveni hodnoty
```

Zdrojový kód 2.2: Matlab GUI – příklad použití ukazatele sama na sebe

Druhým krokem je samotné zpracování a vykonání naprogramované funkce. Mezi nejčastější úlohy patří například navolení parametrů a po stisku tlačítka jejich odeslání, načítání hodnot a jejich následné zobrazení v grafu nebo jakékoliv zpracování dat.

Na níže uvedeném příkladu je funkce, která se vykoná se stiskem tlačítka. Principem této funkce je načtení 4 číselných hodnot z jiných objektů, jejich převedení na formát ip adresy a následné nastavení tcpip komunikace. Číselné hodnoty byly již načteny v jiné funkci, a proto jsou zde zavedeny jako globální proměnné, aby je bylo možné číst. Dále je zde kontrola, jestli byla všechna pole vyplněna, pokud ne, objeví se dialogové okno s chybovou hláškou. V poslední části je definice tcpip komunikace.

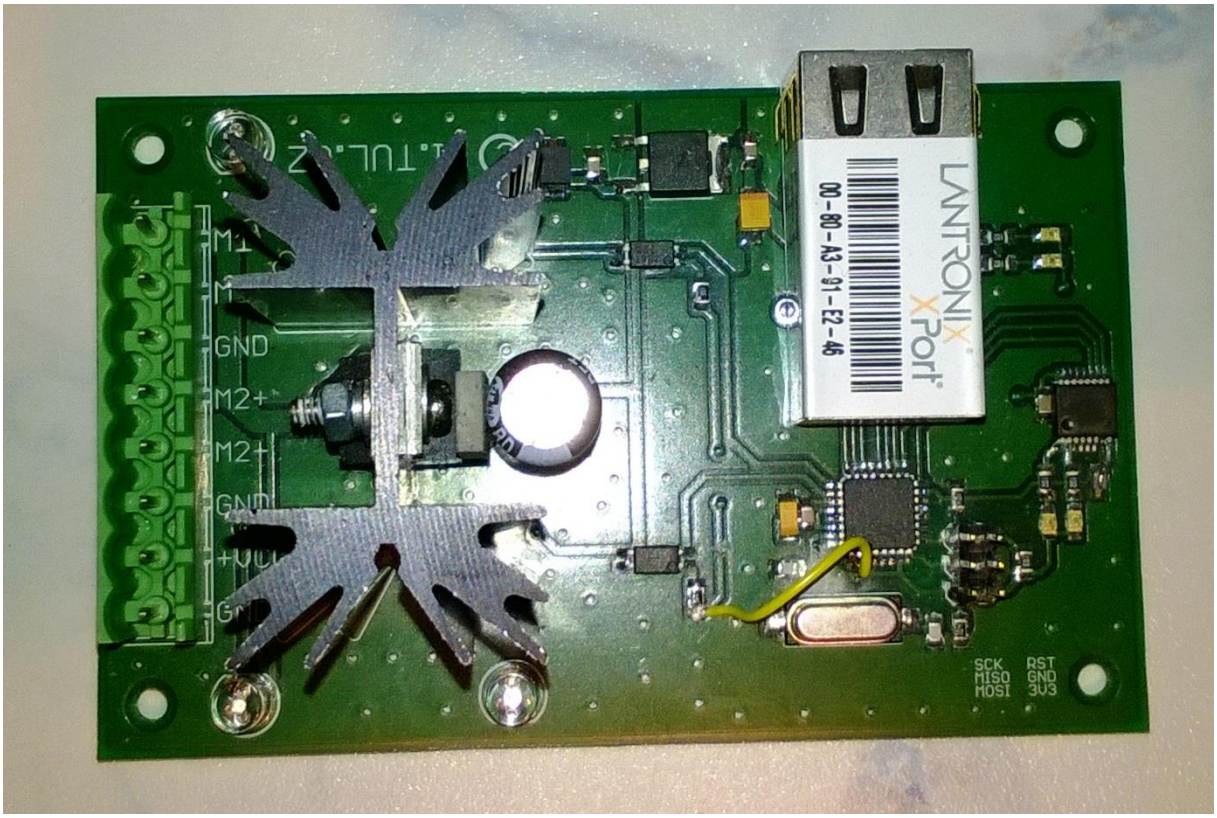
Příklad funkce:

```
% --- Executes on button press in nastaveni.
function nastaveni_Callback(hObject, eventdata, handles)
% hObject    handle to nastaveni (see GCBO)
% eventdata  reserved - to be defined in w future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global t port ip1p ip2p ip3p ip4p
adresa = [ip1p '.' ip2p '.' ip3p '.' ip4p];
if (port == 0)
       port = 80;
       set(handles.Port,'String',port);
end
if (isempty(ip1p) | isempty(ip2p) | isempty(ip3p) | isempty(ip4p))
       errordlg('Není zadaná IP adresa!','Bad Input','modal')
       uicontrol(hObject)
       return
else
       t = tcpip(adresa,port); %147.230.185.127,10001
       set(t,'NetworkRole', 'client');
       set(t,'timeout', 0.5);
       set(t,'InputBufferSize', 100);
       t.terminator = 'CR/LF';
       t.ReadAsyncMode = 'manual';
end
```

Zdrojový kód 2.3: Matlab GUI – příklad funkce

3 Popis fyzikálního modelu

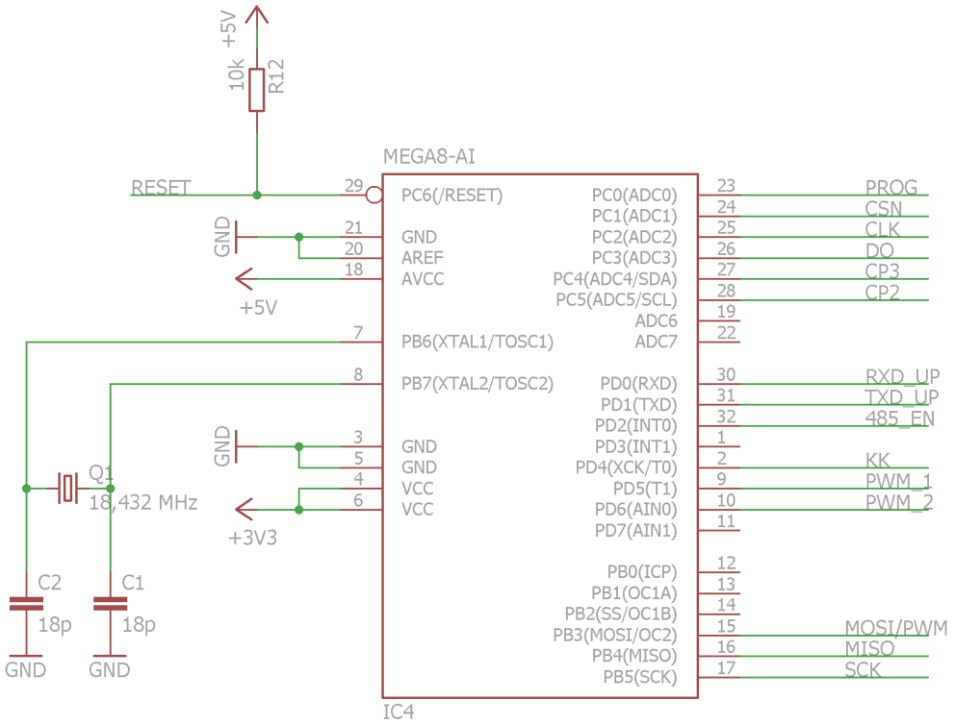
Konstrukce modelu je vyrobena z hliníkových profilů a skládá se ze dvou částí, z podstavce a rotující části. Jejich spojení je provedeno pomocí vodící nerezové tyče, která je uchycena v podstavci dvěma linearsety s kuličkovým pouzdrem pro přenos rotačního pohybu. K podstavci je připevněna deska plošného spoje s řídicí elektronikou (Obrázek 3.1). Rotující část je vytvořena z tenkého přímého hliníkového profilu délky 100 cm, na kterém jsou připevněny dva stejnosměrné elektromotory ve vzájemné vzdálenosti 86 cm. Elektromotory jsou typu MIG400 s maximálními otáčkami naprázdno 18 500 ot/min. Motory jsou osazeny dvou listou vrtulí s průměrem 15 cm.



Obrázek 3.1: Deska plošného spoje s řídicí elektronikou

Model je řízen procesorem ATmega328P od společnosti Atmel. Zapojení procesoru je vidět na Obrázku 3.2. Komunikace modelu je zajištěna ethernetovým portem XPort od společnosti Lantronix. Úhel natočení je snímán bezkontaktním snímačem natočení typu AS5140H, jehož rozsah je 0 – 360° s rozlišením 0,35°. Pro měření úhlu se vyžil dvoupólový magnet, který je umístěný nad snímačem a je připevněný na konci vodící, která se otáčí.

Fotografie celého modelu je k nahlédnutí v Příloze I. Kompletní schéma zapojení desky plošného spoje s řídicí elektronikou je zobrazen v Příloze VI.



Obrázek 3.2: Schéma zapojení pinů procesoru ATMega328P

3.1 Matematický model

Tah motoru je v různých zdrojích různě definován, ale ve výsledku se všechny shodují v tom, že tah je síla, která hýbe tělesem skrze vzduch. Tah je vyvozen z Newtonova zákona akce a reakce, kde akcí je prohnání vzduchu přes vrtuli a reakcí je potom síla tlačící vrtuli vpřed. Protože tvar listu vrtule je podobný konstrukci křídla, vzniká za vrtulí podtlak a před ní přetlak. Díky tomuto jevu vzniká rozdíl hybností vzduchu před a za vrtulí, díky kterému vzniká tah a pohyb vpřed. Lze také říci, že tah je síla, která je vyvozena urychlením vzduchu, který prochází vrtulí.

Působení této síly lze vyjádřit pomocí momentu síly, který popisuje otáčivý účinek síly. Moment síly M lze vypočítat vzorcem

$$M = J \cdot \dot{\omega} = J \cdot \varphi \tag{3.1}$$

kde J je moment setrvačnosti

ω je úhlová rychlost

φ je úhel natočení.

Moment síly je tvořen jednak otáčením motoru, tedy jeho tahem, a otáčením ramene, na kterém je umístěný motor. Výsledný moment síly je jejich rozdílem

$$M = k_1 \cdot \omega_m - k_2 \cdot \omega_s \tag{3.2}$$

kde k_1 a k_2 jsou konstanty s rozměrem $\left[\frac{kg \cdot m^2}{s \cdot rad}\right]$

Za předpokladu, že $\omega = \dot{\varphi}$ a $\dot{\omega} = \ddot{\varphi}$ můžeme rovnici (3.2) přepsat na:

$$J \cdot \ddot{\varphi} = k_1 \cdot \omega_m - k_2 \cdot \dot{\varphi} \tag{3.3}$$

Úpravou rovnice (3.3) získáme obrazový přenos prvního řádu se setrvačností

$$\frac{k_1 \cdot \omega_m}{(J \cdot s + k_2) \cdot s} \tag{3.4}$$

Tímto jsme získali matematický popis modelu, který neuvažuje dynamiku motoru. Aby tomu tak bylo, je potřeba vyjádřit přenos motoru pro otáčky ω_m. Uvažujeme následující přenos druhého řádu:

$$\omega_m = \frac{k_m}{s^2 + k_{m1} \cdot s + k_{m2}} \tag{3.5}$$

Protože se již uvažuje i dynamika motoru, je potřeba počítat i se zátěžovým momentem motoru, který je roven

$$M_z = \omega_m \cdot B \tag{3.6}$$

kde B je koeficient zátěže.

Dosazením rovnice (3.6) do (3.2) získáme výslednou rovnici pro výsledný moment síly se započítáním dynamiky motoru:

$$M = k_1 \cdot \omega_m - \omega_m \cdot B - k_2 \cdot \omega_s \tag{3.7}$$

Abychom získali výsledný obrazový přenos modelu, dosadíme do (3.4) rovnici (3.5):

$$\frac{(k_1 - B) \cdot k_m}{J \cdot s^4 + (k_2 + J \cdot k_{m1}) \cdot s^3 + (k_{m1} \cdot k_2 + J \cdot k_{m2}) \cdot s^2 + k_{m2} \cdot k_2 \cdot s} \tag{3.8}$$

4 Identifikace modelu a návrh stavového regulátoru

Pro identifikaci naměřených dat jsem zvolil metodu polyedrického hledání, která spočívá v určení směru hledání v n-rozměrném prostoru.

Základní idea hledání optima s použitím simplexu spočívá v postupném rušení jeho vrcholů, ve kterých je hodnota účelové funkce z pohledu kritéria optimalizace „nejhorší“ a jejich náhradou vrcholem „lepším“. Simplex se tak „pohybuje“ ze startovací polohy směrem k extrému účelové funkce.

V roce 1965 Nelder a Mead [11] navrhli modifikaci výše popsané metody, která spočívá v úpravě rozměru simplexu podle úspěšnosti či neúspěšnosti zobrazení nového vrcholu. Navrhli, aby v případě, kdy došlo k úspěšnému zobrazení (hodnota účelové funkce se zlepšila), se pokračovalo v tomto směru vyhledávání a aby byl simplex v daném směru „protažen“. V případě neúspěšného pokusu by se měl simplex „zkrátit“. Do nového simplexu doporučili potom zařadit ze dvou vrcholů vždy ten lepší. V literatuře se toto nazývá „pružný polyedr“, protože se simplex svým tvarem přizpůsobuje tvaru povrchu účelové funkce. Jednoduše lze říci, že ve směru k optimu se simplex „prodlužuje“, naopak kolmo na směr k optimu se „zkracuje“.

V programu Matlab lze pro tuto optimalizaci použít funkci *„fminsearch“*. Tato funkce hledá minimum vícerozměrové funkce ze zadaných počátečních bodů.

Volání funkce je následující:

```
X = fminsearch('function',x0,OPTIONS)
```

Zdrojový kód 4.1: Matlab - příklad volání funkce fminsearch

kde 'function' reprezentuje funkci, kterou chceme optimalizovat
x0 jsou počáteční body pro řešení optimalizace
OPTIONS obsahuje nastavení parametrů optimalizace.

Použitý zdrojový text M-filu pro identifikaci soustavy:

```
clc
global tG uG yG

%vstupní data
tG=t;
uG=U;
yG=Y;
figure;plot(tG,uG,tG,yG); %vykreslení vstupních dat
break
%počáteční odhad parametrů
a=[ ]
K= ;
x=[K a] %vektor hledaných parametrů
critT(x) %volání funkce hledající minimum
disp('running...')
%minimalizace kritéria J
%nastavení parametrů minimalizace pro funkci fminsearch
OPTIONS=optimset('TolFun',1e-6,'MaxFunEvals',200,'Display','iter');
x=fminsearch('critT',x,OPTIONS);
disp('optimalizovany vektor x')
x
disp('Hodnota kriteria J')
critT(x)

B=x(1);
A=[x(2:5) 1];
sys=tf(B,A);
roots(A)
%reakce nalezeného systému na původní buzení
[yi,ti]=lsim(sys,uG,tG);
%porovnání reakce nalezeného systému s původními daty
figure;plot(tG,yG,ti,yi);
```

Zdrojový kód 4.2: Matlab - zdrojový kód použité funkce pro identifikaci soustavy

Hledání minima probíhá v samostatné kriteriální funkci:

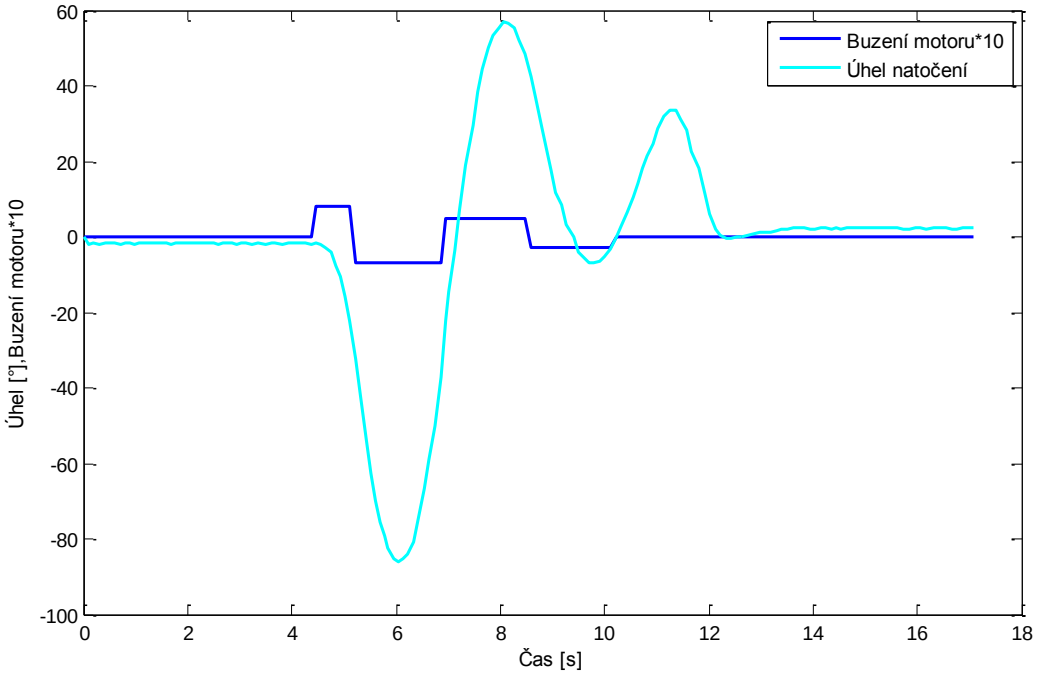
```
function f=critT(x)
global tG uG yG
%% přiřazení proměnných, které popisují systém = dekódování x
B=x(1);
A=[x(2:length(x)) 1];
sys=tf(B,A); %obrazový přenos
[yi,ti]=lsim(sys,uG,tG); %výpočet odezvy systému na buzení
plot(tG,yG,ti,yi); %vykreslení do grafu
pause(0.1);
%% přiřazení hodnoty do kritéria

f=sum((yi-yG).*(yi-yG));
```

Zdrojový kód 4.3: Matlab - zdrojový kód kriteriální funkce

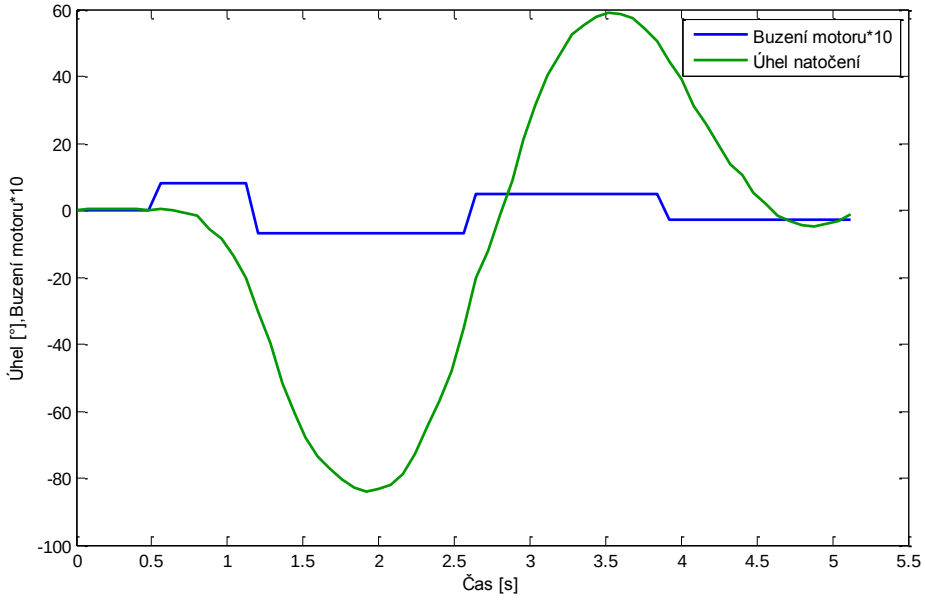
4.1 Identifikovaný matematický model soustavy

Vzhledem k tomu, že se jedná o rotační soustavu, která bez regulátoru nedosáhne ustálené hodnoty polohy, použil jsem pro identifikaci dynamickou charakteristiku. Tu jsem získal tak, že jsem provedl několik skokových změn hodnoty PWM modulace, která řídí vstupní napětí motorů, a sledoval dynamické chování soustavy. Naměřená data, ze kterých jsem při identifikaci vycházel, jsou vidět na Grafu 4.1.



Graf 4.1: Naměřená dynamická charakteristika

Aby bylo možné výše naměřená data využít pro identifikaci, bylo potřeba je upravit. Úprava spočívala v odstranění neužitečných dat. V tomto měření to jsou data z počátku měření před první skokovou změnou hodnoty buzení motoru. Dále byla odstraněna data z konce měření, protože bylo zapotřebí ručního ustálení, a tak data nemají žádnou užitečnou hodnotu.

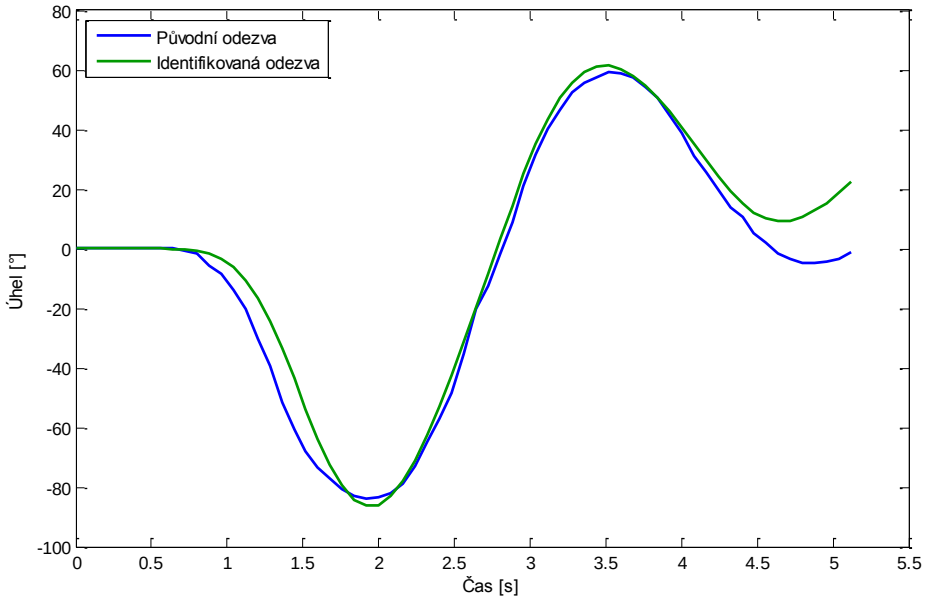


Graf 4.2: Vstupní data pro identifikaci

Takto upravená data sloužila jako vstup do výše uvedeného M-filu, který obsahuje funkci pro identifikaci přenosové funkce. Výstupem byla tato přenosová funkce:

$$F(s) = \frac{-231960}{43.26 \cdot s^4 + 128.4 \cdot s^3 + 887 \cdot s^2 + 486.8 \cdot s} \tag{4.1}$$

Při porovnání původního průběhu s identifikovaným (Graf 4.3) lze vidět, že oba průběhu jsou s menšími odchylkami totožné, a je tedy možné použít přenosovou funkci (4.1) pro popis tohoto modelu.



Graf 4.3: Porovnání původního systému s identifikovaným

4.2 Návrh stavového regulátoru a estimátoru

Pro návrh stavového regulátoru a estimátorujsem využil mnou naprogramovaný M-file v programu Matlab. Vstupním parametrem je přenosová funkce systému, který chceme řídit, a výstupem je matice stavového regulátoru R a matice estimátoru L. Tento M-file se vykonává v následujících krocích:

1. Převod přenosové funkce do stavového popisu.
2. Převod spojitého stavového popisu na diskrétní stavový popis.
3. Výpočet matice estimátoru L pomocí Ackermannovy formule.
4. Výpočet matice regulátoru R pomocí Riccatiho rovnice.

Použitý zdrojový text M-filu pro návrh regulátoru:

```
disp('Sestavení přenosové funkce')
sys=tf(-2.3196e+05,[43.2550 128.4183 887.0161 486.7940 0])
[num,den] = tfdata(sys,'v');
rad=length(roots(den));

disp('Převod na spojitý stavový popis')
[A,B,C,D]=ssdata(ss(sys)) %sestavení stavového popisu
A=fliplr(flipud(A));
B=fliplr(flipud(B));
C=fliplr(flipud(C));
D=fliplr(flipud(D));


disp('Převod na diskrétní stavový popis')
pz=0.05; %perioda vzorkování
sys2=ss(A,B,C,D)
sysd=c2d(sys2,pz) %převod spojitého stavového popisu na diskrétní
[M,N,C1,D1]=ssdata(sysd)


disp('Návrh estimátoru pomocí Ackermanovy funkce')
LambdaE=[0 0 0 0]; %minimální počet kroků
L=acker(M',C1',LambdaE); %výpočet matice estimátoru L
L=L'


disp('Návrh diskrétního stavového regulátoru')
Mr=[M zeros(rad,1);C1*M eye(1)];
Nr=[N;C1*N];
Cr=[C1 0];
Q=eye(rad+1);
Q(1,1)=10; % minimální počet kroků
%výpočet Riccatiho rovnice
Pj=Q;
for I=1:1000;
   I=I;
   Gj=1+Nr'*Pj*Nr;
   Pj=Q+Mr'*Pj*(eye(rad+1)-Nr*(Gj^(-1))*Nr'*Pj)*Mr;
end;
R=(Gj^(-1))*Nr'*Pj*Mr
```

Zdrojový kód 4.4: Matlab – script pro návrh matice stavového regulátoru

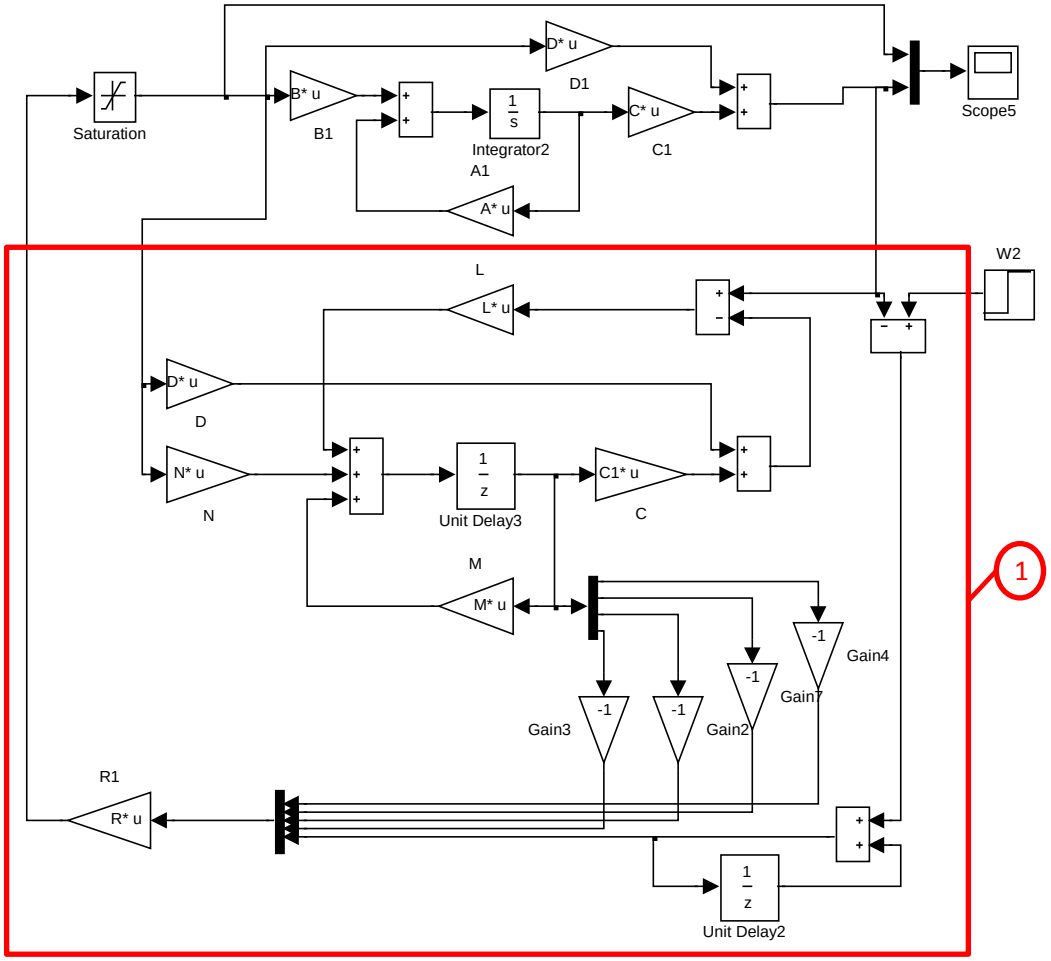
Výstupem byly následující matice:

$$L = \begin{bmatrix} -0.0910 \\ -3.7640 \\ -23.3364 \\ -24.3634 \end{bmatrix} \tag{4.2}$$

$$R = [102{,}5532 \quad 11.1740 \quad 2.6741 \quad 0.7543 \quad -0.3158] \tag{4.3}$$

5 Převod stavového regulátoru na funkci

Hlavním cílem této práce bylo implementovat stavové řízení do procesoru modelu. To se skládá ze dvou funkcí, estimace vnitřních stavů a regulace soustavy. Vizuálně je to zobrazeno na Obrázku 5.1, na kterém je v programu Simulink vytvořen regulační obvod s estimátorem a stavovým regulátorem. V červeném rámečku je vyznačena oblast, kterou je potřeba implementovat do procesoru.



Obrázek 5.1: Grafické vyznačení části modelu (1), jejíž funkci se bude programovat

Podle kapitoly 1.4 (rovnice (1.24) a (1.27)), platí pro estimaci vnitřních stavů a výstupu následující vztahy:

$$\hat{x}(k+1) = M\hat{x}(k) + Nu(k) + L \cdot \big(y(k) - \hat{y}(k)\big)$$

$$\hat{y}(k) = C\hat{x}(k) + Du(k)$$

Pro výpočet akčního zásahu regulátoru platí rovnice (1.44)

$$u = -R \cdot \begin{bmatrix} x_e \\ x_s \end{bmatrix}$$

Výše uvedené rovnice jsem převedl do matlabovského kódu a vytvořil funkci, jejíž vstupními parametry jsou žádaná hodnota *w* a výstupní veličina *y*. Výstupem této funkce je vypočtený akční zásah *u*.

Ve funkci se nejprve spočítají estimace stavových veličin, které slouží jako vstupní parametry pro výpočet akčního zásahu.

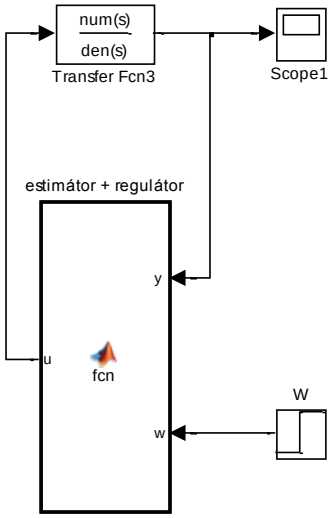
Použitý zdrojový kód funkce pro výpočet akčního zásahu:

```
function [u] = fcn(y,w)
%#codegen
persistent e e_soucet_past e_soucet y_rozdil soucet soucet_past soucet_R v
xe;
%inicializace proměnných při prvním spuštění
if isempty(v)
    e_soucet_past = 0;
    v = 0;
    soucet_past = [0;0;0;0];
    soucet_R = [0;0;0;0;0];
    xe = [0;0;0;0];
    e_soucet = 0;
end

M = []; %definice matic stavového popisu
N = [];
C = [];
D = [];
R = [];
L = [];
%výpočet stavové proměnné Xs
e = w - y;
e_soucet = e + e_soucet_past;
e_soucet_past = e_soucet;
%výpočet estimovaného výstupu
ye = D*v + C*soucet_past;
%výpočet rozdílu skutečné a estimované hodnoty výstupu y
y_rozdil = y - ye;
%výpočet budoucích stavových veličin
soucet = L*y_rozdil + N*v + M*xe;
%výpočet stavových veličin
xe = N*v + M*xe;
x1e = xe(1,1);
x2e = xe(2,1);
x3e = xe(3,1);
x4e = xe(4,1);
soucet_past = soucet;
%sestavení vektoru stavových veličin
soucet_R = [-1*x1e;-1*x2e;-1*x3e;-1*x4e;e_soucet];
%výpočet akčního zásahu a jeho následné omezení v mezích
v = R*soucet_R;
if v >= 1
    v = 1;
elseif v <= -1
    v = -1;
else
    v = v;
end
u = v;
```

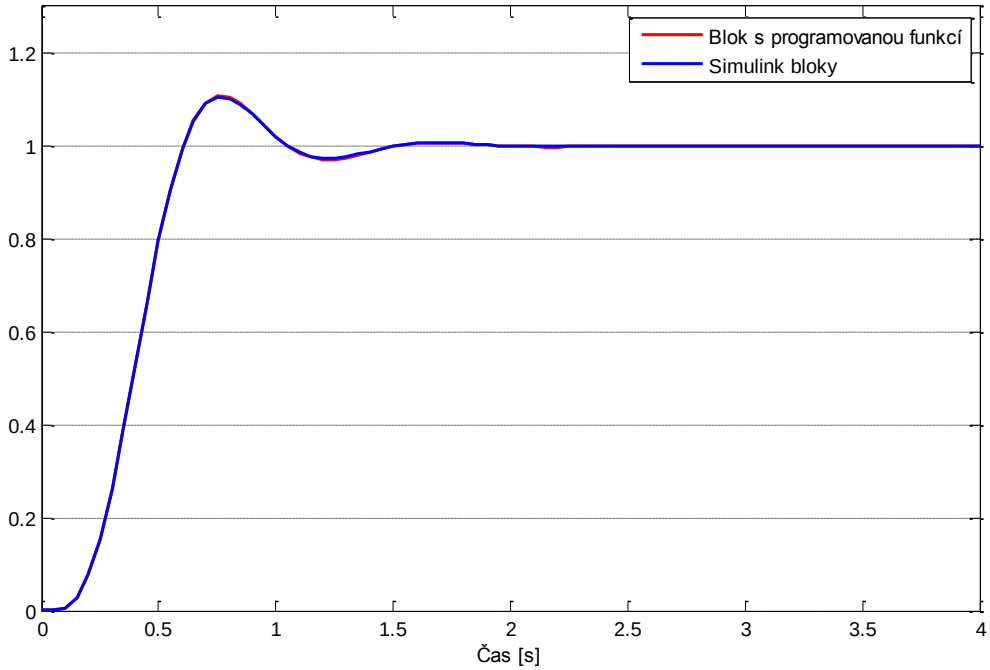
Zdrojový kód 5.1: Matlab - kód výpočtu akčního zásahu stavového regulátoru

Pro otestování takto vytvořené funkce jsem v programu Simulink vytvořil blokové schéma (viz Obrázek 5.2), které obsahuje blok s přenosovou funkcí a programovatelný blok, jehož obsahem je výše uvedená funkce.

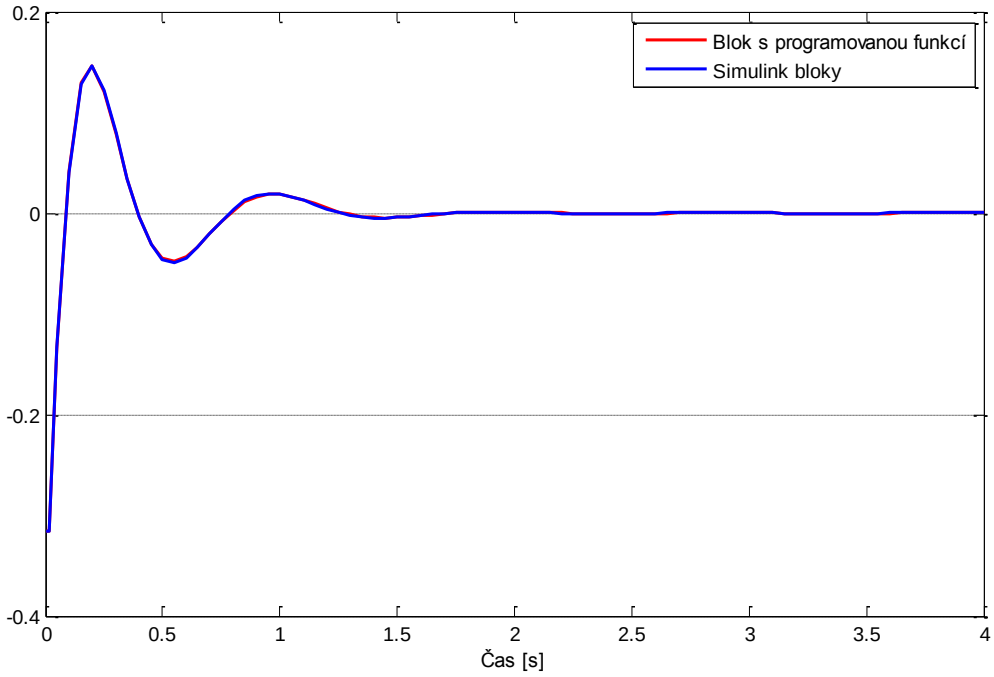


Obrázek 5.2: Model v Simulinku s blokem obsahující naprogramovanou funkci

Naměřená data, akční zásah a výstupní veličinu, z výše uvedeného modelu (Obrázek 5.2) jsem porovnal s daty, která jsem získal z měření na modelu podle Obrázku 5.1. Jejich porovnání je vidět na Grafech 5.1 a 5.2. Měření se prováděla s periodou vzorkování $T = 0{,}05\ s$.



Graf 5.1: Porovnání výstupní veličiny z blokového schématu a programované funkce



Graf 5.2: Porovnání akčního zásahu z blokového schématu a programované funkce

Jak je vidět na grafech výše, naměřená data jsou téměř identická a lze tedy říci, že naprogramovaná funkce odpovídá fungování estimátoru se stavovým regulátorem.

6 Popis funkce a kódu mikroprocesoru

Jak již bylo uvedeno v popisu modelu (kapitola 3) programovaným mikroprocesorem byl typ ATmega328P od společnosti Atmel. Programování se provádělo v programu Atmel Studio 6.

Úkolem této diplomové práce byla úprava stávajícího kódu, proto jsem programoval pouze hlavní funkci mikroprocesoru. Nastavení mikroprocesoru, některé převodové a komunikační funkce jsem převzal z již funkčního programu, který jsem měl k dispozici jako předlohu.

Program lze rozdělit na dvě hlavní části. První se věnuje příjmu a třídění vstupních dat. Zde se určuje, jestli se budou načítat prvky matic, nebo jestli se bude pouštět regulační cyklus. Ve druhé části probíhá regulace a odesílání dat o aktuálním úhlu natočení a vypočítaném akčním zásahu.

Na začátku programu je prováděno přijímání dat z PC. Každý řetězec začíná vodícím znakem, jejich přehled je v Tabulce 6.1, a končí středníkem.

Vodící znak	Význam
M	Příjem matice M
N	Příjem matice N
C	Příjem matice C
D	Příjem matice D
R	Příjem matice R
L	Příjem matice L
T	Spuštění programu
P	Zastavení programu
Q	Řád soustavy
W	Příjem žádané hodnoty

Tabulka 6.1: Přehled vodících znaků

Příjem a třídění vstupních dat je prováděn v samostatné smyčce *while*, a to proto, že funkce pro načítání vstupního řetězce vrací hodnotu 0 /1 podle toho, jestli došlo ke čtení dat nebo nikoliv. V případě, že již nedochází k načítání dat, funkce bude vracet hodnotu 0 a dokud se nebudou přijímat nějaká data, bude docházet k přerušení této smyčku a nebude se vykonávat příjem a třídění vstupních dat. Bude se pokračovat rovnou na část programu s regulací.

```
while(1)
{
	rx_int_disable;
	pomc = cbuffer_get(&buffer_in,&znak);
	rx_int_enable;
	if (!pomc) break;
	vstup[i] = znak;
	.
	.
	.
}
```

Zdrojový kód 6.1: Atmel Studio - příjem znaku a jeho zapsání do proměnné

Načítaný řetězec se postupně ukládá do proměnné *znak* typu pole. První znak je vždy již výše zmíněný vodící znak. Pomocí příkazů if probíhá třídění na základě vodícího znaku a nastaví se hodnota pro další krok.

```
if ((znak == 'M') || ( nak == 'Q') || (znak == 'N') || (znak == 'C') ||
(znak == 'D') || (znak == 'R') || (znak == 'W') || (znak == 'B') || (znak ==
'T') || (znak == 'P') || (znak == 'L') || (znak == 'Q'))
	{
		prikaz_uart = znak;

		if ((znak == 'T') || (znak == 'P'))//start stop
		{
			state_uart = 1;
		}
		if ((znak == 'M') || (znak == 'N') || (znak == 'C') || (znak ==
		'D') || (znak == 'R') || (znak == 'L') || (znak ==
		'Q'))//načtení matice
		{
			state_uart = 2;
		}
		if ((znak == 'W'))//načtení čísla
		{
			state_uart = 2;
		}
		}
```

Zdrojový kód 6.2: Atmel Studio - zpracování vodícího znaku

V dalším kroku se vyhodnocuje stav proměnné *state_uart*. Pokud úspěšně proběhlo načtení celého řetězce, nastaví se na hodnotu 1 a přejde se na část jeho zpracování. Jestliže se hodnota proměnné rovná 3, znamená to, že stále probíhá příjem dat. Zároveň se zde kontroluje správnost přijatých znaků. Předpokladem je, že povolené znaky za vodícím jsou čísla, tečka, plus, mínus a středník, který řetězec ukončuje. V případě, že došlo k načtení jiného znaku, vyhodnotí se to jako chyba, a dojde k vynulování vstupního řetězce a bude se vyčkávat na nový příjem dat.

```
switch (state_uart)
{
case 1:
	proved_uart = 1;
	break;
case 2:
	state_uart = 3;
	break;
case 3:// načtení čísla a matice
	if (isdigit(znak) || (znak=='.') || (znak=='+') || (znak=='-'))
	{
		state_uart = 3;
	}
	else if (znak == ';')
	{
	proved_uart = 1;
	}
	else
	{
	error_uart = 1;
	}
	break;
case 10: //po chybě ve znaku se cyklicky zapisuje na pozici 0 ve vstupním
poli, dokud nepřijde vodící znak nového řetězce
	vstup[i] = znak;
	i = -1;
	break;
}
```

Zdrojový kód 6.3: Atmel Studio - zpracování následujících znaků po vodícím znaku

Po úspěšném přijetí řetězce se nastaví proměnná *proved_uart* na hodnotu 1, a tím dojde ke spuštění části programu, která řetězec zpracovává. Zde je opět provedeno třídění podle vodícího znaku. Pro kontrolu, zda-li nebyl přijímaný řetězec poškozen, je po každém jeho přijetí odeslán zpět do PC, kde dochází k jeho porovnání.

Načtou-li se znaky T nebo P, dojde k nastavení proměnné *run* na hodnotu 0 nebo 1 a tím ke spuštění nebo zastavení části programu s regulací modelu.

Bude-li vodící znak Q, dojde k nastavení proměnně *rad*, která reprezentuje řád soustavy, na základě druhého znaku řetězce.

```
case 'Q' ://řád systému
	tx_int_disable;
	cbuffer_put_data(&buffer_out,vstup,i);
	tx_int_enable;
	switch (vstup[1])
	{
		case '2':
			rad = 2; break;
		case '3':
			rad = 3; break;

		case '4':
			rad = 4; break;
	} break;
```

Zdrojový kód 6.4: Atmel Studio – převod řádu systému z řetězce na číslo

V případě, že přijatým řetězcem bude prvek matice (vodící znaky M, N, C, D, R a L), dojde nejprve k vynulování prvků, které jsou na pozicích vyšších, než je řád soustavy a tedy aktuální rozměr matice. Dalším krokem je rozdělení řetězce, který má pevný formát (podrobněji v kapitole 8). Nejprve se část obsahující číselnou informaci převede funkcí *atof()* na číslo a to se zapíše do proměnné *Npom*. Podle čtvrtého znaku řetězce se určí, jestli je prvek kladné nebo záporné číslo. Součástí přijatého řetězce na pozicích dva a tři jsou souřadnice prvku. Posledním krokem je nahrání obsahu proměnné *Npom* do proměnné, která odpovídá souřadnicím prvku, například N21. To znamená, že se jednalo o prvek matice N s pozicí ve druhém řádku prvního sloupce. Toto třídění je zajištěno funkcí *switch*.

```
case 'N' ://nulování prvků, které jsou „nad“ aktuálním rozměrem matice
        switch (rad)
        {
                case 2:
                        N31=0.0;N41=0.0; break;
                case 3:
                        N41=0.0; break;
        }
        tx_int_disable;
        cbuffer_put_data(&buffer_out,vstup,i);
        tx_int_enable;
        //zpracování na číslo
        for (j = 4;j < 10;j++)
        {
                N[l] = vstup[j];
                l++;
        }
        Npom = atof(N);
        if (vstup[3] == '+')
        {
                Npom = Npom;
        }
        else if (vstup[3] == '-')
        {
                Npom = 0 - Npom;
        }
```

Zdrojový kód 6.5: Atmel Studio – převod prvku matice z řetězce na číslo

Pokud proběhlo celé načítání dat v pořádku, bude se tento cyklus přerušovat hned na začátku a automaticky se budě provádět část programu s načítáním polohy a výpočtem akčního zásahu.

Pokračování v programu je podmíněno funkcí *if*, která sleduje hodnotu proměnné *time2sample*. Pokud bude rovna nule, skočí se na konec programu, jinak bude spočítána poloha a proběhne vyhodnocení proměnné *run*, která spouští nebo vypíná regulační cyklus. Znamená to tedy, že informace o poloze se zpracovává neustále, zatímco regulace se

vykonává pouze žádost uživatele. Nastavení proměnné *time2sample* na hodnotu 1 se provádí každých 0,05s v přerušení časovače 1.

Pokud byl uživatelem spuštěn program (hodnota proměnné *run* je rovna 1), spustí se motory a začne se vypočítávat akční zásah. Postup pro jeho výpočet byl odvozen v kapitole 5. Postup pro počítání s maticemi jsem nevolil pomocí funkcí, který by byl pravděpodobně efektivnější, nýbrž pomocí počítání jejich jednotlivých prvků. Tento postup jsem zvolil proto, že řád soustavy je proměnný a změna řádu soustavy tento výpočet nijak neovlivní, protože prvky matic, které jsou „navíc“ mají nulovou hodnotu.

Před aplikací akčního zásahu se provede jeho omezení v rozsahu od -1 do 1 a zároveň v případě příliš malých hodnot se jeho zdvojnásobí. Důvod pro zesílení malých akčních zásahů je popsán v kapitole 9.

```
e = W - y;
e_soucet = e + e_soucet_past;
e_soucet_past = e_soucet;
ye = D11*z + (C11*soucet_past11 + C12*soucet_past21 + C13*soucet_past31 +
C14*soucet_past41);
y_rozdil = y - ye;
soucet11 = L11*y_rozdil + N11*z + (M11*x1e + M12*x2e + M13*x3e + M14*x4e);
soucet21 = L21*y_rozdil + N21*z + (M21*x1e + M22*x2e + M23*x3e + M24*x4e);
soucet31 = L31*y_rozdil + N31*z + (M31*x1e + M32*x2e + M33*x3e + M34*x4e);
soucet41 = L41*y_rozdil + N41*z + (M41*x1e + M42*x2e + M43*x3e + M44*x4e);
x1e = N11*z + (M11*x1e + M12*x2e + M13*x3e + M14*x4e);
x3e = N31*z + (M31*x1e + M32*x2e + M33*x3e + M34*x4e);
x2e = N21*z + (M21*x1e + M22*x2e + M23*x3e + M24*x4e);
x4e = N41*z + (M41*x1e + M42*x2e + M43*x3e + M44*x4e);
soucet_past11 = soucet11;
soucet_past21 = soucet21;
soucet_past31 = soucet31;
soucet_past41 = soucet41;
z = -R11*x1e - R12*x2e - R13*x3e - R14*x4e + R15*e_soucet;
u = (float)z;
if (u >= 1.0) {
      u = 1.0;
}
else if (u <= -1.0) {
      u = -1.0;
}
else if  ((u <= 0.05) && (u > 0)) {
      u = 2*u;
}
else if ((u >= -0.05) && (u < 0)) {
      u = 2*u;
}
else   {
u = u;
}
reg_out(u);
```

Zdrojový kód 6.6: Atmel Studio - výpočet akčního zásahu stavového regulátoru

Po výpočtu a aplikaci akčního zásahu se provede poslední krok programu a to odeslání dat uživateli do PC. Odesílané jsou informace o aktuální poloze a akčním zásahu. Hodnoty se odesílají v hexadecimálním formátu, proto jsou nejprve převedeny na celočíselnou hodnotu a poté pomocí i*tox()* převedeny do hexadecimálního formátu a odeslána jako řetězec.

```
        akcni_zasah = (u + 1) * 32767.5;
        poloha = (y + 90) * 327.675;
        tx_int_disable;
        itox(akcni_zasah,pomA);
        itox(poloha,pomP);
        hex_string2[0] = '!';
        hex_string2[1] = pomA[0];
        hex_string2[2] = pomA[1];
        hex_string2[3] = pomA[2];
        hex_string2[4] = pomA[3];
        hex_string2[5] = pomP[0];
        hex_string2[6] = pomP[1];
        hex_string2[7] = pomP[2];
        hex_string2[8] = pomP[3];
        cbuffer_put_data(&buffer_out,hex_string2,9);
        tx_int_enable;
```

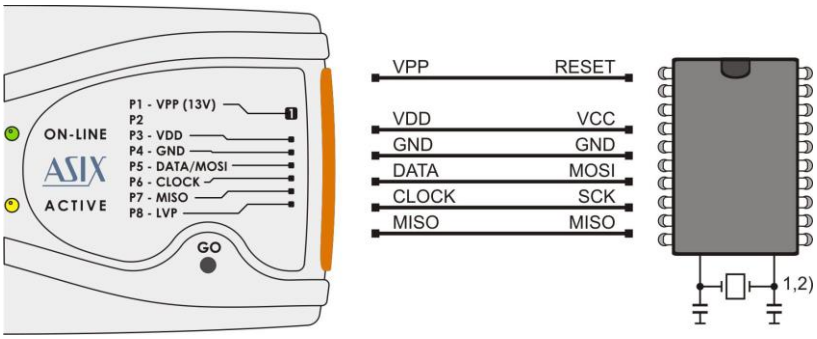
Zdrojový kód 6.7: Atmel Studio - převod a odeslání aktuálních hodnot

7 Programátor ASIX PRESTO

PRESTO je USB programátor [5] využitelný k programování různých součástek, jako jsou například mikroprocesory, sériové paměti EEPROM a další. Programátor je napájený pomocí USB a aplikaci může napájet napětím 5V nebo může během programování využít externí napájecí napětí z aplikace.

Obrázek 7.1: Programátor ASIX PRESTO s propojovacím kabelem

Programátor je s programovanou aplikací propojen pomocí programovacího kabelu, jehož popis pinů je popsán v tabulce 7.1 a schéma zapojení je uvedeno v Obrázku 5.2.



Obrázek 7.2: Schéma zapojení pinů programátoru PRESTO

Pin	Název	Typ	Popis
P1	VPP	I/O	VPP výstup
P2			
P3	VDD	PWR	Napájení vstup/výstup
P4	GND	PWR	Napájení vstup/výstup
P5	DATA/MOSI	I/O	Log. vstup/výstup
P6	CLOCK	O	Log. výstup
P7	MISO	I	Log. vstup
P8	LVP	I/O	Log. Vstup/výstup

Tabulka 7.1: Popis pinů programovacího kabelu

Pro práci s programátory od firmy ASIX se používá program UP. Tento program nabízí mnoho možností a nastavení spojených s programováním součástek. I přesto je jednoduchý na ovládání i pro méně zkušené uživatele, kteří zvládnou programování aplikace pomocí tří následujících kroků.

S každým spuštěním programu je potřeba vybrat programátor, který budeme používat. Pokud je k počítači připojený a nekomunikuje s jinými programy, zobrazí se jeho sériové číslo a bude možné spustit test komunikace. Tuto volbu lze kdykoliv změnit v nastavení programu.

Dále je potřeba vybrat programovanou součástku. Tu lze vybrat při prvním spuštění programu nebo kdykoliv změnit v projektu v kontextovém menu pod položkou *Součástka*.

Posledním krokem je výběr HEX souboru, který chceme do součástky nahrát. Po jeho výběru se jeho obsah zobrazí v okně *Paměť programu*. Nyní již stačí v kontextovém menu pod položkou *Součástka* vybrat možnost *Programovat*. Zobrazí se dialogové okno, které informuje o postupu, chybách a dokončení programování součástky.

Pozn.: Tento postup zjednodušený postup je použitelný za předpokladu, že programátor je již předem nastavený, jinak je potřeba postupovat dle dokumentace k programátoru.

8 Popis funkce a kódu GUI aplikace

Aplikace se skládá ze šesti základních ovládacích bloků. Nejprve je potřeba nastavit komunikaci s modelem. To se provede vyplněním ip adresy do položky *„Nastaveni IP adresy“* a následným stiskem tlačítka *„Nastavení TCP/IP“*.

Po nastavení ip adresy je potřeba nastavit žádanou hodnotu a stavový popis a odeslat je do modelu. U stavového popisu se nejprve vybírá řád systému a poté se objeví pole pro vyplnění matic. Pro jejich odeslání slouží tlačítko *„Odeslat nastavení“*. Vyplněný a odeslaný stavový popis lze stiskem tlačítka *„Uložit nastavení“* uložit do proměnné typu .mat. Soubor se uloží do složky, odkud byla aplikace spuštěna. Pro odeslání žádané hodnoty, je potřeba vyplnit hodnotu do pole *„Nastavení žádané hodnoty“* a stisknout tlačítko *„Odeslat žádanou hodnotu“*. Pokud došlo k nesprávnému odeslání dat, objeví se dialogové okno s informací o špatném odeslání. Po jeho potvrzení je možné odeslat data znovu.

Pokud odeslání všech hodnot proběhlo v pořádku, lze nyní spustit řízení modelu regulátorem stisknutím tlačítka *„Start“*. Pro vypnutí regulace slouží stejné tlačítko, na kterém se změnil text na *„Stop“*.

Po spuštění regulace modelu se zobrazují v objektu „Aktuální hodnoty“ přijímané hodnoty polohy a akčního zásahu. Také se zde zobrazuje aktuálně nastavení žádaná hodnota. Tyto hodnoty se zároveň vykreslují do grafu, ve kterém je možné sledovat hodnoty v grafické formě. Po zastavení programu se vykreslená data z grafu ukládají do .mat souboru a data v grafu se smažou, aby byl graf prázdný pro další spuštění.

V průběhu chodu regulace je možné měnit žádanou hodnotu, ale není možné měnit ostatní nastavení

Návrh aplikace se skládá ze dvou částí. První je návrh grafické části. Zde se řeší rozměry okna aplikace, komponenty a jejich rozmístění, případně další vlastnosti. Rozmístění komponent pro tuto aplikaci je možné vidět v Příloze II, která zobrazuje finální návrh ve vývojovém prostředí GUIDE. V Příloze III lze vidět snímek, jak vypadá grafická aplikace v režimu „online“.

Druhou součástí aplikace je M-file, který obsahuje tzv. callback funkce, do kterých se programuje funkce jednotlivých komponent.

Při inicializaci M-file se provádí nulování hodnot, nastavení viditelnosti komponent, nastavení zobrazení grafu, jako jsou osy, popisy a zobrazení mřížky.

Pro zpracování ip adresy a nastavení komunikace je použita následující funkce, která se volá se stiskem tlačítka. Principem je složení ip adresy a její kontroly ze zadaných hodnot. Pokud je zadaná ip adresa v pořádku, proběhne nastavení komunikace.

```
function nastaveni_Callback(hObject, eventdata, handles)
% hObject    handle to nastaveni (see GCBO)
% eventdata  reserved - to be defined in w future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global t port ip1p ip2p ip3p ip4p
adresa = [ip1p '.' ip2p '.' ip3p '.' ip4p];
if (port == 0)
   port = 80;
   set(handles.Port,'String',port);
end
if (isempty(ip1p) | isempty(ip2p) | isempty(ip3p) | isempty(ip4p))
    errordlg('Není zadaná IP adresa!','Bad Input','modal')
    uicontrol(hObject)
    return
else
    t = tcpip(adresa,port);
    set(t,'NetworkRole', 'client');
    set(t,'timeout', 0.5);
    set(t,'InputBufferSize', 30);
    t.terminator = 'LF';
    t.ReadAsyncMode = 'continuous';
end
```

Zdrojový kód 8.1: Matlab GUI - zpracování ip adresy

Odeslání matic do modelu je rozděleno do několika kroků. Nejprve proběhne kontrola, jestli byl vyplněný stavový popis. Pokud ano, z vyplněných polí se složí matice.

```
if (isempty(rad) || (rad == 1))
    errordlg('Nejprve zadejte data k odeslání!','Bad Input','modal')
    uicontrol(hObject)
    set(handles.text75,'Visible','off');
    return
else
    set(handles.text75,'Visible','on');
    odeslano_M = 0; odeslano_N = 0; odeslano_C = 0; odeslano_D = 0;
    odeslano_R = 0; odeslano_L = 0;
    switch rad
        case 2
            M = [m11 m12;m21 m22]; N = [n11;n21]; C = [c11 c12];
            D = [d11]; R = [r11 r12 r13]; L = [l11;l21];
        case 3
            M = [m11 m12 m13;m21 m22 m23;m31 m32 m33];
            N = [n11;n21;n31]; C = [c11 c12 c13]; D = [d11];
            R = [r11 r12 r13 r14];
            L = [l11;l21;l31];
        case 4
             M = [m11 m12 m13 m14;m21 m22 m23 m24;m31 m32 m33 m34;m41 m42
             m43 m44];
            N = [n11;n21;n31;n41]; C = [c11 c12 c13 c14]; D = [d11];
            R = [r11 r12 r13 r14 r15];
            L = [l11;l21;l31;l41];
    end
```

Zdrojový kód 8.2: Matlab GUI - kontrola vyplnění stavového popisu a složení matic

Poté, co se vytvořili matice, dojde postupně k odesílání dat. To probíhá tak, že se prvek matice, nebo číslo převede na řetězec podle pevně daného vzoru (viz Tabulka 8.1) a postupně se odesílají do modelu.

Hodnota	Formát odesílaného řetězce
Žádaná hodnota	W±00.0;
Řád systému	Q0;
Prvek matice M, N	MXY±0.0000;
Prvek matice C	CXY±00.0000;
Prvek matice D	D11±0.0000;
Prvek matice R, L	RXY±000.0000;

Tabulka 8.1: Formáty odesílaných řetězců

Pro složení řetězců jsem použil pro každou matici funkci zvlášť. Jako příklad je uvedené zpracování matice R na řetězce (viz Příloha IV). Princip je takový, že se načte prvek matice a podle jeho hodnoty se převede na řetězec a doplní se o zbylé znaky, aby splňoval předepsaný vzor. Poté dojde k jeho odeslání. Pomocí příkazu *query* se získá odezva modelu, která by měla stejná jako odeslaný řetězec. Pokud se shodují, přejde se na poslání následujícího prvku matice. Takto se to provede pro všechny prvky matice a po úspěšném odeslání celé matice se přejde na další matici.

Pokud se v pořádku odešlou všechny hodnoty, je možné stavový popis uložit do souboru. K tomu slouží následující funkce, která se volá po stisknutí tlačítka.

```
function save_Callback(hObject, eventdata, handles)
% hObject    handle to save (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%ulozeni popisu do souboru
global odeslano_M odeslano_N odeslano_C odeslano_D odeslano_R odeslano_L
global m11 m12 m13 m14 m21 m22 m23 m24 m31 m32 m33 m34 m41 m42 m43 m44
global n11 n21 n31 n41 c11 c12 c13 c14 d11 r11 r12 r13 r14 r15
global l11 l21 l31 l41 rad
%ukládá se pouze úspěšně odeslané nastaveni
if
((odeslano_M==1)&(odeslano_N==1)&(odeslano_C==1)&(odeslano_D==1)&(odeslano
_R==1)&(odeslano_L==1))
    matice_M = [m11 m12 m13 m14;m21 m22 m23 m24;m31 m32 m33 m34;m41 m42
                m43 m44];
    matice_N = [n11;n21;n31;n41];
    matice_C = [c11 c12 c13 c14];
    matice_D = [d11];
    matice_R = [r11 r12 r13 r14 r15];
    matice_L = [l11;l21;l31;l41];
    rad_soustavy = rad;
    filename = datestr(now,'mmmm_dd_yyyy_HH_MM_SS');
    filename = strcat('Nastaveni_',filename,'.mat');
save(filename,'matice_M','matice_N','matice_C','matice_D','matice_R','mati
ce_L','rad_soustavy');
else
    errordlg('Nejprve odešlete nastavení do procesoru!','Bad
    Input','modal')
    uicontrol(hObject)
    return
end
```

Zdrojový kód 8.3: Matlab GUI - uložení aktuálního stavového popisu do souboru

Nejprve dojde ke kontrole, jestli byla všechna data úspěšně odeslána. Pokud ano, vytvoří se soubor, jehož jméno je ve formátu *mmmm_dd_yyyy_HH_MM_SS*, které symbolizuje aktuální čas a datum. Příkazem *save* jsou specifikované proměnné, které se ukládají.

Odesláním dat se „uvolnila“ možnost spuštění programu tlačítkem *„Start“*. Tím se spouští obsáhlá funkce (viz Příloha IV), která obsahuje tři hlavní části.

První je příjem dat a jejich zpracování. Zpracování probíhá tak, že se přijatý řetězec rozdělí na čtyři a čtyři znaky, které reprezentují hexadecimální číslo. Tyto čísla se převedou na decimální číslo a přepočítají se z celočíselného na desetinné číslo podle postupu, jaký byl zvolen při programování procesoru (viz kapitola 7). Takto přepočítaná čísla se poté zobrazují na obrazovce v poli *„Aktuální hodnoty“*.

Další částí je ukládání hodnot do proměnné, která slouží jako databáze pro vykreslování hodnot v grafu. Kromě měřených hodnot se zde ukládají informace o aktuálním čase, kdy byl vzorek naměřen, a doba, po kterou program běží. Tento čas tvoří časovou osu při vykreslování grafu. Jeho výpočet se provádí odečítáním aktuálního času od času spuštění.

Poslední částí je vykreslení hodnot do grafu. Je zde zobrazen akční zásah, aktuální úhel natočení a žádaná hodnota.

Po stisku tlačítka „Stop“ pro ukončení regulace, proběhne odeslání příkazu pro zastavení programu a následně dojde k uložení měřených dat do .mat souboru.

```
set(hObject,'String','Start');
fopen(t);
fprintf(t,'P');
fclose(t);
%uložení souboru s daty
filename = strcat(filename,'.mat'); %vytvoření názvu souboru
save(filename,'hodnoty'); %uložení dat do aktuální složky
clear global hodnoty; %smazání obsahu proměnné s daty
```

Zdrojový kód 8.4: Matlab GUI - obsah funkce tlačítka stop

V rámci většiny funkcí jsou nastavená různá ošetření, jako je například zadávání špatných hodnot, kontrola odeslání, rozsahů atd. Pokud se nějaký z těchto stavů nastane, je volána funkce pro zobrazení dialogového okna s popisem chyby. Příklad použití:

```
function IP1_Callback(hObject, eventdata, handles)
% hObject    handle to IP1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of IP1 as text
%        str2double(get(hObject,'String')) returns contents of IP1 as a
double
global ip1 ip1p
ip1 = str2double(get(hObject,'string'));
if (isnan(ip1) | (ip1<0) | (ip1>255))
    errordlg('Špatně zadaná IP1 adresa!','Bad Input','modal')
    uicontrol(hObject)
    ip1p = '';
    return
else
    ip1p = sprintf('%d',ip1);
end
```

Zdrojový kód 8.5: Matlab GUI - příklad ošetření špatně zadaných hodnot

9 Měření

Při měření jsem pracoval s identifikovanou přenosovou funkcí dle rovnice (4.1), navrženou maticí estimátoru L (rovnice (4.2)) a s vypočítanou maticí regulátoru R (rovnicev(4.3)). Podle kapitoly 1.3 jsem nejprve převedl přenosovou funkci (4.1) do stavového popisu. Poté jsem tento popis převedl ze spojité do diskrétní časové oblasti. Výsledkem byly následující matice:

$$M = \begin{bmatrix} 1 & 0{,}025 & 0{,}0012 & 0{,}0001 \\ 0 & 0{,}9998 & 0{,}0992 & 0{,}0095 \\ 0 & -0{,}0067 & 0{,}9755 & 0{,}1843 \\ 0 & -0{,}0648 & -0{,}2395 & 0{,}8387 \end{bmatrix} \tag{9.1}$$

$$N = \begin{bmatrix} 0 \\ 0{,}0051 \\ 0{,}1517 \\ 1{,}4742 \end{bmatrix} \tag{9.2}$$

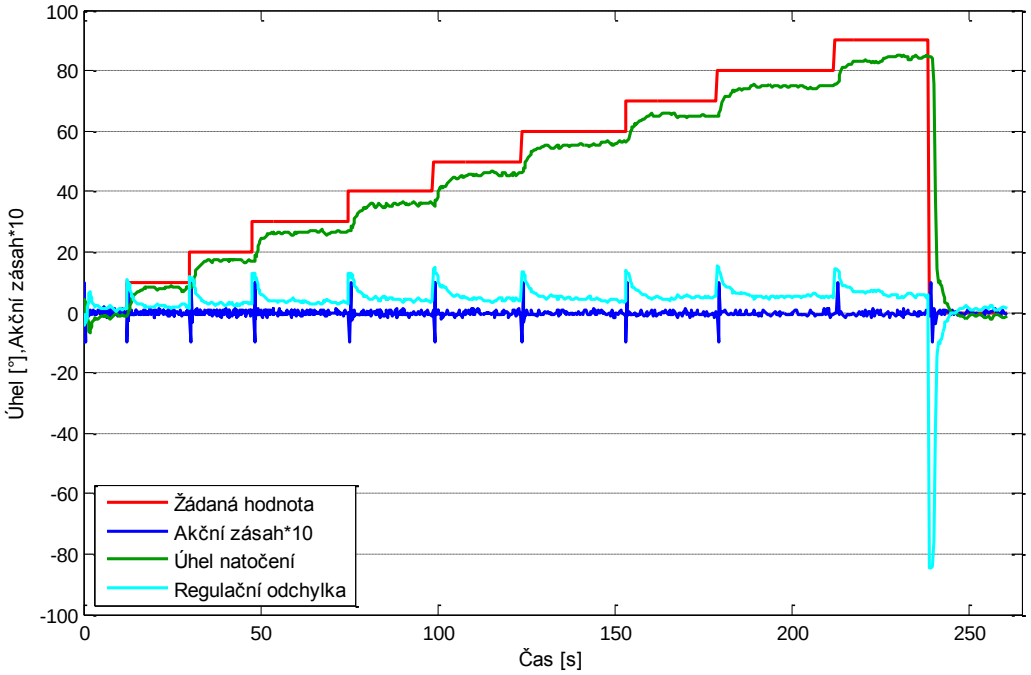
$$C = [-41{,}8954 \quad 0 \quad 0 \quad 0] \tag{9.3}$$

$$D = [0] \tag{9.4}$$

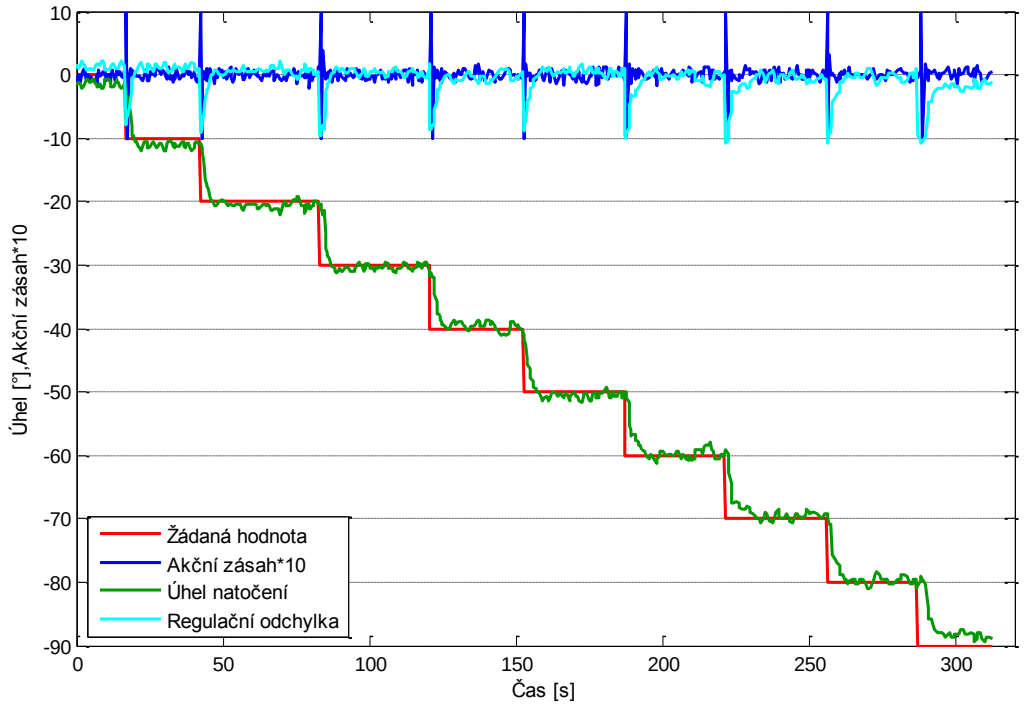
$$L = \begin{bmatrix} -0{,}091 \\ -3{,}754 \\ -23{,}3364 \\ -24{,}3634 \end{bmatrix} \tag{9.5}$$

$$R = [102{,}5532 \quad 11{,}1740 \quad 2{,}6741 \quad 0{,}7543 \quad -0{,}3158] \tag{9.6}$$

Po nahrání těchto matic do modelu, jsem provedl měření, ve kterém jsem proměřil postupně celý rozsah (od -90° do 90°) s krokem 10°.



Graf 9.1: Proměření rozsahu otáčení modelu v kladném směru



Graf 9.2: Proměření rozsahu otáčení modelu v záporném směru

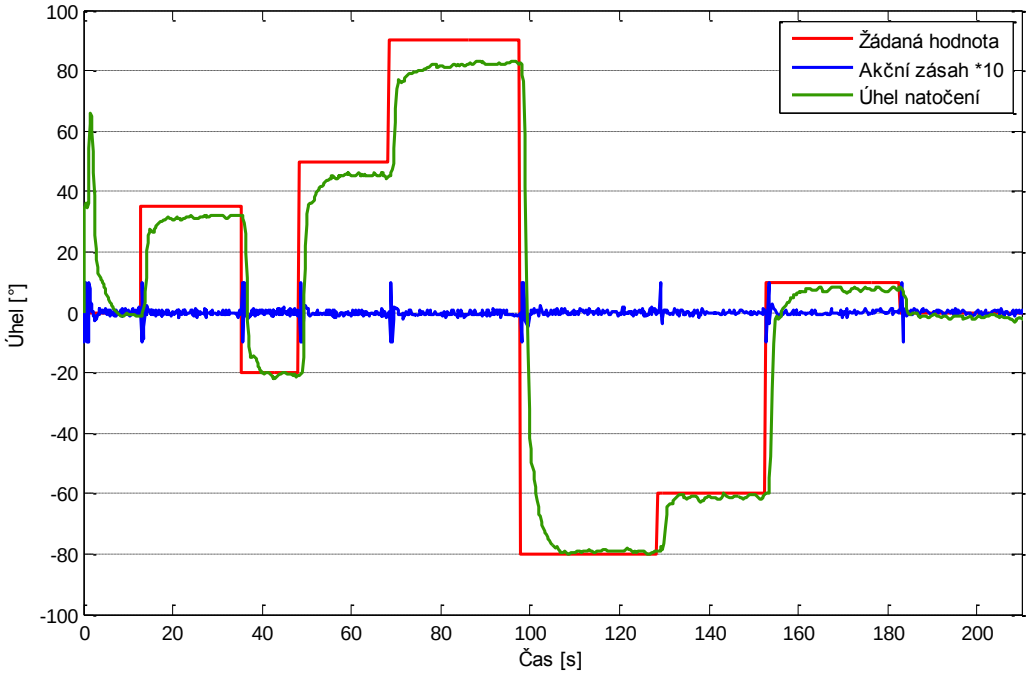
Jak lze vidět z Grafů 9.1 a 9.2, při postupném zvyšování úhlu natočení v kladném rozsahu (0 - 90°) rostla regulační odchylka, zatímco při postupném snižování úhlu se regulační odchylka pohybovala kolem hodnoty 2°.

Tento jev mohl mít jednak mechanický původ, a to takový, že v pohybu jedním směrem vznikalo větší tření než ve směru opačném.

Druhá možnost vzniku tohoto jevu je taková, že se při návrhu uvažuje, že soustava obsahuje integrační složku, kterou opravdu při otáčení obsahuje, ale v momentě, kdy je v klidu, již tuto složku neobsahuje. Proto, nízké akční zásahy nemají dostatečnou sílu a nezpůsobí otáčení soustavy a tím nedojde k doregulování regulační odchylky.

Takto vzniklou nelinearitu jsem alespoň mírně eliminoval tím způsobem, že jsem nízké akční zásahy v rozmezí od -0,05 do 0,05 zdvojnásobil, aby se zvýšila citlivost.

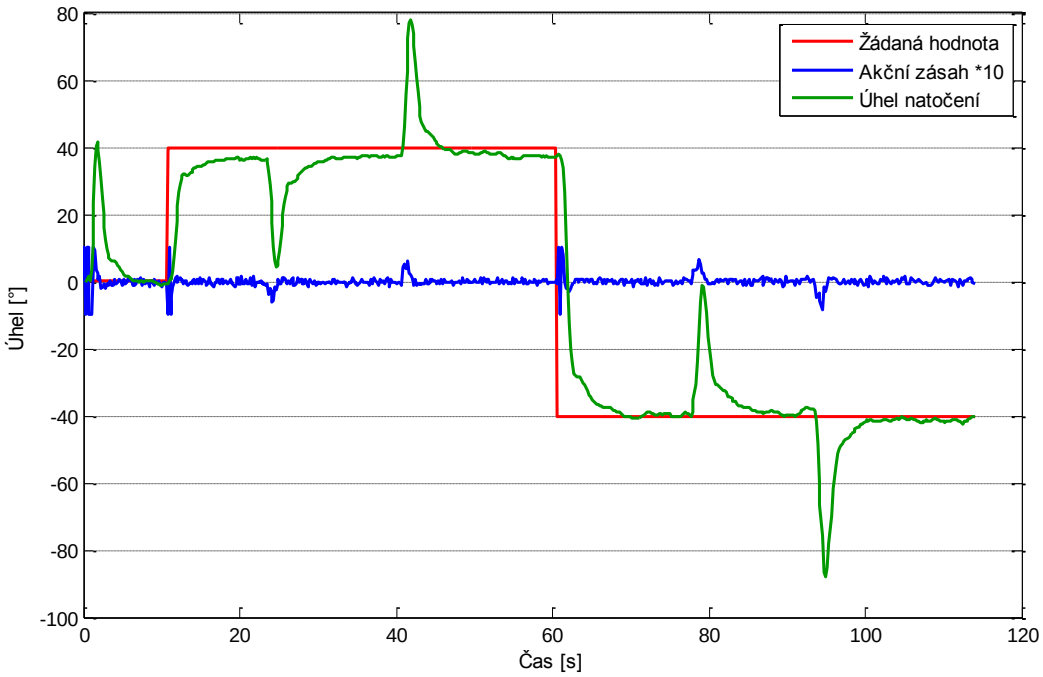
Dále jsem proměřil dynamické vlastnosti soustavy, a to tak, že jsem provedl několik skokových změn žádané hodnoty v celém rozsahu.



Graf 9.3: Změřená dynamická charakteristika

Z Grafu 9.3 lze vidět, že i při měření při různých změnách žádané hodnoty, přetrvávala regulační odchylka v kladném směru otáčení. Je zde také vidět způsob, jakým regulátor funguje, a to ve smyslu, že oproti simulaci zde není žádný překmit, naopak se provede první výrazná změna a poté dochází k postupnému doregulování na žádanou hodnotu. Také si lze všimnout, že změny akční nejsou nijak veliké. To se shoduje s předpokladem ze simulace, kde také byly malé akční zásahy.

Jako poslední jsem provedl měření s poruchovou veličinou (Graf 9.4). Postup měření byl takový, že jsem provedl vychýlení v obou směrech otáčení a monitoroval jsem reakci modelu. Tento způsob jsem provedl jak v kladném, tak i v záporném úhlu natočení.



Graf 9.4: Měření odezvy na poruchovou veličinu

Z Grafu 9.4 lze vidět, že po vychýlení ramene modelu, byla odchylka rychle a relativně přesně zregulována. Stejně jako v průběhu celého měření ani regulování poruchové veličiny nedocházelo k výrazným překmitům nebo regulační odchylce po ustálení.

Závěr

Stavové řízení je oproti jiným metodám popisu výhodnější. Mezi jeho výhody patří například možnost popsat systém s více vstupy / výstupy, nebo systémy se složitou vnitřní strukturou. Hlavními výhodami jsou však jednoduché návrhy estimátoru a stavového regulátoru.

Vývojové prostředí GUIDE v programu Matlab má mé osobní hodnocení vysoké. To především proto, že je to jednoduchý, přehledný a intuitivní nástroj pro tvorbu grafických rozhraní s rozsáhlou webovou podporou. Práci s tímto nástrojem zvládne každý uživatel znalý programování v programu Matlab, protože funkce se vytvářejí do M-file a používají se standardní funkce.

Co se týče jednotlivých bodů zadání, tak ty se mi podařilo splnit všechny. Jejich stručné řešení a výsledky jsou popsány v následujících odstavcích.

Prvním cílem bylo identifikovat fyzikální model. Nejprve bylo potřeba určit matematický model. Zde jsem vycházel z rovnice pro výpočet momentu síly, protože tah motoru je síla a otáčení soustavy také vytváří moment síly. Dále bylo potřeba zahrnout do tohoto modelu dynamiku stejnosměrného motoru. Tu jsem do výpočtu zavedl tím, že je známá obecná přenosová funkce pro přenos z napětí na otáčky motoru. Výsledný matematický model (rovnice (3.8)) vyšel dle očekávání čtvrtého řádu.

V dalším kroku se tento matematický model aproximoval spojitou přenosovou funkcí. Nejprve jsem naměřil dynamickou charakteristiku chování modelu, ze které jsem při identifikaci vycházel. Pro identifikaci jsem vytvořil script v programu Matlab, který využívá funkci *fminsearch*. Vstupem byla naměřená data a známá struktura matematického modelu. Po několika optimalizacích vyšla výsledná astatická přenosová funkce čtvrtého řádu.

Pro tuto identifikovanou přenosovou funkci jsem navrhl estimátor a stavový regulátor. Estimátor jsem zvolil deterministický Luenbergerův estimátor, neboli diskrétní estimátor úplného řádu, který umožňuje estimovat nejen vnitřní stavy, ale i výstupní veličinu. Výstupem návrhu byla matice estimátoru L (rovnice (4.2)). Stavový regulátor (rovnice (4.3)) jsem navrhl pomocí Riccatiho rovnice a to tak, že jsem uvažoval i astatickou složku modelu a tím výsledný stavový regulátor odpovídá svým chováním PI regulátoru.

Aby bylo možné stavový regulátor převést do programu procesoru, bylo zapotřebí převést rovnice pro estimaci vnitřních stavů, výstupu a výpočtu akčního zásahu z matematického vyjádření do programovaného kódu. Takto navržený výpočet jsem použil

nejprve v programu Simulink, abych vyzkoušel jeho funkčnost v simulaci. Výsledné průběhy vyšly téměř identické, a lze tedy říci, že převod skončil úspěchem.

Poté co byl hotový návrh výpočtu akčního zásahu, jsem v programu Atmel Studio vytvořil program, který načítá data stavového popisu a vypočítává akční zásah (podrobnější popis v kapitole 7).

Dalším krokem bylo vytvoření grafické aplikace v programu Matlab. K tomu slouží vývojové prostředí GUIDE. Zde jsem vytvořil aplikaci, která umožňuje uživateli zadat parametry stavové popisu a matici regulátoru, nastavit ip adresu pro komunikaci PC s modelem, ovládat žádanou hodnotu a sledovat v grafu aktuální hodnoty. Mezi další možnosti aplikace patří ukládání naměřených hodnot a aktuálního stavového popisu do souboru (podrobnější popis v kapitole 8).

Posledním bodem mé diplomové práce bylo závěrečné ověření funkce regulace. Z grafů v kapitole 9 je vidět, že regulátor funguje relativně spolehlivě, s ohledem na to, že se jedná o astatickou a nestabilní soustavu. Z měření je vidět, že oproti simulacím je ustálení výstupní veličiny sice pomalejší, ale za to nedochází k žádným překmitům. Horších vlastností dosahoval v kladném směru otáčení, kde jeho regulační odchylka s rostoucím úhlem rostla, což je pravděpodobně způsobené větším mechanickým třením v tomto směru otáčení. V opačném směru otáčení tento jev nenastal, a regulační odchylka se pohybovala přibližně v rozmezí 2°, která byla způsobena tím, že regulátor vypočítával příliš malé akční zásahy, které nebyly dostatečně veliké natolik, aby byly schopny překonat sílu potřebnou pro otočení ramene modelu. Toto jsem částečně eliminoval tím, že jsem nízké akční zásahy v rozmezí od -0,05 do 0,05 zdvojnásobil a zmenšil tím regulační odchylku.

Použitá literatura

[1] HEROUT, Pavel. *Učebnice jazyka C*. 6. vyd. České Budějovice: KOPP, 2009. ISBN 978-80-7232-383-8.

[2] Stránky Atmel. [online]. [cit. 2015-05-01]. Dostupné z: http://www.atmel.com/

[3] Stránky Matlab. [online]. [cit. 2015-05-01]. Dostupné z: https://www.mathworks.com/

[4] BALÁTĚ, Jaroslav. *Automatické řízení*. BEN-Technická literatura, 2004. ISBN 978-80-7300-1.

[5] *Referenční příručka*. [online]. 2015 [cit. 2015-05-01]. ASIX. Dostupné z: http://www.asix.cz/download/programmers/presto_cz.pdf

[6] MELICHAR, Jiří. *Lineární systémy 2: učební text*. [online]. 2011 [cit. 2015-05-01]. Západočeská univerzita v Plzni. Dostupné z: http://www.kky.zcu.cz/uploads/courses/ls2/LS2_Ucebni_texty_2011.pdf

[7] MELICHAR, Jiří. *Lineární systémy 1: učební text*. [online]. 2010 [cit. 2015-05-01]. Západočeská univerzita v Plzni. Dostupné z: http://www.kky.zcu.cz/uploads/courses/ls1/LS1-Ucebni-texty-2010.pdf

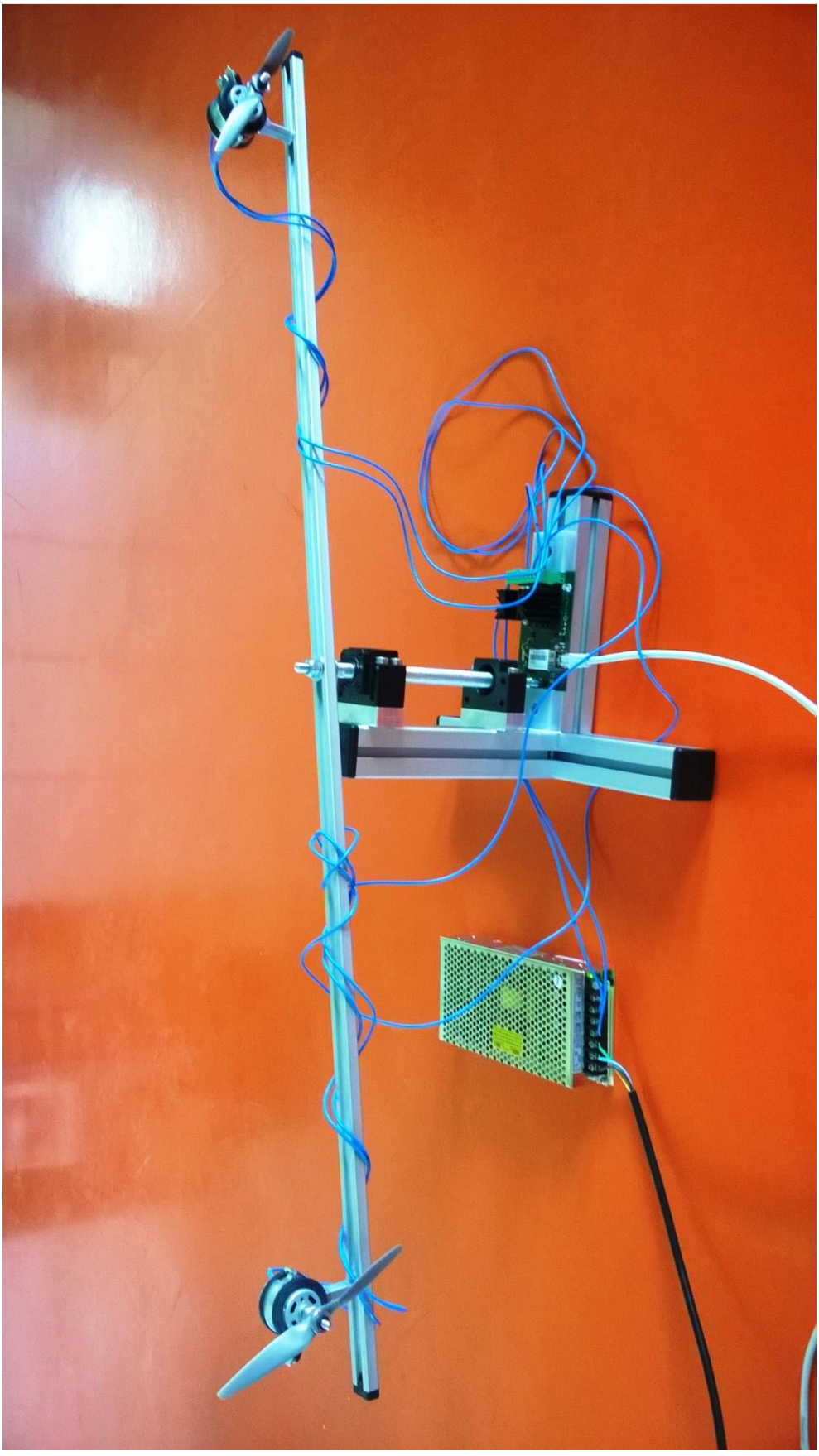
[8] MODRLÁK, Osvald a Lukáš HUBKA. *Automatické řízení: učební text*. 1. vyd. Liberec: Technická univerzita v Liberci, 2012. ISBN 978-80-7372-850-2.

[9] DOSTÁL, Petr a Radek Matušů. *Stavová a algebraická teorie řízení*. 1. vyd. Zlín: Univerzita Tomáše Bati ve Zlíně, 2010. ISBN 978-80-7318-991-4.

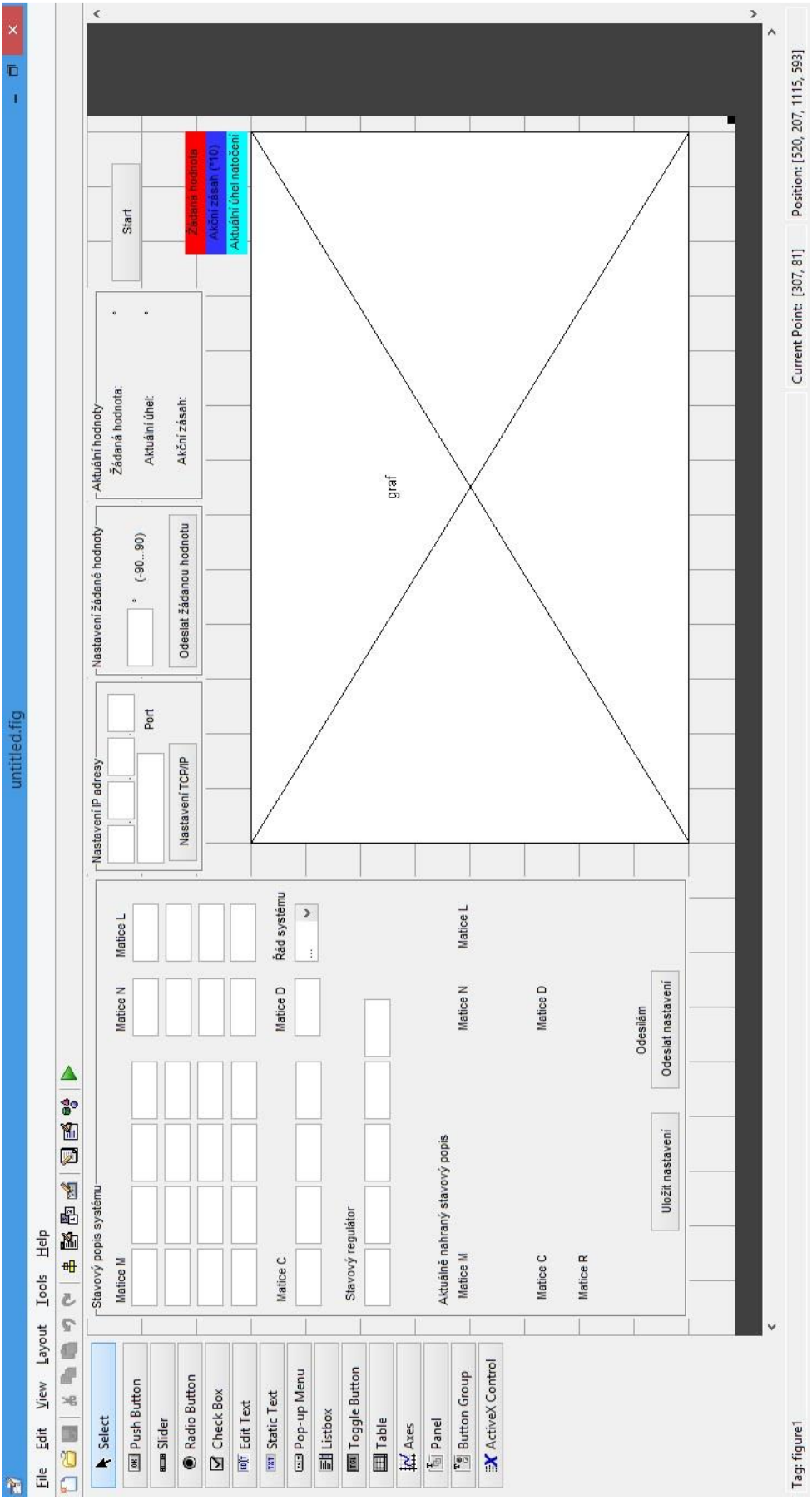
[10] OŽANA, Štěpán. *Navrhování a realizace regulátorů*. 1. vyd. Ostrava: Vysoká škola báňská - Technická univerzita, 2012. ISBN 978-80-248-2605-9.

[11] NELDER, J. A.; MEAD, R. A.. Simplex Method for Function Minimization. *Computer Journal*, vol 7, 1965, p. 308-313.

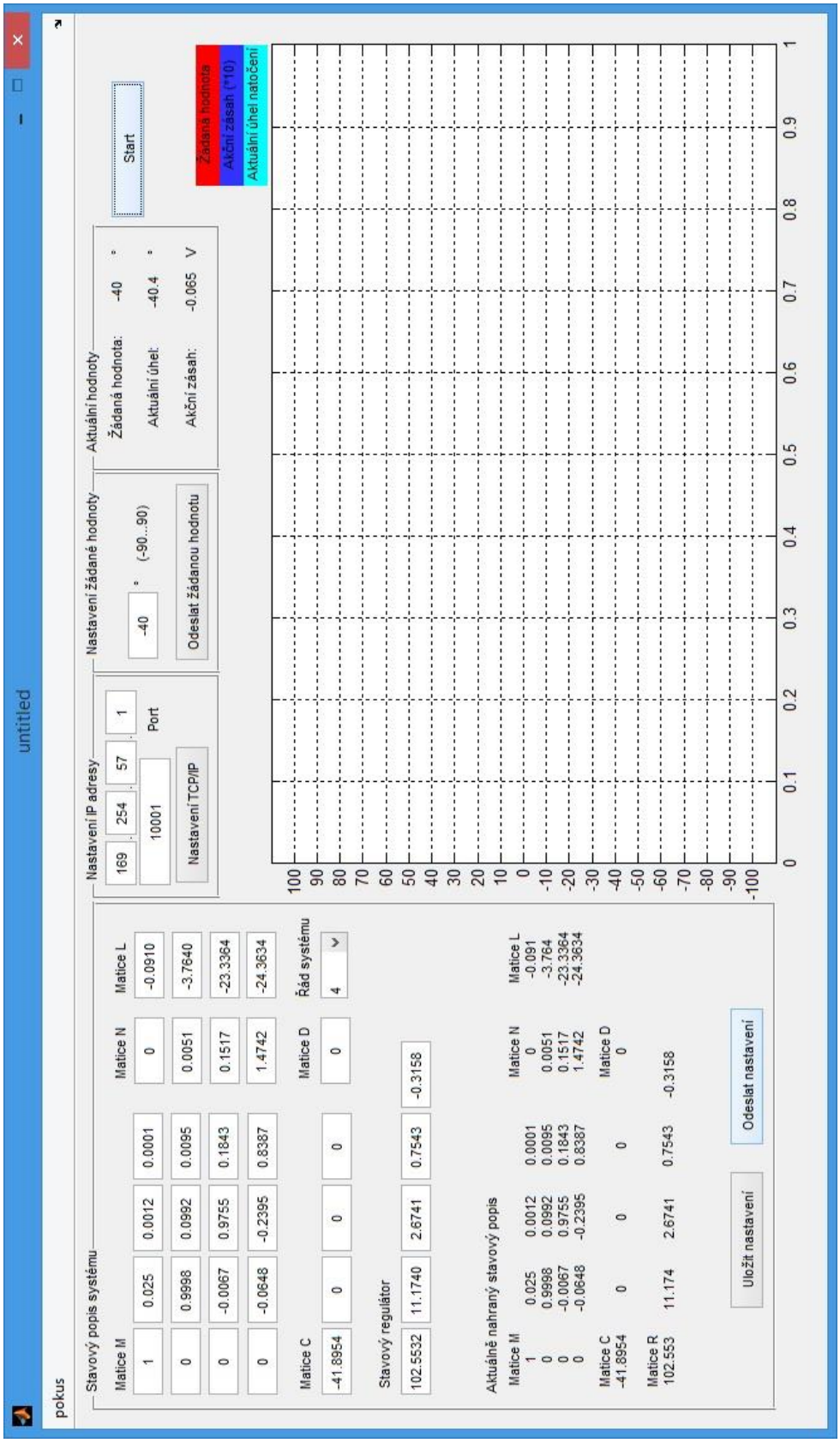
Příloha I – Fotografie modelu



Příloha II – Snímek grafické aplikace v GUIDE



Příloha III – Snímek grafické aplikace v chodu



Příloha IV – Zdrojový kód převodu matice na řetězce

```
Rod = '';
    [Rx Ry] = size(R);
    for(j = 1:Ry)
        Rod = '';
        pomj = num2str(j);
        Rod = ['R' '1' pomj];
        if (R(1,j)>=0 && R(1,j)<10)
            pom = sprintf('%.4f',R(1,j));
            pom = ['00' pom];
        elseif (R(1,j)>=0 && R(1,j)<100)
            pom = sprintf('%.4f',R(1,j));
            pom = ['0' pom];
        elseif (R(1,j)>=0 && R(1,j)<1000)
            pom = sprintf('%.4f',R(1,j));
        elseif (R(1,j)<0 && R(1,j)>-10)
            pom = sprintf('%.4f',R(1,j)) ;
            pom = ['-00' pom(2:end)];
        elseif (R(1,j)<0 && R(1,j)>-100)
            pom = sprintf('%.4f',R(1,j)) ;
            pom = ['-0' pom(2:end)];
        elseif (R(1,j)<0 && R(1,j)>-1000)
            pom = sprintf('%.4f',R(1,j)) ;
        end
        if R(1,j)>=0
            Rod = [Rod '+' pom];
        else
            Rod = [Rod pom];
        end
        Rod = [Rod ';'];
        fopen(t);
        vystup = query(t,Rod,'%s','%s')
        fclose(t);
        if(strcmp(Rod,vystup)==0)
             errordlg('Špatně odesláno (matice R)! Odešlete dat znovu.','Bad
             Input','modal')
            uicontrol(hObject)
            set(handles.text75,'Visible','off');
            return
        end
    end
       odeslano_R = 1;
```

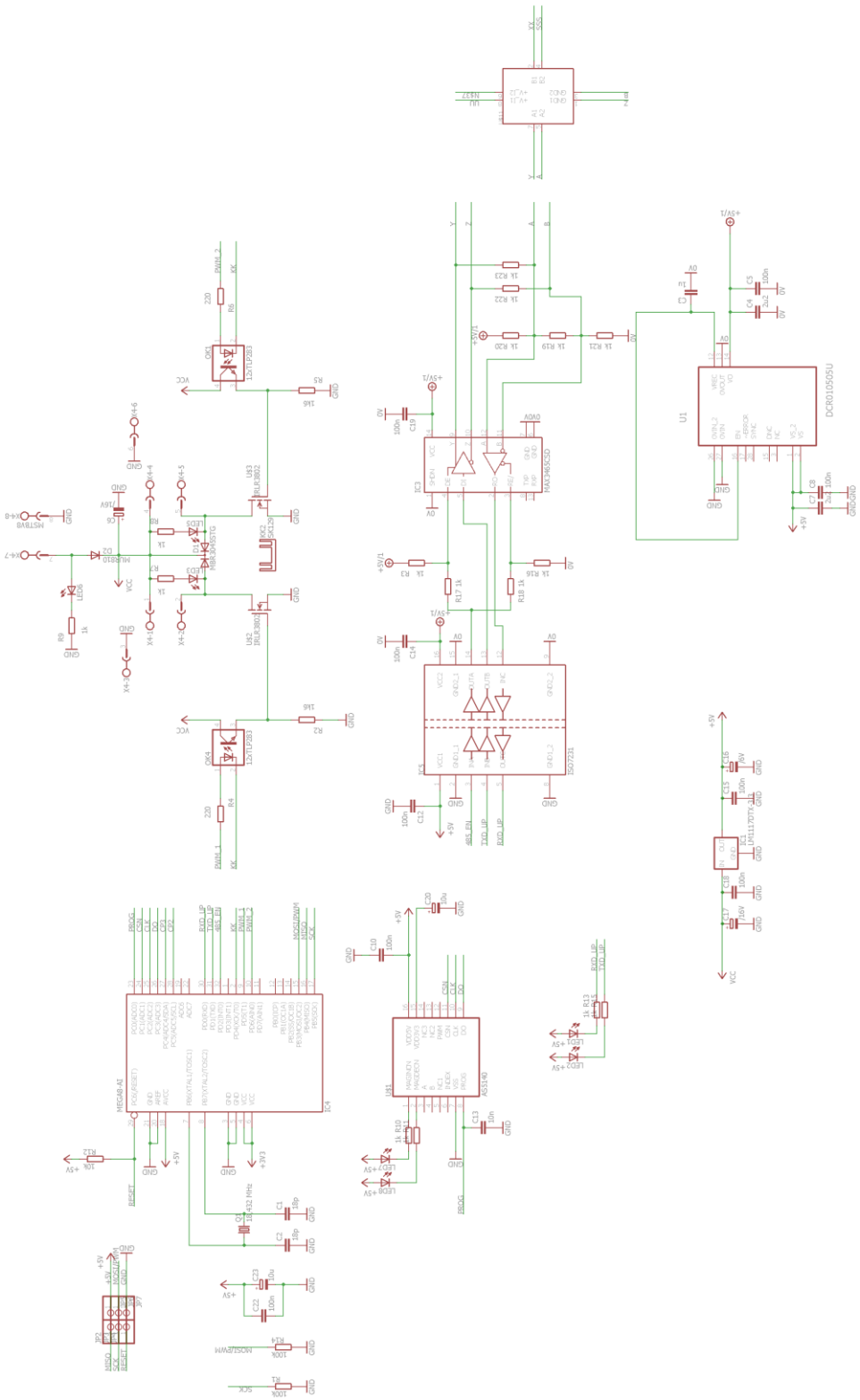
Příloha V – Zdrojový kód tlačítka Start

```
function togglebutton2_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of togglebutton2
global odeslano_M odeslano_N odeslano_C odeslano_D odeslano_R odeslano_L odeslano_w
global t w filename hodnoty
cla; %vymazání grafu před spuštěním
if get(hObject,'Value') %stisk start
    if ((odeslano_M==1)&(odeslano_N==1)&(odeslano_C==1)&(odeslano_D==1)&
       (odeslano_R==1)&(odeslano_L==1)&(odeslano_w==1))
        set(hObject,'String','Stop');
        fopen(t);
        fprintf(t,'T');
        fclose(t);
        j = 2;
        start_time = datestr(now,'mmmm dd, yyyy HH:MM:SS.FFF');
        filename = datestr(now,'mmmm_dd_yyyy_HH_MM_SS');
        %-------------------------------------------------------------------------
        start_hodiny = [start_time(end-11) start_time(end-10)]; %načtení hodin
        start_hodiny = str2double(start_hodiny); %převod na číslo
        start_hodiny = start_hodiny * 60 * 60; % převod na vteřiny
        %-------------------------------------------------------------------------
        start_minuty = [start_time(end-8) start_time(end-7)]; %načtení minut
        start_minuty = str2double(start_minuty); %převod na číslo
        start_minuty = start_minuty * 60; %převod na vteřiny
        %-------------------------------------------------------------------------
        start_vteriny = [start_time(end-5) start_time(end-4) start_time(end-3)
        start_time(end-2) start_time(end-1) start_time(end)]; %načtení vteřin
        start_vteriny = str2double(start_vteriny); %převod na číslo
        %-------------------------------------------------------------------------
        start_cas = start_vteriny + start_minuty + start_hodiny; %počáteční čas ve
        vteřinách
        %-------------------------------------------------------------------------
        while get(hObject,'Value')
            fopen(t);
            slovo = fscanf(t,'%c',20)
            fclose(t);
            pom = isempty(slovo);
            if (pom == 0) %pokud se něco načetlo, pokračuje se
                %buffer v bufferu - vybrání potřebných dat
                for (p = 1:20) %načtení 8mi znaku za vodicím znakem !
                    if slovo(p) == '!'
                        vystup = slovo(p+1:p+8);
                        break
                    end
                end
                akc_zasah = '';
                uhel = '';
                akc_zasah = vystup(1:4);
                uhel = vystup(5:8);
                %převod z HEX na dekadické číslo
                %přepočet akčního zásahu
                    akc_zasah = hex2dec(akc_zasah);
                    akc_zasah = akc_zasah / 32767.5;
                    akc_zasah = akc_zasah - 1;
                    akc_zasah = roundn(akc_zasah,-3);
                    set(handles.akcni_zasah,'String',akc_zasah);
                    %přepočet úhlu
                    uhel = hex2dec(uhel);
                    uhel = uhel / 327.675;
                    uhel = uhel - 90;
                    uhel = roundn(uhel,-1);
```

```
                    set(handles.aktualni_uhel,'String',uhel);
                    %ukládání do proměnné data, která se ukládá a zároveň se z ni
                    Vykresluji data
                    hodnoty(j,1) = w;
                    hodnoty(j,2) = akc_zasah;
                    hodnoty(j,3) = uhel;
                    pom_cas = datestr(now,'mmmm dd, yyyy HH:MM:SS.FFF');
                    %-----------------------------------------------------------------
                    aktualni_hodiny = [pom_cas(end-11) pom_cas(end-10)]; %načtení
                    hodin
                    aktualni_hodiny = str2double(aktualni_hodiny); %převod na číslo
                    hodnoty(j,4) = aktualni_hodiny;
                    aktualni_hodiny = aktualni_hodiny * 60 * 60; %převod na vteřiny
                    %-----------------------------------------------------------------
                    aktualni_minuty = [pom_cas(end-8) pom_cas(end-7)]; %načteni
                    minut
                    aktualni_minuty = str2double(aktualni_minuty); %převod na číslo
                    hodnoty(j,5) = aktualni_minuty;
                    aktualni_minuty = aktualni_minuty * 60; %převod na vteřiny
                    %-----------------------------------------------------------------
                    aktualni_vteriny = [pom_cas(end-5) pom_cas(end-4) pom_cas(end-
                    3) pom_cas(end-2) pom_cas(end-1) pom_cas(end)]; %načtení vteřin
                    aktualni_vteriny = str2double(aktualni_vteriny); %převod na
                    číslo
                    hodnoty(j,6) = aktualni_vteriny;
                    %-----------------------------------------------------------------
                    aktualni_cas = aktualni_vteriny + aktualni_minuty +
                    aktualni_hodiny; %aktuální čas ve vteřinách
                    %vykresleni do grafu
                    cas = aktualni_cas - start_cas;
                    hodnoty(j,7) = cas; %čas běhu ve vteřinách
                    plot(hodnoty(:,7),hodnoty(:,1),'r'); %žádaná hodnota
                    hold on;
                    plot(hodnoty(:,7),10*hodnoty(:,2),'b'); %akční zásah, do grafu
                    násoben *10, aby byl vidět
                    hold on;
                    plot(hodnoty(:,7),hodnoty(:,3),'c'); %aktuální úhel
                    %nastaveni limit, popisu os
                    grid on;
                    set(gca,'YLim',[-110 110]);
                    set(gca,'YTick',-100:10:100);
                    set(gca,'YTickLabel',{'-100';'-90';'-80';'-70';'-60';'-50';'-
                         40';'-30';'-20';'-10';'0';'10';'20';'30';
                        '40';'50';'60';'70'; '80';'90';'100'});
                    %posun na další řádek, pro zapsání do souboru
                    j = j + 1;
            else
                errordlg('Žádná data!','Bad Input','modal')
                uicontrol(hObject)
                return
            end
            pause(0.001);
        end
    else
        errordlg('Špatně zadaná nebo odeslaná data!','Bad Input','modal')
        uicontrol(hObject)
        set(hObject,'Value',0); %nastaveni tlačítka do původní polohy
        return
    end
else %stisk stop
end
```

Příloha VI – Kompletní schéma zapojení desky plošného spoje



Příloha VII – obsah přiloženého DVD

- Text diplomové práce
  - Jan Beran – Diplomová práce.pdf
  - Jan Beran – Diplomová práce.docx
  - Jan Beran – Zadání diplomové práce
- Zdrojové kódy
  - pro mikroprocesor (programováno v programu Atmel Studio)
  - aplikace pro PC (programováno v programu Matlab)