Univerzita Pardubice
Fakulta elektrotechniky a informatiky

APLIKACE PREDIKTIVNÍHO REGULÁTORU V JEDNODUCHÉM ŘÍDICÍM SYSTÉMU

Jiří Vinduška

Diplomová práce

2019

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2018/2019**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Jiří Vinduška**
Osobní číslo:	**I17199**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Řízení procesů**
Název tématu:	**Aplikace prediktivního regulátoru v jednoduchém řídicím systému**
Zadávající katedra:	**Katedra řízení procesů**

Zásady pro vypracování:

Cíl: Navrhnou a aplikovat prediktivní regulátor v jednoduchém řídicím systému, kterým může být jednočipový mikropočítač nebo PLC.
Obsah teoretické části: Prediktivní regulace - kritérium, model řízeného systému, způsob řešení optimalizační úlohy. Modelování a identifikace - matematicko-fyzikální analýza, experimentální identifikace.
Obsah implementační části: Návrh vhodného algoritmu prediktivního regulátoru, simulační ověření navrženého řešení, přenesení do cílové platformy, ověření při řízení reálné soustavy, diskuze výsledků.

Rozsah grafických prací:

Rozsah pracovní zprávy: **80**

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

Seznam odborné literatury:

CAMACHO, E. F.; BORDONS, C. Model Predictive Control (Second Edition). London : Springer-Verlag London Limited, 2007. 405 s.
ROSSITER, J. A. Model-based Predictive Control - A Practical Approach. Boca Raton (Florida) : CRC Press, 2004. 318 s.
MIKLEŠ, J.; FIKAR, M. Modelovanie, identifikácia a riadenie procesov 2. Identifikácia a optimálne riadenie. Bratislava : STU v Bratislave, 2004. 267 s.

Vedoucí diplomové práce: **Ing. Daniel Honc, Ph.D.**
Katedra řízení procesů

Datum zadání diplomové práce: **1. listopadu 2018**
Termín odevzdání diplomové práce: **17. května 2019**

Univerzita Pardubice Česká republika

L.S.

Ing. Zdeněk Němec, Ph.D.
děkan

Ing. Daniel Honc, Ph.D.
vedoucí katedry

V Pardubicích dne 6. listopadu 2018

Prohlášení

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 17. 05. 2019

Bc. Jiří Vinduška

Poděkování

Chtěl bych poděkovat panu Ing. Danielu Honcovi, Ph.D. za odborné vedení, za pomoc a rady při zpracovávání diplomové práce. Také bych rád poděkoval rodině, přítelkyni a přátelům za psychickou podporu po celou dobu studia.

V Pardubicích dne 17. 05. 2019

Bc. Jiří Vinduška

ANOTACE

Diplomová práce je zaměřena na implementaci prediktivního regulátoru v jednoduchém řídícím systému. Práce popisuje různé metody prediktivního řízení založené na modelu soustavy. Je zde popsána metoda nejmenších čtverců pro získání modelu soustavy, také jsou zde popsány nejčastěji používané modely soustav. V rámci praktické části byly vytvořeny knihovny pro prediktivní řízení a identifikaci soustav v jazyce C++, které lze použít v jednočipových mikroprocesorech.

KLÍČOVÁ SLOVA

Prediktivní řízení, Metoda nejmenších čtverců, Arduino UNO, regulace, modelování a identifikace

TITLE

IMPLEMENTATION OF PREDICTIVE CONTROLLER IN SIMPLE CONTROL SYSTEM

ANNOTATION

The aim of the thesis on the implementation of predictive controller in simple control systems. Various methods of predictive control based on model of the system are described in this thesis. Furthermore, least squares method for obtaining the system model is described as well as the most commonly used process models. Libraries for predictive control and identification of controlled system were made in programming language C++ and can be used in microcontrollers.

KEYWORDS

Predictive control, least squares method, Arduino UNO, regulation, modelling and identification

OBSAH

SEZNAM ZKRATEK A ZNAČEK

AR	Auto-regressive
ARMAX	Auto-regressive moving average with exogenous input
CARIMA	Correlated auto-regressive moving average
GPC	Generalized Predictive Controller
MA	moving average
MPC	Model Predictive Controller
MNČ	Metoda nejmenších čtverců
OE	Output Error
PID	Proportional Integral Derivative controller

SEZNAM SYMBOLŮ PROMĚNNÝCH VELIČIN A FUNKCÍ

α	parametr filtrování žádané hodnoty
d	dopravní zpoždění
e	regulační odchylka
N_1	minimální horizont žádané hodnoty
N_2	maximální horizont žádané hodnoty
N_u	horizont akční veličiny
q	penalizace akční veličiny
r	penalizace regulační odchylky
u	akční veličina
w	žádaná hodnota
y	regulována veličina
Z_s	zesílení soustavy

SEZNAM ILUSTRACÍ

SEZNAM TABULEK

ÚVOD

Prediktivní řízení je moderní metoda pro optimální řízení složitých soustav za respektování omezení a možné využití informace o znalosti dopředných průběhů žádaných hodnot a poruch. Za posledních pár desetiletí si buduje dobrou pověst jak ve vědecké oblasti, tak i v samotném průmyslu. Prakticky se jedná o online optimalizační úlohu, kdy je na základě znalosti modelu řízené soustavy počítán optimální akční zásah při daném horizontu sledování žádané hodnoty a horizontu akční veličiny. Lze takto řídit soustavy s dopravním zpožděním, neminimální fází, nelineárními systémy, vícerozměrovými systémy, které se hůře řídí běžnými regulátory. Důvodem, proč není prediktivní řízení častěji používáno pro řízení v průmyslu, může být například to, že pro většinu regulovaných soustav stačí jednoduché regulátory, že je aplikace prediktivního regulátoru pracná, že návrh negarantuje stabilitu, že záleží na kvalitě modelu řízeného systému. I přes tyto problémy má, ale prediktivní regulace velký potenciál pro aplikace, kde řízení jednoduchými regulátory nestačí.

V této práci bude popsána metodu prediktivního řízení vycházející ze vstupně-výstupního a stavového popisu řízené soustavy, uvedu kritérium a způsob sestavení prediktoru a výpočtu optimálních akčních zásahů. Dále bude uvedena jednoduchá aplikace prediktivního regulátoru pro soustavy prvního řádu s dopravním zpožděním. Budou uvedeny základní metody pro identifikaci soustavy metodou nejmenších čtverců. V praktické části budou implementovány knihovny popsané v teoretické části. Dále bude praktická část obsahovat popis těchto knihoven a budou provedeny regulační experimenty s různými průběhy žádané hodnoty pro reálnou soustavu druhého řádu.

Cílem této práce je zjistit možnost aplikace prediktivního regulátoru v jednoduchých řídicích systémech, porovnání metod prediktivního řízení s klasickým PID regulátorem a otestování regulátorů na reálné soustavě.

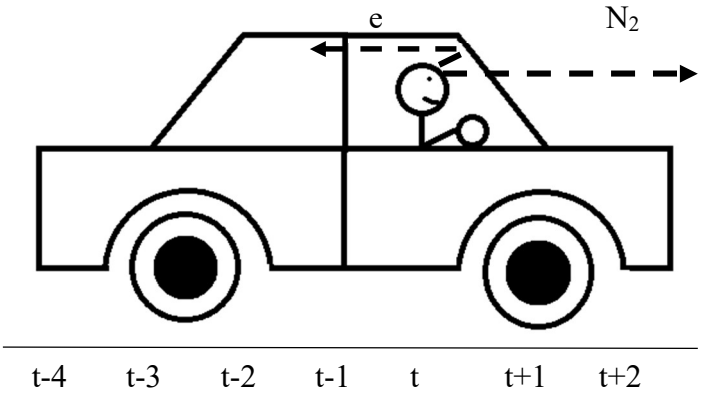
1 TEORETICKÁ ČÁST

1.1 Prediktivní řízení

Popis prvních metod prediktivního řízení se datuje do konce sedmdesátých let minulého století. Termín prediktivní řízení založené na modelu soustavy, anglicky Model Predictive Control (MPC), je souhrnem různých metod, které mají společné vlastnosti. Mezi hlavní společnou vlastnost patří využití modelu pro výpočet predikcí v budoucím čase, všechny metody využívají pro výpočet regulačního zákonu minimalizaci nějakým způsobem zadaného kritéria neboli účelové funkce. Poslední společnou vlastností těchto metod je výpočet budoucích predikcí v každém kroku řízení. V této práci se budu věnovat obecné metodě prediktivního řízení, anglicky Generalized Predictive Control (GPC).

Mezi hlavní výhody MPC patří jeho možnost využití v jednoduchých i velmi složitých systémech, v systémech s dopravním zpožděním, v systémech s neminimální fází, nebo v nestabilních systémech. Pomocí prediktivního řízení lze řídit i více rozměrné soustavy. Prediktivní řízení zavádí do systému zpětnou vazbu v podobě rozdílu mezi měřenou a predikovanou hodnotou regulované veličiny. Výsledný regulační zákon při neuvažování omezení je jednoduchý na implementaci. Prediktivní řízení je velice atraktivní pro neznalou veřejnost. Bohužel, i tento druh řízení má své nevýhody, mezi které můžeme zařadit složitý výpočet regulačního zákonu při daných omezeních pomocí kvadratického programování a nutnosti využití online metody určování parametrů procesu u dynamicky měnících se systémů. Hlavní nevýhodou oproti klasickému řízení pomocí PID regulátoru je nutnost znalosti modelu (Camacho, 2007).

Prediktivní řízení si můžete představit jako řízení automobilu ukázaného na obr. 1, když řidič, prediktivní regulátor, řídí automobil, má nějaký známý omezený horizont řízení N_2. Řidič má pojem o stavu svého vozidla, tudíž zná model vozidla. Má k dispozici různé akční zásahy, jako je přidání plynu, nebo brzdění pro sledování horizontu řízení. Plánuje cestu automobilem za jízdy podle stavu silnice, například do ostré zatáčky si řidič intuitivně nadjede, aby byl průjezd co nejhladší. Klasický PID regulátor by takové auto řídil jen podle minulých regulačních odchylek (e), to je jako kdyby řidič řídil automobil jen pomocí zpětných zrcátek. Toto srovnání není úplně fair vůči klasickým regulátorům, protože nemají znalost horizontu řízení, ve skutečnosti se prediktivní metody řízení využívají jako zadavatelé žádané hodnoty pro PID regulátory (Pekař, 2007).



Obr. 1.1 – Srovnání prediktivní metody řízení pomocí automobilu

1.1.1 Obecná metoda prediktivního řízení

Kritérium

Obecná metoda prediktivního řízení využívá následující kvadratickou formu účelové funkce jako kritérium pro minimalizaci akčních zásahů

$$J = \sum_{i=N_1}^{N_2} r_i\left(\hat{y}\,(k+i) - w(k+i)\right)^2 + \sum_{\mathrm{i}=1}^{N_\mathrm{u}} q_i \Delta u(k+i-1)^2, \tag{1.1}$$

kde $\hat{y}$ je predikovaná regulovaná veličina neboli budoucí výstup soustavy v čase k,

w – budoucí žádaná hodnota a

Δu – budoucí přírůstek akční veličiny.

Nepoužívá se zde absolutní hodnota akční veličiny, protože by z hlediska minimálního kritéria docházelo k trvalé regulační odchylce.

Kritérium má také parametry N_1 minimální horizont sledování žádané veličiny, N_2 maximální horizont sledování žádané veličiny a N_u horizont akční veličiny. Zde platí, že $N_2 > N_1$ a $N_2 > N_\mathrm{u}$. N_1 a N_2 udávají minimální a maximální potřebnou znalost predikce regulované veličiny a žádané hodnoty. N_1 je nastaveno záměrně tak velké, jako je čas dopravního zpoždění soustavy, pokud existuje, jinak je nastaveno na jedničku a hodnota N_2 by měla být tak velká, jako je čas, který soustava potřebuje k dosažení 90 % žádané hodnoty z 10 % žádané hodnoty při jednotkovém skoku a dané periodě vzorkování. Velikostí parametru N_u zvyšujeme, či snižujeme výpočtovou náročnost (Mikleš, 2004).

Parametry r a q jsou váhami kritéria, první váha r penalizací regulační odchylky bude nastavena na 1 a druhou vahou q penalizací akční veličiny se nastaví požadovaná omezení regulátoru. Obě váhy jsou v intervalu od nuly do jedné.

Pokud budoucí žádaná hodnota není známa ($w(k+i)$), lze využít následujícího vztahu rovnice (1.2), pro výpočet optimálních žádaných hodnot pomocí jedné referenční hodnoty $r(k)$, v tomto případě představuje žádanou hodnotu.

$$w(k) = \alpha w(k-1) + (1-\alpha) r(k), \tag{1.2}$$

kde α je parametrem, který se nachází v intervalu $\langle 0,1)$, při nule je průběh žádané hodnoty skokový a čím blíž k jedné, tím je průběh žádané hodnoty hladší (Camacho, 2007).

Maticový zápis kritéria obecné metody prediktivního řízení je následující

$$J = (\hat{\boldsymbol{y}} - \boldsymbol{w})^{\mathrm{T}} \boldsymbol{R} (\hat{\boldsymbol{y}} - \boldsymbol{w}) + \Delta \boldsymbol{u}^{\mathrm{T}} \boldsymbol{Q} \Delta \boldsymbol{u}, \tag{1.3}$$

kde $\hat{\boldsymbol{y}}$ je vektor predikovaných hodnot,

$\boldsymbol{w}$ – vektor budoucích žádaných hodnot,

$\Delta \boldsymbol{u}$ – vektor přírůstků akční veličiny,

$\boldsymbol{R}$ – matice penalizace regulační odchylky a

$\boldsymbol{Q}$ – matice penalizace akční veličiny.

Pro ilustraci zde jsou rozepsány uvedené vektory

$$\hat{\boldsymbol{y}} = \begin{bmatrix} \hat{y}(k+1) \\ \hat{y}(k+2) \\ \hat{y}(k+3) \\ \vdots \\ \hat{y}(k+N_2) \end{bmatrix}, \ \boldsymbol{w} = \begin{bmatrix} w(k+1) \\ w(k+2) \\ w(k+3) \\ \vdots \\ w(k+N_2) \end{bmatrix}, \ \Delta \boldsymbol{u} = \begin{bmatrix} \Delta u(k) \\ \Delta u(k+1) \\ \Delta u(k+2) \\ \vdots \\ \Delta u(k+N_u-1) \end{bmatrix},$$

$$\boldsymbol{R} = \begin{bmatrix} r_1 & 0 & 0 & \cdots & 0 \\ 0 & r_2 & 0 & \cdots & 0 \\ 0 & 0 & r_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & r_{N_2} \end{bmatrix}, \ \mathbf{Q} = \begin{bmatrix} q_1 & 0 & 0 & \cdots & 0 \\ 0 & q_2 & 0 & \cdots & 0 \\ 0 & 0 & q_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & q_{N_2} \end{bmatrix}.$$

Prediktor

Před popisem prediktoru pro řízení zde připomenu dva hlavní popisy soustav. Jedná se o vstupně výstupní popis a stavový popis.

Vstupně–výstupní popis soustavy

V prediktivním řízení je využito diskrétního popisu dynamických systémů pomocí diferenčních rovnic. Vztah mezi vstupem u a výstupem y lineárního diskrétního systému můžeme zapsat diferenční rovnicí s konstantními koeficienty, uvedený v rovnici (1.4)

$$y(k) + a_1 y(k-1) + a_2 y(k-2) + \cdots + a_n y(k-n) = b_1 u(k-1) + b_2 u(k-2) + \cdots b_n u(k-n). \tag{1.4}$$

Minulé hodnoty výstupu jsou použity pro popsání dynamiky systému, hodnota *u(k)* není použita, protože systém reaguje na změnu až v dalším intervalu periody vzorkování. Dolní index n znamená řád soustavy. Pro získání přenosové funkce je použita Z-transformace (Dušek, 2017).

Stavový popis soustavy

Stavový popis soustavy se od vstupně výstupního popisu liší tím, že kromě vstupu a výstupu používá také stav soustavy. Výhodou stavového popisu je jeho obecný zápis pro každý řád soustavy, lišící se pouze v rozměrech matic. Stejně lze popsat i více rozměrné soustavy, které se také liší pouze v rozměrech matic. Následující stavové rovnice jsou použity pro popis lineárního systému

$$\begin{aligned} \boldsymbol{x}(t+1) &= \boldsymbol{A}x(t) + \boldsymbol{B}u(t), \\ y(t) &= \boldsymbol{C}\boldsymbol{x}(t) + \boldsymbol{D}u(t), \end{aligned} \tag{1.5}$$

kde $\boldsymbol{A}$ je maticí dynamiky systému,

$\boldsymbol{B}$ – maticí stavů vstupu,

$\boldsymbol{C}$ – výstupní matice stavu,

$\boldsymbol{D}$ – matice převodu,

$\boldsymbol{x}(t)$ – stavový vektor a

$y(t)$ – výstup systému.

Matice $\boldsymbol{D}$ se v dále popsaných postupech uvažuje nulová. Vstupně výstupní popis lze převést na stavový pomocí matematického postupu snižování derivací. Pomocí vzorců Laplaceovy transformace lze ze stavového popisu vypočítat vstupně výstupní popis (Štěcha, 1999).

Vstupně–výstupní prediktor

Většina vstupně výstupních systému lze po linearizaci popsat uvedeným polynomem

$$A(z^{-1})y(k) = z^{-d}B(z^{-1})u(k-1) + C(z^{-1})e(k), \tag{1.6}$$

kde *A* je výstupním polynomem soustavy,

B – vstupním polynomem soustavy,

C – polynomem poruchy (bílého šumu),

$y(k)$ – výstup soustavy,

$u(k-1)$ – minulý vstup do soustavy,

$e(k)$ – bílý šum, poruchový signál s nulovým průměrem a

d – dopravní zpoždění soustavy.

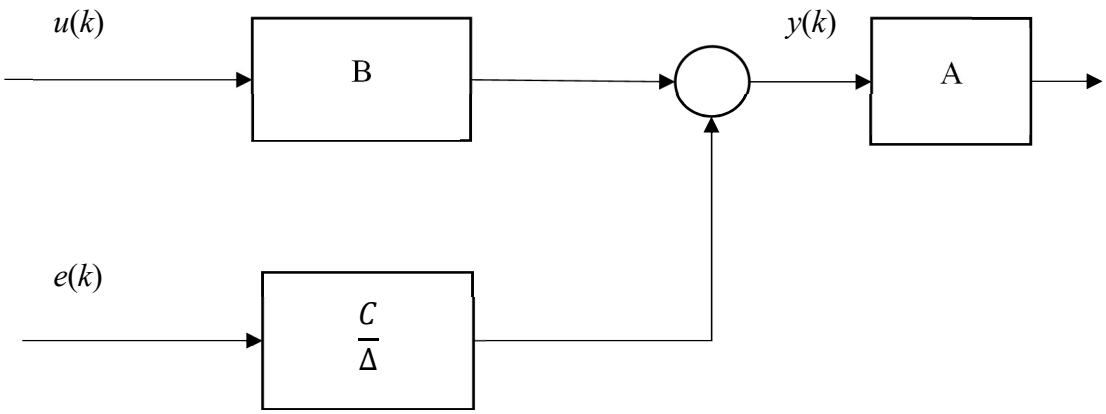
Tento model se skládá z modelu soustavy a modelu poruchy nazývaný ARMA, auto regresivní s pohyblivým průměrem, je vhodnější použít model poruchy CARMA, rovnice (1.7), kontrolovaný auto regresivní model s pohyblivým průměrem. Pokud je porucha nestacionární, je použit přírůstkový tvar, porucha je integrována, tento model se nazývá CARIMA a lze ho popsat rovnicí (Rossiter, 2003)

$$A(z^{-1})y(k) = z^{-d}B(z^{-1})u(k-1) + C(z^{-1})\frac{e(k)}{\Delta}, \tag{1.7}$$

kde $\Delta = 1 - z^{-1}$,

$e(k) = y(k) - \hat{y}(k)$.

Takto se do systému zavede zpětná vazba. Polynom C je volitelný a funguje jako filtrační složka. V běžné praxi se volí polynom prvního řádu, kde první parametr je roven jedné a druhý -0,8. Blokové schéma uvedeného popisu je uvedeno na obr. 1.2. Tento systém je postačujícím popisem většiny technologických procesů (Honc, 2018a).



Obr. 1.2 – Blokové schéma CARIMA modelu soustavy

Pro prediktor musíme provést následující úpravy, při uvažování, že soustava nemá dopravní zpoždění

$$A(z^{-1})y(k) = B(z^{-1})u(k-1) + C(z^{-1})\frac{e(k)}{\Delta}, \tag{1.8}$$

$$\Delta A(z^{-1})y(k) = B(z^{-1})\Delta u(k-1) + C(z^{-1})e(k), \tag{1.9}$$

kde $\Delta = 1 - z^{-1}$.

$$\begin{aligned} \tilde{A} &= \Delta A(z^{-1}) = (1-z^{-1})(1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + + a_n z^{-n}) = 1 + \\ &\quad \tilde{a}_1 z^{-1} + \tilde{a}_2 z^{-2} + \cdots + \tilde{a}_n z^{-n} + \tilde{a}_{n+1} z^{-n-1}, \\ B(z^{-1}) &= b_1 z^{-1} + b_2 z^{-2} + \cdots + b_n z^{-n}, \\ C(z^{-1}) &= 1 + c_1 z^{-1} + c_2 z^{-2} + \cdots + b_{nC} z^{-nC}. \end{aligned} \tag{1.10}$$

Kde *n* je řád soustavy a *nC* je řád filtračního polynomu. Rovnice (1.10) jsou dosazeny do (1.8) a rozepsány predikční rovnice až do $(\mathrm{k} + \mathrm{N}_2)$.

$$\begin{aligned} y(k) + \tilde{a}_1 y(k-1) + \tilde{a}_2 y(k-2) + \cdots + \tilde{a}_{n+1} y(k-n) &= b_1 \Delta u(k-1) + \\ b_2 \Delta u(k-2) + \cdots b_n \Delta u(k-n) + e(k) + c_1 e(k-1) + c_2 e(k-2) &+ \\ \cdots + c_{nC} e(k-nC). \end{aligned}$$

Pro (*k*+1), (*k*+2), ..., (*k*+N_2) následovně

$$\begin{aligned} \tilde{y}(k+1) + \tilde{a}_1 y(k) + \tilde{a}_2 y(k-1) + \cdots + \tilde{a}_{n+1} y(k-n) &= b_1 \Delta u(k) + \\ b_2 \Delta u(k-1) + \cdots b_n \Delta u(k-n+1) + e(k+1) + c_1 e(k) &+ \\ c_2 e(k-1) + \cdots + c_{nC} e(k-nC+1), \end{aligned}$$

$$\begin{aligned} \tilde{y}(k+2) + \tilde{a}_1 \tilde{y}(k+1) + \tilde{a}_2 y(k) + \cdots + \tilde{a}_{n+1} y(k-n+1) &= \\ b_1 \Delta u(k+1) + b_2 \Delta u(k) + \cdots b_n \Delta u(k-n+2) + e(k+2) &+ \\ c_1 e(k+1) + c_2 e(k) + \cdots + c_{nC} e(k-nC+2), \end{aligned}$$

$$\begin{aligned} \tilde{y}(k+N_2) + \tilde{a}_1 \tilde{y}(k+N_2-1) + \tilde{a}_2 \tilde{y}(k+N_2-2) + \cdots + \tilde{a}_{n+1} \tilde{y}(k+ \\ N_2 - n + 1) = b_1 \Delta u(k+N_2-1) + b_2 \Delta u(k-N_2-2) + \\ \cdots b_n \Delta u(k-N_2+n) + e(k+N_2) + c_1 e(k+N_2-1) + c_2 e(k+N_2- \\ 2) + \cdots + c_{nC} e(k+N_2-nC). \end{aligned}$$

Kde budoucí chyby predikce jsou neznámé, tedy *e*(*k*+1), *e*(*k*+2), ..., *e*(*k*+*N*2) = 0. Takto napsané rovnice se dají zapsat přehledněji v maticovém tvaru, predikce jsou na levé straně, známé hodnoty aktuální a minulé na pravé straně

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ \tilde{a}_1 & 1 & 0 & 0 & 0 & 0 \\ \tilde{a}_2 & \tilde{a}_1 & 1 & 0 & 0 & 0 \\ \tilde{a}_3 & \tilde{a}_2 & \cdots & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \tilde{y}(k+1) \\ \tilde{y}(k+2) \\ \tilde{y}(k+3) \\ \vdots \\ \tilde{y}(k+N_2-1) \\ \tilde{y}(k+N_2) \end{bmatrix} =$$

$$\begin{bmatrix} b_1 & 0 & 0 & 0 & 0 & 0 \\ b_2 & b_1 & 0 & 0 & 0 & 0 \\ b_3 & b_2 & b_1 & 0 & 0 & 0 \\ b_4 & b_3 & \cdots & b_1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & b_1 \end{bmatrix} \begin{bmatrix} \Delta u(k) \\ \Delta u(k+1) \\ \Delta u(k+2) \\ \vdots \\ \Delta u(k+N_u-2) \\ \Delta u(k+N_u-1) \end{bmatrix} +$$

$$\begin{bmatrix} -\tilde{a}_1 & -\tilde{a}_2 & -\tilde{a}_3 & -\tilde{a}_4 & \cdots & -\tilde{a}_{n+1} \\ -\tilde{a}_2 & -\tilde{a}_3 & -\tilde{a}_4 & \cdots & -\tilde{a}_{n+1} & 0 \\ -\tilde{a}_3 & -\tilde{a}_4 & \cdots & -\tilde{a}_{n+1} & 0 & 0 \\ -\tilde{a}_4 & \cdots & -\tilde{a}_{n+1} & 0 & \cdots & 0 \\ \vdots & \iddots & 0 & 0 & \cdots & 0 \\ -\tilde{a}_{n+1} & 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} y(k) \\ y(k-1) \\ y(k-2) \\ \vdots \\ y(k-n-1) \\ y(k-n) \end{bmatrix} +$$

$$\begin{bmatrix} b_2 & b_3 & b_4 & b_5 & \cdots & b_n \\ b_3 & b_4 & b_5 & \cdots & b_n & 0 \\ b_4 & b_5 & \cdots & b_n & 0 & 0 \\ b_5 & \cdots & b_n & 0 & \cdots & 0 \\ \vdots & \iddots & 0 & 0 & \cdots & 0 \\ b_n & 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} \Delta u(k-1) \\ \Delta u(k-2) \\ \vdots \\ \Delta u(k-n) \\ \Delta u(k-n+1) \end{bmatrix} +$$

$$\begin{bmatrix} c_1 & c_2 & c_3 & c_4 & \cdots & c_{nC} \\ c_2 & c_3 & c_4 & \cdots & c_{nC} & 0 \\ c_3 & c_4 & \cdots & c_{nC} & 0 & 0 \\ c_4 & \cdots & c_{nC} & 0 & \cdots & 0 \\ \vdots & \iddots & 0 & 0 & \cdots & 0 \\ c_{nC} & 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} e(k) \\ e(k-1) \\ e(k-2) \\ \vdots \\ e(k-nC) \\ e(k-nC+1) \end{bmatrix}.$$

Dále označím všechny matice odpovídajícími symboly

$$\boldsymbol{A}_p\hat{\boldsymbol{y}} = \boldsymbol{B}_p\Delta\hat{\boldsymbol{u}} + \boldsymbol{A}_m\boldsymbol{y} + \boldsymbol{B}_m\Delta\boldsymbol{u} + \boldsymbol{C}_m\boldsymbol{e}, \tag{1.11}$$

kde $\hat{\boldsymbol{y}}$ je vektor predikovaných výstupů,

$\Delta\hat{\boldsymbol{u}}$ – vektor predikovaných přírůstků k akční veličině,

$\boldsymbol{y}$ – vektor minulých výstupů,

$\Delta\boldsymbol{u}$ – vektor minulých přírůstků k akční veličině a

$\boldsymbol{e}$ – vektor minulých chyb predikcí (Honc, 2018a).

Pro vyjádření prediktoru pokračuji vyjádřením $\hat{\boldsymbol{y}}$

$$\hat{\boldsymbol{y}} = \boldsymbol{A}_p{}^{-1}\boldsymbol{B}_p\Delta\hat{\boldsymbol{u}} + \boldsymbol{A}_p{}^{-1}\boldsymbol{A}_m\boldsymbol{y} + \boldsymbol{A}_p{}^{-1}\boldsymbol{B}_m\Delta\boldsymbol{u} + \boldsymbol{A}_p{}^{-1}\boldsymbol{C}_m\boldsymbol{e}.$$

Označení matic příslušnými symboly a úprava

$$\hat{\boldsymbol{y}} = \boldsymbol{G}\Delta\hat{\boldsymbol{u}} + \boldsymbol{F_y y} + \boldsymbol{F_u}\Delta\boldsymbol{u} + \boldsymbol{F_e e},$$

$$\hat{\boldsymbol{y}} = \boldsymbol{G}\Delta\hat{\boldsymbol{u}} + [\boldsymbol{F_y} \quad \boldsymbol{F_u} \quad \boldsymbol{F_e}]\begin{bmatrix}\boldsymbol{y}\\ \Delta\boldsymbol{u}\\ \boldsymbol{e}\end{bmatrix},$$

$$\hat{\boldsymbol{y}} = \boldsymbol{G}\Delta\hat{\boldsymbol{u}} + \boldsymbol{F_p x_p}. \tag{1.12}$$

Rovnice (1.12) je výsledná rovnice prediktoru, kde $\boldsymbol{G}\Delta\hat{\boldsymbol{u}}$ je volná odezva prediktoru a $\boldsymbol{F}_{\mathbf{p}}\boldsymbol{x}_{\mathbf{p}}$ je vnucená odezva prediktoru.

Stavový prediktor

Pro odvození stavového prediktoru bude využito rovnic (1.5) a přidáním dalšího stavu v podobě přírůstků akčního zásahu

$$u(t) = u(t-1) + \Delta u(t), \tag{1.13}$$

$$\begin{bmatrix}x(t+1)\\ u(t)\end{bmatrix} = \begin{bmatrix}\boldsymbol{A} & \boldsymbol{B}\\ \boldsymbol{0} & \boldsymbol{I}\end{bmatrix}\begin{bmatrix}x(t)\\ u(t-1)\end{bmatrix} + \begin{bmatrix}\boldsymbol{B}\\ \boldsymbol{I}\end{bmatrix}\Delta u(t),$$

$$y_{(k)} = [\boldsymbol{C} \quad \boldsymbol{D}]\begin{bmatrix}x(t)\\ u(t-1)\end{bmatrix}, \tag{1.14}$$

kde $\begin{bmatrix}x(t+1)\\ u(t)\end{bmatrix}$ je nový stavový vektor $\boldsymbol{x_p}$(k+1), $u_{(t)}$ představuje nový stav systému, zápis matic 1.14 lze přepsat s novým označením jako

$$\begin{aligned} x_p(k+1) &= \boldsymbol{M x_p}(k) + \boldsymbol{N}\Delta u(t),\\ y(k) &= \boldsymbol{O x_p}(k). \end{aligned} \tag{1.15}$$

Analogicky se vstupně výstupním prediktorem lze rozepsat další predikce až do N_2

$$\boldsymbol{x_p}(k+1) = \boldsymbol{M x_p}(k) + \boldsymbol{N}\Delta u(k),$$

$$\mathrm{y(k)} = \mathbf{O}\mathbf{x}_{\mathbf{p}}\mathrm{(k)},$$

$$\boldsymbol{x_p}(k+2) = \boldsymbol{M}x_p(k+1) + \boldsymbol{N}\Delta u(k+1) = \boldsymbol{M}\left(\boldsymbol{M x_p}(k) + \boldsymbol{N}\Delta u(k)\right) +$$

$$\boldsymbol{N}\Delta u(k+1) = \boldsymbol{M^2 x_p}(k) + \boldsymbol{MN}\Delta u(k) + \boldsymbol{N}\Delta u(t+1),$$

$$y(k+1) = \mathbf{O}\boldsymbol{x}_{\mathbf{p}}\mathrm{(k+1)} = \boldsymbol{OM}x_p(k) + \boldsymbol{ON}\Delta u(k),$$

$$\boldsymbol{x_p}(k+3) = \boldsymbol{M}\boldsymbol{x_p}(k+2) + \boldsymbol{N}\Delta u(k+2) = \boldsymbol{M^3}\boldsymbol{x_p}(k) + \boldsymbol{M^2}\boldsymbol{N}\Delta u(k) + \boldsymbol{M}\boldsymbol{N}\Delta u(k+1) + \boldsymbol{N}\Delta u(k+2),$$

$$y(k+2) = \boldsymbol{O}\boldsymbol{x_p}(k+2) = \boldsymbol{O}\boldsymbol{M^3}\boldsymbol{x_p}(k) + \boldsymbol{O}\boldsymbol{M^2}\boldsymbol{N}\Delta u(k) + \boldsymbol{O}\boldsymbol{M}\boldsymbol{N}\Delta u(k+1) + \boldsymbol{O}\boldsymbol{N}\Delta u(k+2),$$

$$\boldsymbol{x_p}(k+N_2+1) = \boldsymbol{M^{N_2}}\boldsymbol{x_p}(k+N_2) + \sum_{i=0}^{N_2-1} \boldsymbol{M}^{N_1-i-1}\boldsymbol{N}\Delta u(k+i)$$

$$y(t+\mathrm{N}_2) = \boldsymbol{O}\boldsymbol{M^{N_2}}\boldsymbol{x}_{\mathbf{p}}(\mathrm{k}+\mathrm{N}_2+1) + \sum_{i=0}^{N_2-1} \boldsymbol{O}\boldsymbol{M}^{N_1-i-1}\boldsymbol{N}\Delta u(k+i)$$

Predikce $y(t+n)$ zapsané maticově jako

$$\begin{bmatrix} y(t+1) \\ y(t+2) \\ y(t+3) \\ \vdots \\ y(t+N_2) \end{bmatrix} = \begin{bmatrix} \boldsymbol{O}\boldsymbol{N} & 0 & 0 & 0 & 0 \\ \boldsymbol{O}\boldsymbol{M}\boldsymbol{N} & \boldsymbol{O}\boldsymbol{N} & 0 & 0 & 0 \\ \boldsymbol{O}\boldsymbol{M^2}\boldsymbol{N} & \boldsymbol{O}\boldsymbol{M}\boldsymbol{N} & \boldsymbol{O}\boldsymbol{N} & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ \boldsymbol{O}\boldsymbol{M}^{\boldsymbol{N_2-1}}\boldsymbol{N} & \boldsymbol{O}\boldsymbol{M}^{\boldsymbol{N_2-2}}\boldsymbol{N} & \boldsymbol{O}\boldsymbol{M}^{\boldsymbol{N_2-3}}\boldsymbol{N} & \cdots & \boldsymbol{O}\boldsymbol{M}^{\boldsymbol{N_2-N_u}}\boldsymbol{N} \end{bmatrix} \cdot \begin{bmatrix} \Delta u(k) \\ \Delta u(k+1) \\ \Delta u(k+2) \\ \vdots \\ \Delta u(k+N_u) \end{bmatrix} + \begin{bmatrix} \boldsymbol{O}\boldsymbol{M} \\ \boldsymbol{O}\boldsymbol{M^2} \\ \boldsymbol{O}\boldsymbol{M^3} \\ \vdots \\ \boldsymbol{O}\boldsymbol{M^{N_2}} \end{bmatrix} \boldsymbol{x_p}(k).$$

Tento zápis může být matoucí, protože matice $\boldsymbol{M}$, $\boldsymbol{N}$ a $\boldsymbol{O}$ se nachází uvnitř další matice. Označím uvedené matice příslušnými symboly

$$\widehat{\boldsymbol{y}} = \boldsymbol{G}\Delta\widehat{\boldsymbol{u}} + \boldsymbol{F_p}\boldsymbol{x_p}. \tag{1.16}$$

Prediktor vstupně výstupní, rovnice (1.12), a stavový, rovnice (1.16), se v konečném výsledku po označení matic shodují. Můžeme tedy použít jakýkoliv prediktor pro vyjádření regulačního zákonu, lišit se bude pouze konstantami uvnitř matic $\boldsymbol{G}$ a $\boldsymbol{F}_{\mathbf{p}}$ v matici $\boldsymbol{x}_{\mathbf{p}}$, kde v případě vstupně výstupního bude obsahovat vektory $\boldsymbol{y}$, $\Delta\boldsymbol{u}$ a $\boldsymbol{e}$ a u stavového popisu bude obsahovat upravený stavový vektor s přírůstkem akční veličiny $\boldsymbol{x}_{\mathbf{p}}(k)$ (Honc, 2018a).

Regulační zákon

Regulační zákon prediktivního regulátoru získáme dosazením rovnice (1.12), nebo (1.16) do kritéria (1.3) a minimalizováním funkce vůči přírůstkům akční veličiny. Pokud

$$\boldsymbol{F_p x_p} = \boldsymbol{f}.$$

Pak po dosazení

$$J = (\boldsymbol{G}\Delta\hat{\boldsymbol{u}} + \boldsymbol{f} - \boldsymbol{W})^{\mathrm{T}}\boldsymbol{R}(\boldsymbol{G}\Delta\hat{\boldsymbol{u}} + \boldsymbol{f} - \boldsymbol{w}) + \Delta\boldsymbol{u}^{\mathrm{T}}\boldsymbol{Q}\Delta\boldsymbol{u} = (\Delta\hat{\boldsymbol{u}}^{\mathrm{T}}\boldsymbol{G}^{\mathrm{T}} + \boldsymbol{f}^{\mathrm{T}} - \boldsymbol{w}^{\mathrm{T}})\boldsymbol{R}(\boldsymbol{G}\Delta\hat{\boldsymbol{u}} + +\boldsymbol{f} - \boldsymbol{w}) + \Delta\boldsymbol{u}^{\mathrm{T}}\boldsymbol{Q}\Delta\boldsymbol{u} = \Delta\hat{\boldsymbol{u}}^{\mathrm{T}}\boldsymbol{G}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{G}\Delta\hat{\boldsymbol{u}} + \Delta\hat{\boldsymbol{u}}^{\mathrm{T}}\boldsymbol{G}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{f} - \Delta\hat{\boldsymbol{u}}^{\mathrm{T}}\boldsymbol{G}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{w} + \boldsymbol{f}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{G}\Delta\hat{\boldsymbol{u}} + \boldsymbol{f}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{f} — \boldsymbol{f}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{w} - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{G}\Delta\hat{\boldsymbol{u}} - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{f} + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{w} + \Delta\boldsymbol{u}^{\mathrm{T}}\boldsymbol{Q}\Delta\boldsymbol{u},$$

$$J = \Delta\hat{\boldsymbol{u}}^{\mathrm{T}}\underbrace{(\boldsymbol{G}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{G} + \boldsymbol{Q})}_{H}\Delta\hat{\boldsymbol{u}} + \Delta\hat{\boldsymbol{u}}^{\mathrm{T}}\underbrace{\boldsymbol{G}^{\mathrm{T}}\boldsymbol{R}(\boldsymbol{f} - \boldsymbol{w})}_{g} + \underbrace{(\boldsymbol{f}^{\mathrm{T}} - \boldsymbol{w}^{\mathrm{T}})\boldsymbol{R}\boldsymbol{G}}_{g}\Delta\hat{\boldsymbol{u}} + \underbrace{(\boldsymbol{f}^{\mathrm{T}} - \boldsymbol{w}^{\mathrm{T}})\boldsymbol{R}(\boldsymbol{f} - \boldsymbol{w})}_{k}.$$

$$J = \Delta\hat{\boldsymbol{u}}^{\mathrm{T}}\boldsymbol{H}\Delta\hat{\boldsymbol{u}} + 2\boldsymbol{g}\Delta\hat{\boldsymbol{u}} + k. \tag{1.17}$$

Tato kvadratická forma kritéria se minimalizuje vůči $\Delta\hat{\boldsymbol{u}}$, toho lze dosáhnout analyticky bez uvažování omezení, anebo při uvažování omezení musíme využít metod kvadratického programování, například pomocí metody aktivních množin (Honc, 2018a).

V reálném světě se vždy vyskytují nějaká omezení, minimálně co se týče omezení akčních veličin. Metody kvadratického programování jsou nad rámec testování aplikací prediktivních regulátorů této diplomové práce, a proto se jim nebudu věnovat. Pokud dále v textu bude zmiňován regulační zákon, bude ve tvaru bez uvažování omezení (Bertsekas, 1999).

Pomocí derivování kritéria podle $\Delta\hat{\boldsymbol{u}}$ lze vyjádřit minimum funkce

$$J = \frac{1}{2}\Delta\hat{\boldsymbol{u}}^{\mathrm{T}}\boldsymbol{H}\Delta\hat{\boldsymbol{u}} + \boldsymbol{g}\Delta\hat{\boldsymbol{u}} + \frac{k}{2},$$

$$\frac{\mathrm{d}J}{\mathrm{d}\Delta\hat{\boldsymbol{u}}} = \frac{1}{2}\Delta\hat{\boldsymbol{u}}^{\mathrm{T}}\mathbf{H}\Delta\hat{\boldsymbol{u}} + \boldsymbol{g}\Delta\hat{\boldsymbol{u}} + \frac{\mathrm{k}}{2} = \mathbf{H}\Delta\hat{\boldsymbol{u}} + \boldsymbol{g}.$$

Minimum funkce je tedy

$$\boldsymbol{H}\Delta\hat{\boldsymbol{u}} + \boldsymbol{g} = 0. \tag{1.18}$$

Vyjádřením $\Delta\hat{\boldsymbol{u}}$

$$\Delta\hat{\boldsymbol{u}} = -\boldsymbol{H}^{-1}\boldsymbol{g},$$

a dosazení z rovnice (1.18)

$$\Delta\hat{\boldsymbol{u}} = -(\boldsymbol{G}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{G} + \boldsymbol{Q})^{-1}\boldsymbol{G}^{\mathrm{T}}\boldsymbol{R}(\boldsymbol{f} - \boldsymbol{w}) = (\boldsymbol{G}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{G} + \boldsymbol{Q})^{-1}\boldsymbol{G}^{\mathrm{T}}\boldsymbol{R}\big(\boldsymbol{w} - \boldsymbol{F_p}\boldsymbol{x_p}\big),$$

$$\Delta\hat{\boldsymbol{u}} = \boldsymbol{L}\big(\boldsymbol{w} - \boldsymbol{F_p}\boldsymbol{x_p}\big). \tag{1.19}$$

Rovnice (1.19) je výsledným regulačním zákonem, který je stejný jak pro vstupně výstupní prediktor, tak pro stavový prediktor (Honc, 2018a).

1.1.2 Jednoduchá aplikace obecné metody predikce pro soustavy prvního řádu s dopravním zpožděním

V průmyslu převažuje klasické řízení pomocí PID regulátoru nad prediktivním, protože klasický PID regulátor jde velmi snadno nastavit a ladit během provozu. Existuje mnoho metod nastavení PID regulátoru, jako je Ziegler Nicholsova metoda z přechodové charakteristiky, nebo kritických kmitů, Kuhnova metoda vypočtením obsahu nad křivkou atp. Technici v průmyslu nastavují PID regulátor ze zkušeností a kvalita regulace je velmi podobná, jako při nastavení pomocí zmíněných metod (Camacho, 2007).

Prediktivní řízení je pro regulaci takových soustav mnohem složitější na výpočet. Vypočtení matic $\boldsymbol{G}$ a $\boldsymbol{F}$**p** zabere více času než nastavení PID regulátoru, a pokud se proces mění s časem, musí regulátor tyto matice vypočítávat v průběhu, což stojí mnoho výpočetního času, který se musí využít jinak než na výpočet. V této kapitole ukážu nástroje pro jednoduchou implementaci prediktivního regulátoru. Pokud je znám model soustavy a penalizace akční veličiny se v průběhu času nebude měnit, výpočet parametrů může proběhnout pouze na počátku, stejně tak jako u PID regulátoru, a poté probíhá jen krátký výpočet regulačního zákonu, který již nestojí tolik výpočetního času. Tento způsob lze použít, i pokud neznáme přesný model a používáme některou z online metod nejmenších čtverců na její aproximaci, protože se zde nevypočítávají celé matice, ale pouze pár parametrů (Wellstead, 1991).

Aproximace soustavy vyššího řádu na soustavu prvního řádu s dopravním zpožděním

Většina systémů v průmyslu může být popsána lineárním modelem vyššího řádu, ale tento model vyššího řádu se většinou skládá z mnoha systémů prvního řádu položených za sebou. Například systém se dvěma stejnými nádržemi, kde z jedné nádrže kapalina teče do druhé a z druhé teče do jiného zásobníku. První nádrž lze popsat modelem prvního řádu a druhou také, ale výsledný model je druhého řádu s dvojnásobnou časovou konstantou (Camacho, 2007)

$$F(s) = \frac{Z_s}{(\tau s + 1)^N}, \tag{1.20}$$

kde Zs je zesílení soustavy,

N – řád soustavy a

τ – časová konstanta.

Takový model vyššího řádu lze aproximovat modelem prvního řádu s dopravním zpožděním pomocí následující rovnice

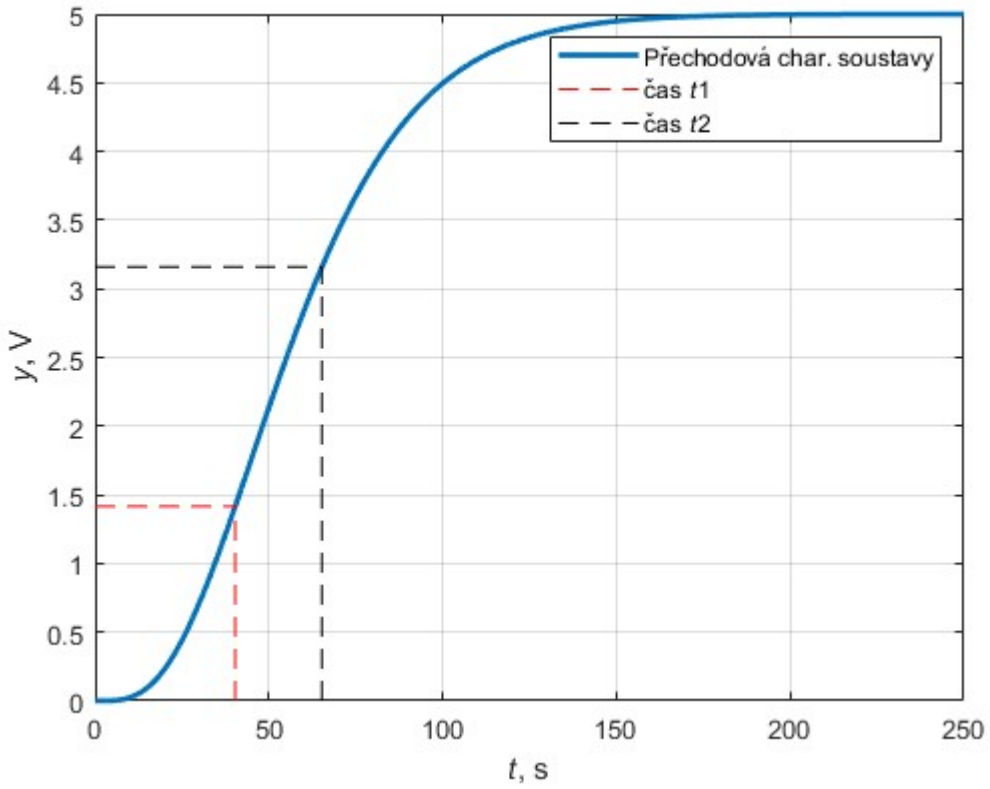
$$F(s) = \frac{Z_s}{(\tau s + 1)} e^{-s\tau_d}, \tag{1.21}$$

kde τ_d je dopravní zpoždění soustavy.

Pro ilustraci, jak postupovat při aproximaci, byl vytvořen model soustavy pátého řádu

$$F(s) = \frac{5}{(15s + 1)^5}. \tag{1.22}$$

Přechodová charakteristika je zobrazena na následujícím obr. 1.3.



Obr. 1.3 – Přechodová charakteristika soustavy pátého řádu

Parametry τ a τ_d pro aproximaci soustavou prvního řádu s dopravním zpožděním se vypočtou pomocí následujících vztahů

$$\tau = 1{,}5(t_2 - t_1),$$

$$\tau_d = 1{,}5(t_1 - \tfrac{1}{3}t_2).$$

Časy t_1, t_2 je možné zjistit z přechodové charakteristiky. Parametr t_1 je čas kdy y dosáhne 28,3 % své ustálené hodnoty a čas t_2 je 63,2 % ustálené regulované hodnoty. Vypočtené hodnoty se poté dosazují do rovnice (1.21). Pokud není známo zesílení soustavy, lze ho vypočítat pomocí následujícího vztahu

$$Z_s = \frac{\Delta y}{\Delta u}.$$

Pokud časová konstanta dopravního zpoždění je násobkem periody vzorkování (T_s), spojitý model soustavy prvního řádu s dopravním zpožděním se dá převést na diskrétní model soustavy prvního řádu s dopravním zpožděním pomocí následujících vztahů

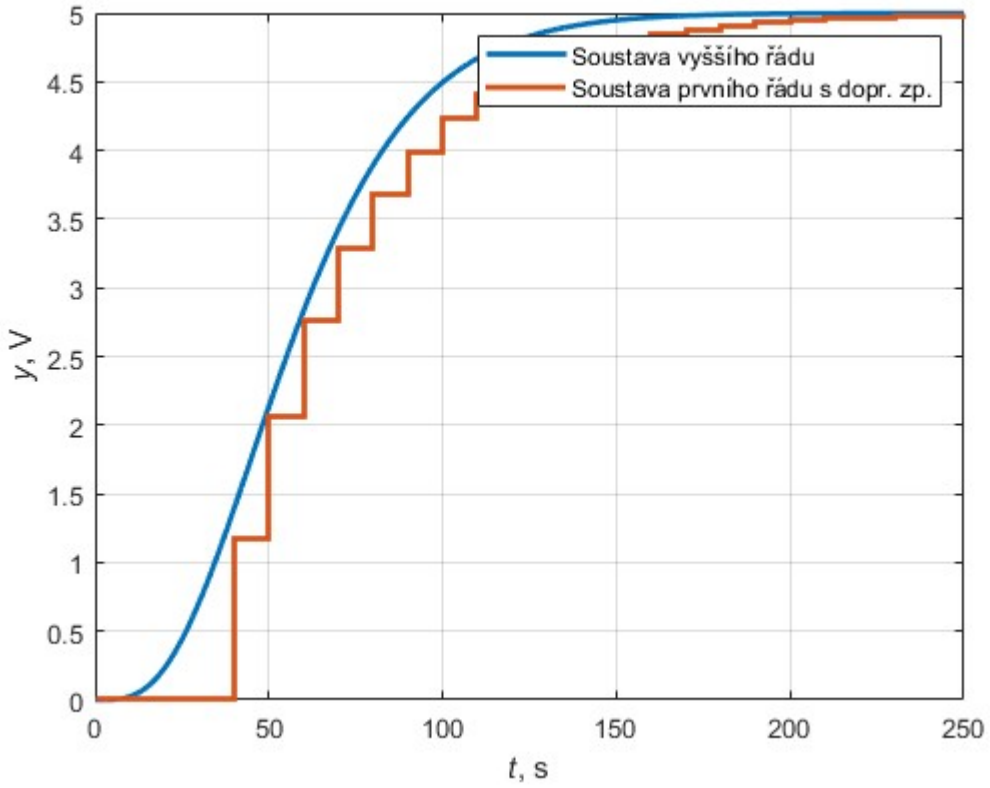
$$a = -e^{-\frac{\tau}{T_s}},$$

$$b = Z_s(1 + a),$$

$$d = \frac{\tau_d}{T_s}.$$

Tyto vypočtené parametry se poté dosadí do diskrétního popisu soustavy prvního řádu s dopravním zpožděním (Camacho, 2007)

$$G(z^{-1}) = \frac{bz^{-1}}{1 + az^{-1}} z^{-d}. \tag{1.23}$$



Obr. 1.4 – Výsledek aproximace soustavou prvního s dopravním zpožděním

Na obr. 1.4 je zobrazena výsledná diskrétní soustava prvního řádu s dopravním zpožděním a periodou vzorkování 10 sekund v porovnání s původní soustavou pátého řádu.

Vyjádření regulačního zákonu

Pokud použijeme pro popis soustavy prvního řádu s dopravním zpožděním CARIMA model uvedený v rovnici (1.7), můžeme soustavu popsat následující diferenční rovnicí za předpokladu, že filtrační polynom C je rovný jedné, potom

$$y(k) + ay(k-1) = bz^{-d}u(k-1) + \frac{e(k)}{\Delta}.$$

Stejným upravením, jakým bylo postupováno v minulé kapitole, je možné rovnici rozepsat na

$$y(k) = (1-a)y(k-1) + ay(k-2) + b\Delta u(k-1-d) + e(k).$$

První predikce (k+1)

$$y(k+1) = (1-a)y(k) + ay(k-1) + b\Delta u(k-d) + e(k+1).$$

Kde $e(k+1)$ je rovno 0 a $e(k)$ nehraje v dalších výpočtech žádnou roli, kvůli zvolení polynomu C rovného 1. Obecně je rovnice rozepsána jako

$$y(k+d+i) = (1-a)y(k+d+i) + ay(k+d+i-1) + b\Delta u(k+d+i-2). \tag{1.24}$$

Rovnici (1.24) je možno rozepsat a vytvořit prediktor stejně jako v případě rovnice (1.12) a předcházejících. Takto vytvořený prediktor dosadíme do kritéria, kde $N_1 = d + 1$, N_2 je libovolný horizont řízení a $N_u = N_2 - d$. Výsledný regulační zákon je ve tvaru

$$\Delta\hat{\boldsymbol{u}} = \boldsymbol{L}(\boldsymbol{w} - \boldsymbol{F_p x_p}). \tag{1.25}$$

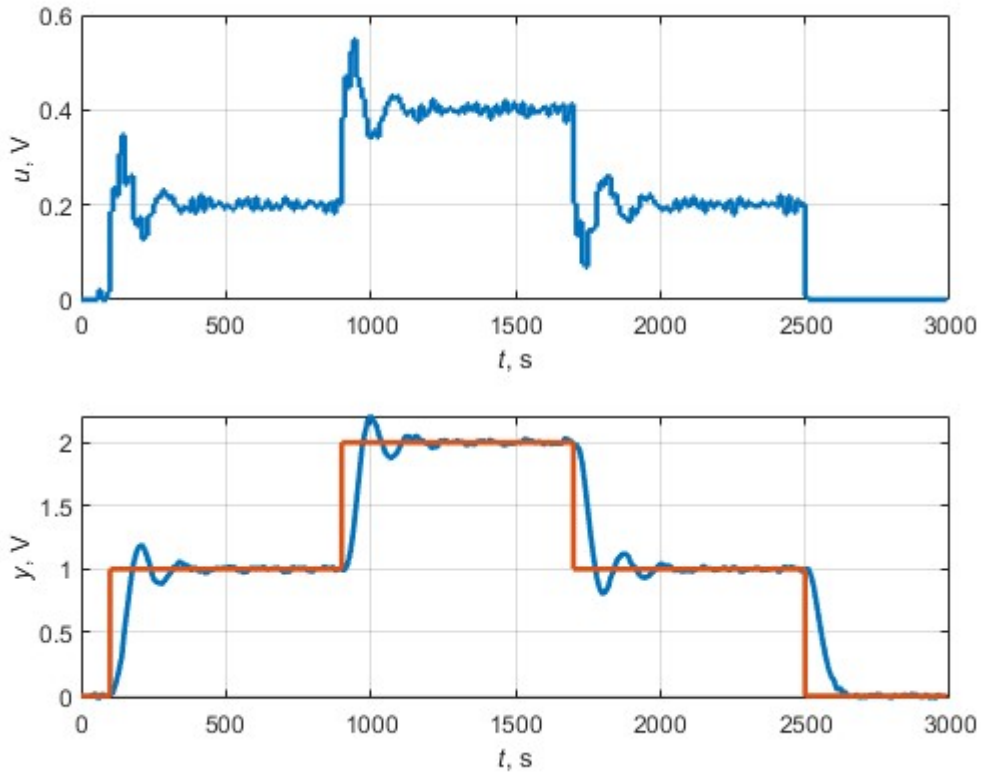
Pokud není znám vektor $\boldsymbol{w}$, známe pouze jednu žádanou hodnotu označenou r, jako v případě PID regulátoru, můžeme regulační zákon 1.25 přepsat do tvaru

$$\Delta u = l_r r(k) + l_{y1} y(k+d) + l_{y2} y(k+d-1). \tag{1.26}$$

Parametry l_r, l_{y1}, l_{y2} pro pevný horizont žádané hodnoty $N_2 = 15$, který je dostačující pro většinu technologických procesů, je možné vypočítat pomocí dalších parametrů k_{11}, k_{21}, k_{31}, k_{12}, k_{22}, k_{32}, které byly vypočteny autory Camacho a Bordons(2007). Pomocí nich lze vypočítat velice přesné hodnoty l_r, l_{y1}, l_{y2}, které se blíží skutečným hodnotám, jinak by se musely vypočítat pomocí vztahů uvedených v kapitole 1.1.3. Parametry se dají vypočítat pomocí následujících rovnic

$$\begin{aligned}
k_{11} &= -e^{(0{,}3598 - 0{,}9127q + 0{,}3165q^2)}, \\
k_{21} &= -e^{(0{,}0875 - \ ,2309q \quad ,508 \quad ^2)}, \\
k_{31} &= 1{,}05, \\
k_{12} &= e^{(-1{,}7383 \quad ,404 \quad)}, \\
k_{22} &= e^{(-0{,}32157 - 0{,}8192 \quad \ ,3109q^2)}, \\
k_{32} &= 1{,}045. \\
l_{y1} &= k_{11} - k_{21}\frac{a}{(k_{31}+a)}, \\
l_{y2} &= k_{21} - k_{22}\frac{a}{(k_{32}+a)}, \\
l_r &= -l_{y1} - l_{y2}.
\end{aligned} \tag{1.27}$$

Tyto výpočty se provedou pouze na počátku procesu. Při regulaci probíhá výpočet pouze prediktoru rovnice (1.24) a regulačního zákonu (1.26). Regulační zákon musí být vynásoben převrácenou hodnotou zesílení soustavy tak, aby z regulátoru a soustavy vznikl zpětnovazební systém se zesílením rovný jedné. Na následujícím obrázku (obr.1.5) je zobrazen odsimulovaný regulační pochod uvedeného regulátoru se soustavou uvedenou v rovnici (1.22) (Camacho, 2007). Horní graf zobrazuje akční zásah, dolní graf zobrazuje oranžovým průběhem žádanou hodnotu a modrým regulovanou veličinu.



Obr. 1.5 – Regulační pochod prediktivního regulátoru pro s. prvního řádu. s dopr. zp.

1.2 Modelování a identifikace

Pro popis dynamických technologických procesů lze využít aparát matematicko-fyzikální analýzy, kde nejprve bilancujeme energii vstupů a výstupů systému, a poté vytváříme přesný matematicko-fyzikální model. Výsledkem tohoto modelu je přesná reprezentace stavů systému a je většinou nelineární. Pro účely prediktivního řízení je nutné získat model lineární, kde jsou parametry známé a neměnné. Tento model již není přesnou interpretací reálných stavů

systému. Parametry systému se získávají experimentováním s reálnou soustavou. Většinou do soustavy posíláme známý vstup a měříme odezvu systému, se kterou je dále pracováno.

Pokud je možné využít některého programového vybavení, které umí simulovat lineární systémy, například programovací jazyky R a Python, nebo MATLAB, lze použít deterministický model, ve kterém nejsou uvažovány chyby. Pomocí modelu se vypočítá odhadovaný výstup, který se porovná se skutečnou hodnotou a minimalizují se čtverce odchylek mezi odhadovaným výstupem a reálným výstupem. Používá se kritérium metody nejmenších čtverců

$$K = \sum_{k=1}^{N} (\tilde{y}(i) - y(i))^2, \tag{1.28}$$

kde $\tilde{y}$ je odhadovaný výstup soustavy (neplést s predikovaným výstupem soustavy) a

y – reálný (naměřený) výstup soustavy.

Metodu nejmenších čtverců je vhodné použít při popisu soustav pomocí stochastických modelů a jejich diferenčních rovnic. Stochastické modely počítají s náhodnou poruchou soustavy (Honc, 2018b).

1.2.1 Modely systémů

Deterministické modely

Pokud nejsou uvažovány náhodné vlivy, je možné využít přenosu systémů k popisu soustav. Mezi nejčastěji používané modely lze zařadit soustavu prvního řadu bez dopravního zpoždění, soustavu prvního řadu s dopravním zpožděním, soustavu druhého řadu s dvojnásobnou časovou konstantou, soustava druhého řadu s dvěma různými časovými konstantami a soustava vyššího řadu.

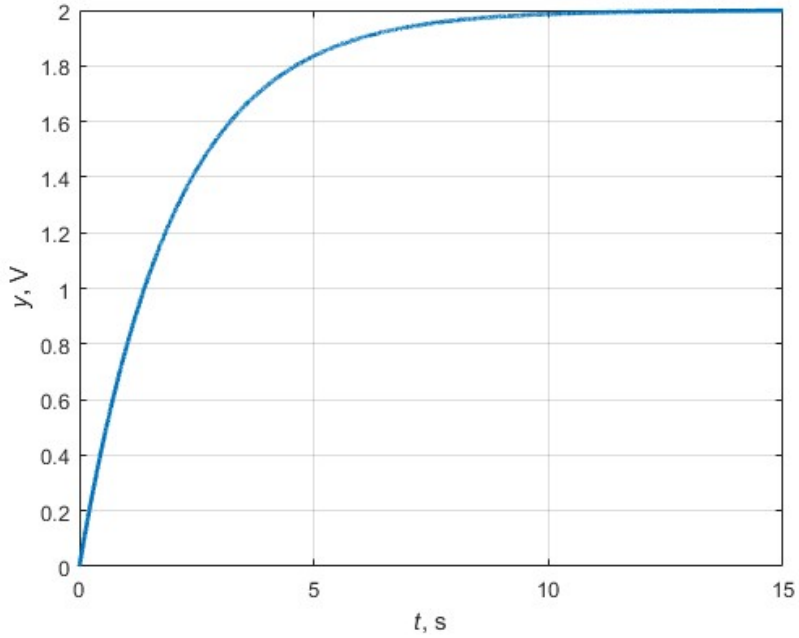
Soustava prvního řádu

Soustava prvního řádu bez dopravního zpoždění je popsána přenosem ve tvaru

$$F(s) = \frac{Z_s}{\tau s + 1}, \tag{1.29}$$

kde Z_s je zesílení soustavy a

τ – časová konstanta soustavy.



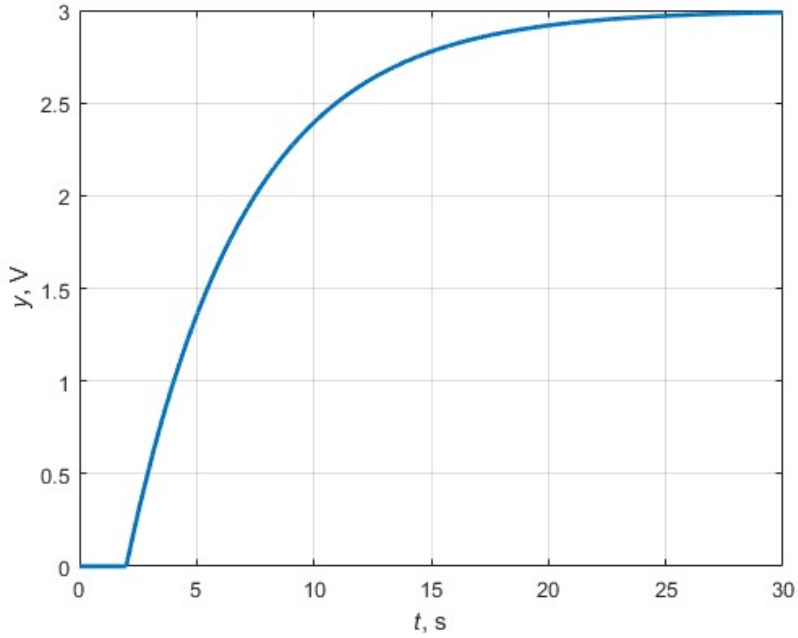
Obr. 1.6 – Přechodová charakteristika systému popsaného rovnicí (1.29)

Soustava prvního řádu s dopravním zpožděním

Soustava prvního řádu s dopravním zpožděním je popsána přenosem ve tvaru

$$F(s) = \frac{Z_s}{(\tau s + 1)} e^{-s\tau_d}, \tag{1.30}$$

kde τ_d je dopravní zpoždění soustavy.



Obr. 1.7 – Přechodová charakteristika systému popsaného rovnicí (1.30)

Soustava druhého řádu

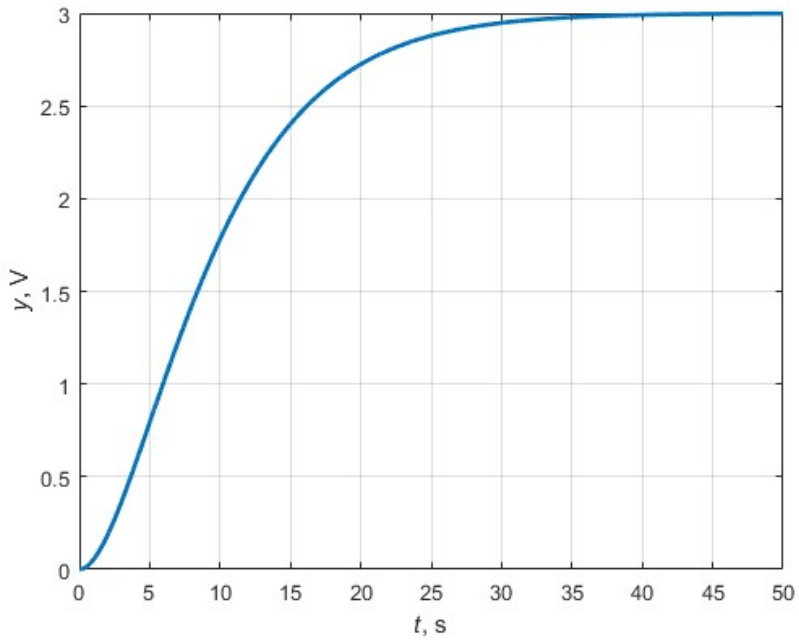
Obecně lze zapsat přenos pro soustavy druhého řádu takto

$$F(s) = \frac{Z_s}{T^2 s^2 + 2\xi T s + 1}, \tag{1.31}$$

kde T je časová konstanta a

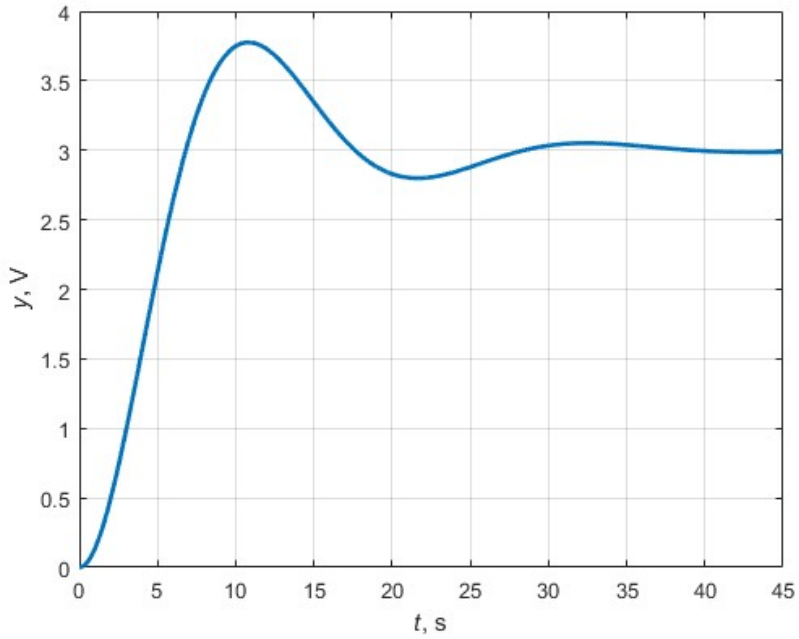
ξ – relativní tlumení.

Pro $\xi = 1$ se jedná o soustavu druhého řadu s dvojnásobnou časovou konstantou. Jedná se o dvě stejné soustavy prvního řádu zapojené za sebou (Honc, 2018b).



Obr. 1.8 – Přechodová char. soustavy druhého řadu s dvojnásobnou časovou konstantou

Pro $\xi > 1$ systém druhého řádu nekmitá. Jeho přenos obsahuje dvě různé časové konstanty a pro $\xi \epsilon (0,1)$ se jedná o kmitavý systém druhého řadu. Pro $\xi < 0$ se jedná o nestabilní systém.

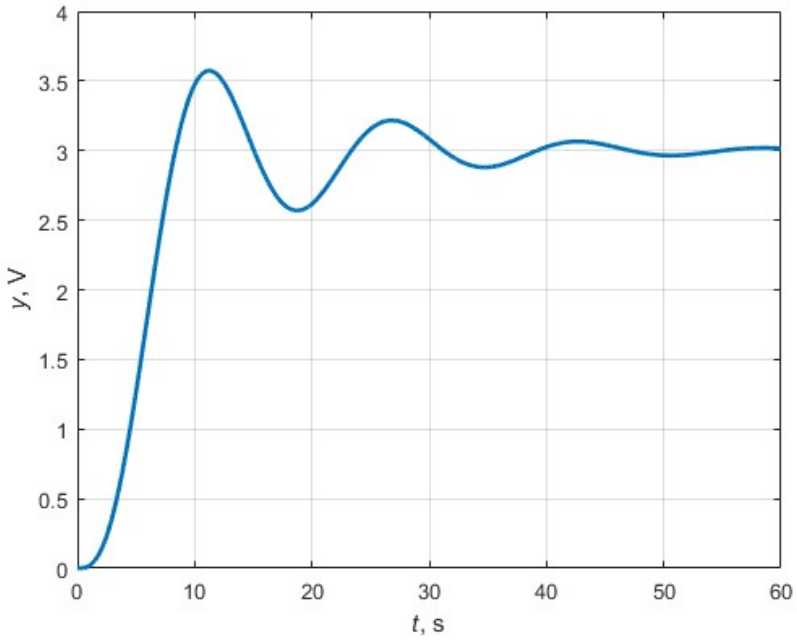


Obr. 1.9 – Přechodová char. systému druhého řádu s komplexně sdruženými kořeny

Soustavy vyššího řádu

Většina systémů v reálném provozu je ovšem vyššího řadu, který se musí poté aproximovat některou z výše uvedených soustav nižšího řádu. Obecný přenos soustavy vyššího řádu

$$F(s) = \frac{Z_s}{a_n s^n + a_{n-1} s^{n-1} + \cdots + a_0}. \tag{1.32}$$



Obr. 1.10 – Přechodová charakteristika vyššího řádu

Všechny soustavy vyššího řádu než prvního, se vyznačují existencí inflexního bodu, kde se charakteristika mění z konvexní na konkávní.

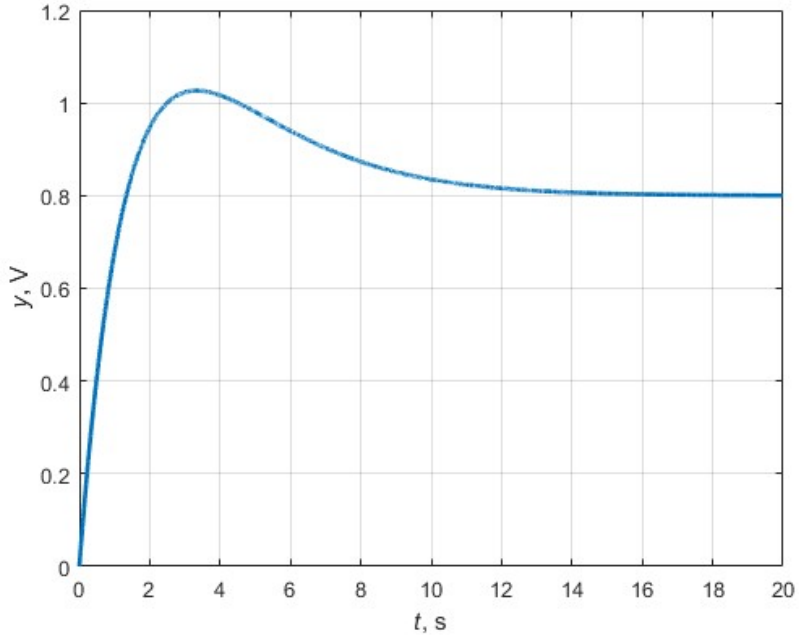
Soustavy s minimální a neminimální fází

Pokud v čitateli přenosu není prostý člen *Zs*, ale polynom vyššího stupně, vznikají systémy s minimální a neminimální fází

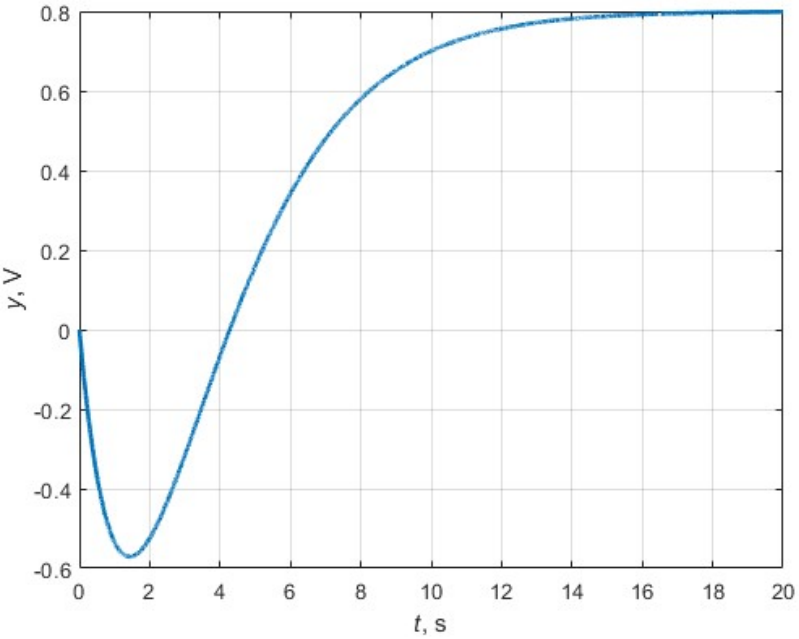
$$F(s) = \frac{\pm b_1 s + b_0}{a_2 s^2 + a_1 s + a_0} \tag{1.33}$$

Pro kladný parametr b_1 se jedná o systém s minimální fází a pro záporný s neminimální fází. Tyto systémy jsou relativního prvního řádu, ale nechovají se jako klasická soustava

prvního řádu. Systém s minimální fází překmitne, naopak systém s neminimální fází podkmitne pod časovou osu. Toto chování je způsobeno nenulovou derivací v čase 0 (Dorf, 1998).



Obr. 1.11 – Přechodová charakteristika systému s minimální fází



Obr. 1.12 – Přechodová charakteristika systému s neminimální fází

Stochastické modely

Stochastické modely mají vstup nebo výstup ovlivněny náhodnou poruchou, nejčastěji bílým šumem. Bílý šum je náhodný signál s nulovým průměrem. Pro popis těchto modelů se využívají diferenční rovnice. Mezi tyto modely patří ARMA, ARX, OE, ARMAX a již výše zmíněný CARMA (Kupka, 2018).

ARMA je model poruchy, kde proces y_r je reprezentovaný bílým šumem, který prochází do procesu přes lineární systém. Proces lze popsat rovnicí

$$y_r = \frac{C(z^{-1})}{A(z^{-1})} r(k), \tag{1.34}$$

kde *C* a *D* jsou polynomy ntého stupně a

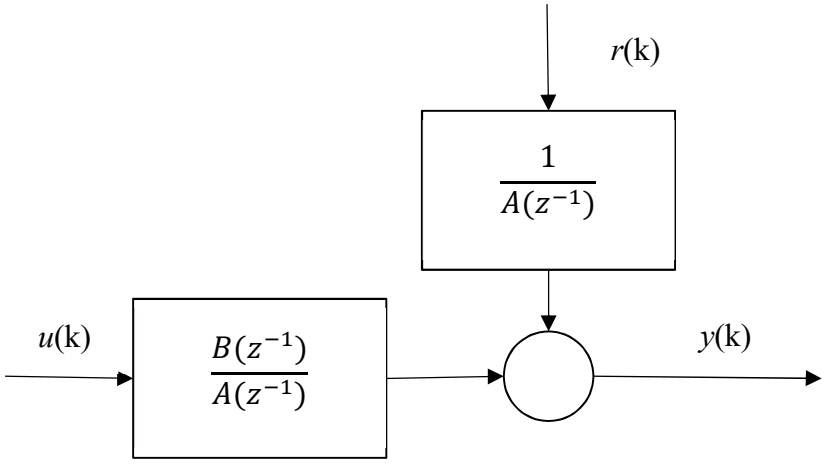
r – náhodná složka (bílý šum).

Pokud stupeň polynomu *C* je nulový, poté se z ARMA modelu stává AR (auto regressive) model, a pokud stupeň polynomu D je nulový, poté se z modelu stává MA (moving avarage) model.

ARX je modelem systému, do kterého vstupuje model poruchy skládající se z části AR a vstupu X, který představuje akční veličinu. Model lze popsat rovnicí

$$y(k) = \frac{B(z^{-1})}{A(z^{-1})} u(k) + \frac{1}{A(z^{-1})} r(k). \tag{1.35}$$

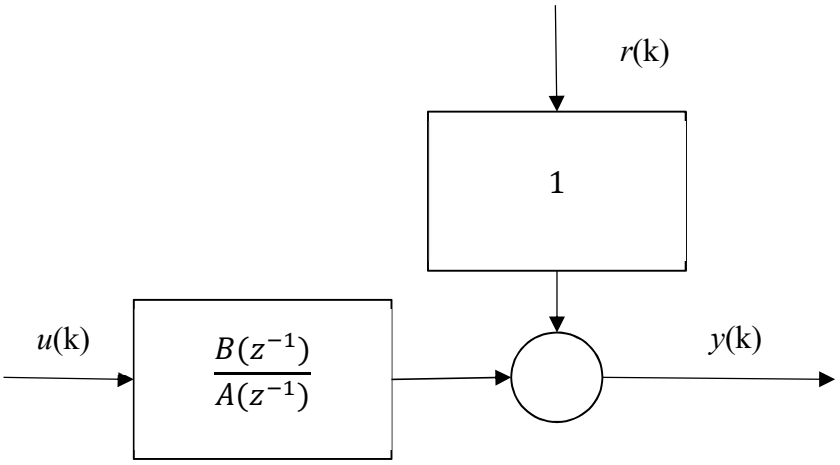
Tento model je nejpoužívanější při identifikaci systémů (Kupka, 2018).



Obr. 1.13 – Blokové schéma ARX modelu

Output Error (OE), u tohoto modelu je předpoklad, že k výstupu se přičítá náhodná složka *r*

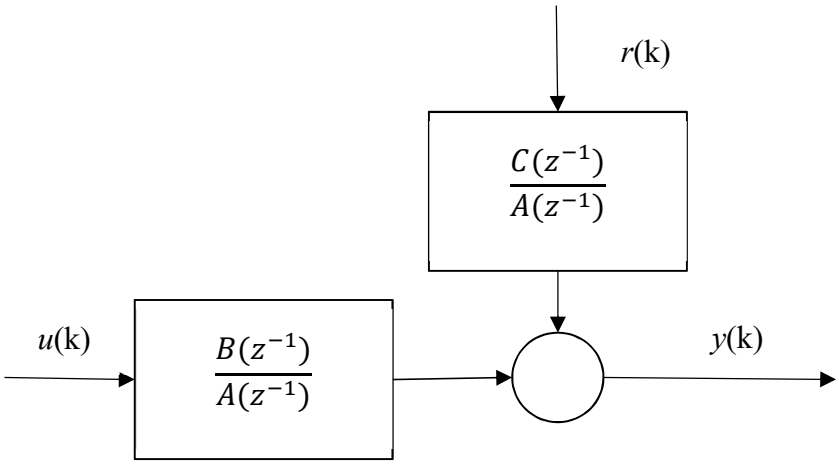
$$y(k) = \frac{B(z^{-1})}{A(z^{-1})}u(k) + r(k). \tag{1.36}$$



Obr. 1.14 – Blokové schéma OE modelu

Model ARMAX je modelem systému a modelem poruchy ve tvaru ARMA, který je popsán rovnicí (Kupka, 2018)

$$y(k) = \frac{B(z^{-1})}{A(z^{-1})}u(k) + \frac{C(z^{-1})}{A(z^{-1})}r(k). \tag{1.37}$$



Obr. 1.15 – Blokové schéma modelu ARMAX

1.2.2 Metoda nejmenších čtverců

Jak již bylo výše uvedeno, odhad parametrů soustavy pomocí metody nejmenších čtverců (MNČ) je založen na minimalizaci čtverců odchylek mezi odhadovaným výstupem a měřeným výstupem. Existují dvě varianty metody nejmenších čtverců. Metoda offline, kdy se provádí odhad parametrů až po naměření všech hodnot a metoda online, při které se provádí odhad parametrů v každém kroku měření. Kritérium MNČ je rovnice (1.28), která se dá zapsat maticově (Balátě, 2004)

$$K = (\tilde{\boldsymbol{y}} - \boldsymbol{y})^{\mathrm{T}}(\tilde{\boldsymbol{y}} - \boldsymbol{y}). \tag{1.38}$$

Při uvažování ARX modelu (1.35) upraveném do tvaru

$$Ay(k) = Bu(k) + r(k).$$

Je možné diferenční rovnici rozepsat do tvaru

$$y(k) + a_1 y(k-1) + \cdots + a_{na} y(k-na) = b_1 u(k-1) + \cdots + b_{nb} u(k-nb) + r(k), \tag{1.39}$$

kde $a_1, \ldots, a_n$ a $b_1, \ldots, b_n$ jsou parametry modelu,

na – řád polynomu *A,*

nb – řád polynomu *B* a

r(*k*) – bílý šum.

Rovnici (1.39) lze upravit do tvaru

$$y(k) = -a_1 y(k-1) - \cdots - a_{na} y(k-na) + b_1 u(k-1) + \cdots + b_{nb} u(k-nb) + r(k).$$

Rovnici (1.39) je možné zapsat rozepsat pro *k*=n, ..., *N*

$$y(n) = -a_1 y(n-1) - \cdots - a_n y(n-na) + b_1 u(n-1) + \cdots + b_{nb} u(n-nb) + r(n),$$

$$y(n+1) = -a_1 y(n) - \cdots - a_n y(n+1-na) + b_1 u(n) + \cdots + b_{nb} u(n+1-nb) + r(n+1),$$

$$y(N) = -a_1 y(N-1) - \cdots - a_n y(N-na) + b_1 u(N-1) + \cdots + b_{nb} u(N-nb) + r(N).$$

Tento zápis maticově

$$\begin{bmatrix} \mathrm{y(n)} \\ \mathrm{y(n+1)} \\ \vdots \\ \mathrm{y(N)} \end{bmatrix} =$$

$$\begin{bmatrix} -\mathrm{y(n-1)} & \cdots & -\mathrm{y(n-na)} & \mathrm{u(n-1)} & \cdots & \mathrm{u(n-nb)} \\ -\mathrm{y(n)} & \cdots & -\mathrm{y(n+1-na)} & \mathrm{u(n)} & \cdots & \mathrm{u(n+1-nb)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -\mathrm{y(N-1)} & \cdots & -\mathrm{y(N-na)} & \mathrm{u(N-1)} & \cdots & \mathrm{u(N-nb)} \end{bmatrix} \cdot \tag{1.40}$$

$$\begin{bmatrix} \mathrm{a_1} \\ \vdots \\ \mathrm{a_{na}} \\ \mathrm{b_1} \\ \vdots \\ \mathrm{b_{nb}} \end{bmatrix} + \begin{bmatrix} \mathrm{r(n)} \\ \mathrm{r(n+1)} \\ \vdots \\ \vdots \\ \mathrm{r(N)} \end{bmatrix}.$$

Matice a vektory jsou označeny jako

$$\tilde{\boldsymbol{y}} = \boldsymbol{F}\boldsymbol{x} + \boldsymbol{r} \tag{1.41}$$

kde $\tilde{\boldsymbol{y}}$ je vektorem odhadů výstupu,

$\boldsymbol{F}$ – matice minulých vstupů a výstupů,

$\boldsymbol{x}$ – vektor parametrů rovnice a

$\boldsymbol{r}$ – vektor poruch (bílého šumu).

Hodnoty $\tilde{y}$ jsou vždy zatíženy některým druhem šumu r. Rovnice (1.41), při uvažování $\boldsymbol{r} = 0$, je dosazena do (1.38), poté

$$K = (\boldsymbol{F}\boldsymbol{x} - \boldsymbol{y})^{\mathrm{T}}(\boldsymbol{F}\boldsymbol{x} - \boldsymbol{y}) = \boldsymbol{x}^{\mathrm{T}} \underbrace{\boldsymbol{F}^{\mathrm{T}}\boldsymbol{F}}_{\boldsymbol{H}} \boldsymbol{x} - \boldsymbol{x}^{\mathrm{T}}\boldsymbol{F}^{\mathrm{T}}\boldsymbol{y} \underbrace{-\boldsymbol{y}^{\mathrm{T}}\boldsymbol{F}}_{\boldsymbol{g}} \boldsymbol{x} + \underbrace{\boldsymbol{y}^{\mathrm{T}}\boldsymbol{y}}_{\boldsymbol{k}},$$

$$K = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x} + \boldsymbol{2}\boldsymbol{g}^{\mathrm{T}}\boldsymbol{x} + k. \tag{1.42}$$

Je hledáno minimum rovnice (1.42), které je rovno

$$\mathbf{x} = -\boldsymbol{H}^{-1}\boldsymbol{g} = (\boldsymbol{F}^{\mathrm{T}}\boldsymbol{F})^{-1}\boldsymbol{F}^{\mathrm{T}}\boldsymbol{y}. \tag{1.43}$$

Při $\boldsymbol{r} = 0$ je vektor $\boldsymbol{x}$ vektorem nevychýlených odhadů, středních hodnot (Dušek, 2017).

Rekurzivní metoda nejmenších čtverců

Základní myšlenkou rekurzivní metody nejmenších čtverců, v některých literaturách označována jako online MNČ, je výpočet odhadů parametrů vektoru $\boldsymbol{x}$ v každém kroku měření. To znamená, že odhad v čase k je vypočítán pomocí odhadu vypočítaném v čase $k - 1$ atd. Tato metoda výpočtu parametrů je často používaná pro odhad modelů a zpřesňování parametrů

modelu v každém kroku pro výpočet aktuální volné a vázané odezvy v prediktivním řízení. Online MNČ je také méně náročná na paměť, protože nevyžaduje uložení všech naměřených údajů (Honc, 2018b).

V kroku k je možné rovnici vektoru odhadů parametrů (1.43) napsat jako

$$\boldsymbol{x} = \left(\boldsymbol{F}(k)^{\mathrm{T}}\boldsymbol{F}(k)\right)^{-1}\boldsymbol{F}(k)^{\mathrm{T}}\boldsymbol{y}(k),$$

$$\boldsymbol{P}(k) = \left(\boldsymbol{F}(k)^{\mathrm{T}}\boldsymbol{F}(k)\right)^{-1}. \tag{1.44}$$

Rovnice (1.44) pro krok $k + 1$

$$\boldsymbol{P}(k+1) = \left(\boldsymbol{F}(k+1)^{\mathrm{T}}\boldsymbol{F}(k+1)\right)^{-1} = (\boldsymbol{F}(k)^{\mathrm{T}}\boldsymbol{F}(k) + \boldsymbol{f}(k+1)^{\mathrm{T}}\boldsymbol{f}(k+1))^{-1},$$

kde $\boldsymbol{f}(\mathrm{k}+1)$ je vektorem následujících vstupů a výstupů zařazených za sebou. Tento vektor je dalším řádkem matice $\boldsymbol{F}$ v (1.40).

Dále lze dosadit

$$\boldsymbol{P}(k+1) = (\boldsymbol{P}(k)^{-1} + \boldsymbol{f}(k+1)^{\mathrm{T}}\boldsymbol{f}(k+1))^{-1}. \tag{1.45}$$

Pro výraz $\boldsymbol{F}^{\mathrm{T}}\boldsymbol{y}$, který je označen písmenem $\boldsymbol{B}$, je možné použít stejné pravidlo

$$\boldsymbol{B}(k+1) = \boldsymbol{F}(k+1)^{\mathrm{T}}\boldsymbol{y}(k+1) = \boldsymbol{F}(k)^{\mathrm{T}}\boldsymbol{y}(k) + \boldsymbol{f}(k+1)^{\mathrm{T}}y(k+1). \tag{1.46}$$

Protože výrazy $k+1$ jsou při každém měření bez použití prediktoru neznámé a využívání prediktoru nemá význam, jsou rovnice (1.45) a (1.46) posunuty zpět do aktuálního kroku k

$$\begin{aligned}\boldsymbol{P}(k) &= (\boldsymbol{P}(k-1)^{-1} + \boldsymbol{f}(k)^{\mathrm{T}}\boldsymbol{f}(k))^{-1},\\ \boldsymbol{B}(k)^{\mathrm{T}} &= \boldsymbol{F}(k-1)^{\mathrm{T}}\boldsymbol{y}(k-1) + \boldsymbol{f}(k)^{\mathrm{T}}y(k).\end{aligned} \tag{1.47}$$

Rovnice (1.47) jsou vstupním bodem pro další rozšíření. Je použita substituce

$$(\boldsymbol{A} + \boldsymbol{B}^{\mathrm{T}}\boldsymbol{C}\boldsymbol{D})^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{B}(\boldsymbol{C}^{-1} + \boldsymbol{D}\boldsymbol{A}^{-1}\boldsymbol{B})^{-\mathbf{1}}\boldsymbol{D}\boldsymbol{A}^{-1},$$

kde $\boldsymbol{A} = \boldsymbol{P}(k-1)^{-1}$,

$\boldsymbol{B} = \boldsymbol{f}(k)^{\mathrm{T}}$,

$\boldsymbol{C} = 1$ a

$\boldsymbol{D} = \boldsymbol{f}(k)$.

Poté lze první z rovnic (1.47) rozepsat následovně

$$\boldsymbol{P}(k) = (\boldsymbol{P}(k-1)^{-1} + \boldsymbol{f}(k)^{\mathrm{T}}\boldsymbol{f}(k))^{-1} = \boldsymbol{P}(k-1) - \boldsymbol{P}(k-1)\boldsymbol{f}(k)^{\mathrm{T}}[1 + \boldsymbol{f}(k)\boldsymbol{P}(k-1)\boldsymbol{f}(k)^{\mathrm{T}}]^{-1}\boldsymbol{f}(k)\boldsymbol{P}(k-1), \tag{1.48}$$

Pro zkrácení výrazu je použita další substituce

$$\boldsymbol{m_k} = \boldsymbol{P}(k-1)\boldsymbol{f}(k)^{\mathrm{T}}[1 + \boldsymbol{f}(k)\boldsymbol{P}(k-1)\boldsymbol{f}(k)^{\mathrm{T}}]^{-1}. \tag{1.49}$$

Do rovnice (1.43) je dosazeno (1.47), (1.48) a (1.49)

$$\boldsymbol{x}(k) = \boldsymbol{P}(k)\boldsymbol{B}(k) = \Big(\boldsymbol{P}(k-1) - \boldsymbol{m_k}\boldsymbol{f}(k)\boldsymbol{P}(k-1)\Big)\Big(\boldsymbol{F}(k-1)^{\mathrm{T}}\boldsymbol{y}(k-1) + \boldsymbol{f}(k)^{\mathrm{T}}y(k)\Big) = \underbrace{\boldsymbol{P}(k-1)\boldsymbol{F}(k-1)^{\mathrm{T}}\boldsymbol{y}(k-1)}_{\boldsymbol{x}(k-1)} + \boldsymbol{P}(k-1)\boldsymbol{f}(k)^{\mathrm{T}}y(k) - \boldsymbol{m_k}\boldsymbol{f}(k)\underbrace{\boldsymbol{P}(k-1)\boldsymbol{F}(k-1)^{\mathrm{T}}\boldsymbol{y}(k-1)}_{\boldsymbol{x}(k-1)} - \boldsymbol{m_k}\boldsymbol{f}(k)\boldsymbol{P}(k-1)\boldsymbol{f}(k)^{\mathrm{T}}y(k),$$

$$\boldsymbol{x}(k) = \boldsymbol{x}(k-1) - \boldsymbol{m_k}\boldsymbol{f}(k)\boldsymbol{x}(k-1) + \boldsymbol{P}(k-1)\boldsymbol{f}(k)^{T}y(k) - \boldsymbol{m_k}\boldsymbol{f}(k)\boldsymbol{P}(k-1)\boldsymbol{f}(k)^{T}y(k).$$

Výraz $\boldsymbol{P}(k-1)\boldsymbol{f}(\mathrm{k})^{\mathrm{T}}y(k)$ lze rozšířit o jednotkovou matici pomocí $\boldsymbol{m_k}\boldsymbol{m_k}^{-1}$, pak

$$\boldsymbol{x}(k) = \boldsymbol{x}(k-1) - \boldsymbol{m_k}\boldsymbol{f}(k)\boldsymbol{x}(k-1) + \boldsymbol{m_k}\boldsymbol{m_k}^{-1}\boldsymbol{P}(k-1)\boldsymbol{f}(k)^{T}y(k) - \boldsymbol{m_k}\boldsymbol{f}(k)\boldsymbol{P}(k-1)\boldsymbol{f}(k)^{T}y(k) = \boldsymbol{x}(k-1) - \boldsymbol{m_k}\boldsymbol{f}(k)\boldsymbol{x}(k-1) + \boldsymbol{m_k}[\boldsymbol{m_k}^{-1}\boldsymbol{P}(k-1)\boldsymbol{f}(k)^{T} - \boldsymbol{f}(k)\boldsymbol{P}(k-1)\boldsymbol{f}(k)^{T}]\,y(k).$$

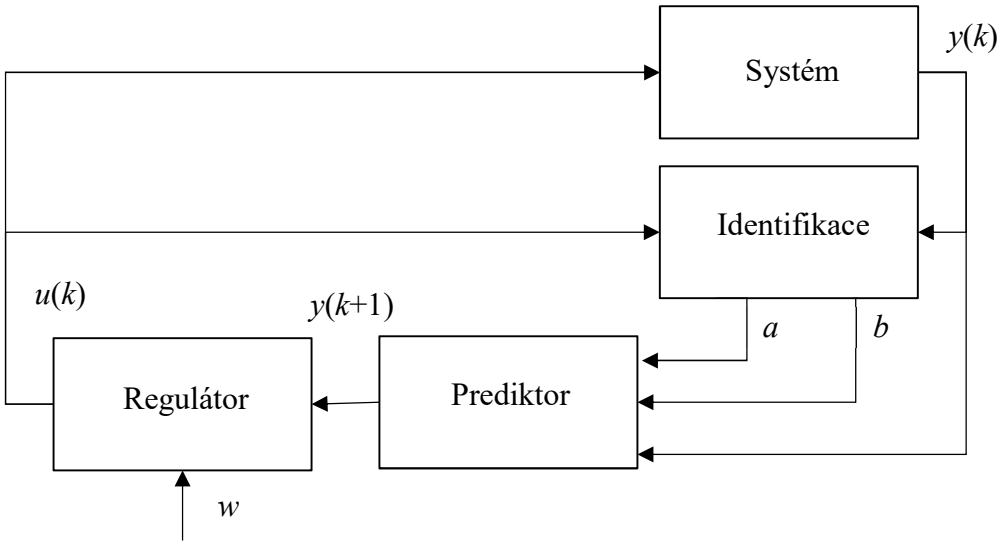
Kde po dosazení z (1.49) vznikne

$$\boldsymbol{x}(k) = \boldsymbol{x}(k-1) - \boldsymbol{m_k}\boldsymbol{f}(k)\boldsymbol{x}(k-1) + \boldsymbol{m_k}\,y(k),$$

$$\mathbf{x}(k) = \boldsymbol{x}(k-1) + \boldsymbol{m_k}(y(k) - \boldsymbol{f}(k)\boldsymbol{x}(k-1)). \tag{1.50}$$

Pro aktuální výpočet vektoru odhadů je použita rovnice (1.50). V každém kroku měření je naplněn vektor $\boldsymbol{f}$ aktuálními hodnotami *y* a *u*, dále jsou vypočteny rovnice (1.48) a (1.50), kde rovnice (1.50) obsahuje aktuální odhady parametrů systému (Honc, 2018b).

Ideální řízení pomocí prediktivního regulátoru obsahuje identifikační modul, který využívá online metodu nejmenších čtverců, pro určení co nejpřesnějších parametrů modelu, který je využit k výpočtu prediktoru a regulačního zákonu. Tento proces je pomocí bloků znázorněn na obr. 1.16 (Aström, 1995).



Obr. 1.16 – Blokové schéma prediktivního řízení

1.3 Jednoduchý řídicí systém

Pro otestování vyvinutých knihoven je použit vývojový kit Arduino UNO, který je možné zařadit mezi jednoduché mikroprocesory. „Jednoduchý“ mikroprocesor je v této diplomové práci označení pro mikroprocesor, který nemá vlastní operační systém a zpracovává program nahraný ve FLASH paměti. Má jeden procesor s jedním vláknem, je tedy bez možnosti více vláknového programování. Arduino UNO je osazeno mikroprocesorem ATMEGA328P.

Obr. 1.17 – Arduino UNO

Vývojová deska osazená mikroprocesorem ATmega328P, obr. 1.17, má 14 digitálních vstupů, které se programově dají změnit na výstupy, 6 analogových vstupů a 16 MHz krystal. Deska je také osazena všemi nutnými prostředky k jednoduchému připojení k PC pomocí USB.

Pracovní napětí Arduino Uno je 5 V, z 14 digitálních vstupů a výstupů má 6, které jsou schopny pulzně šířkové modulace. Čtyři pracují s maximální frekvencí 490 Hz a dva pracují s frekvencí 980 Hz. Analogové vstupy mají 10 bitové rozlišení, tudíž maximální schopnost rozlišení u 5 V napětí je 0,005 V. Mikroprocesor obsahuje 32 KB FLASH paměti, 2 KB SRAM paměti pro ukládání mezi výpočtů a 1 KB EEPROM paměti, která drží data i v případě vypnutí napájení desky. Arduino lze programovat v jazyce C a C++. Autoři této desky vyvinuli knihovny ulehčující nastavení desky a portů, díky široké komunitě lidí, která se aktivně podílí na vývoji. Je možné využít široké spektrum knihoven pro různé aplikace (Voda, 2015).

2 PRAKTICKÁ ČÁST

Praktická část se věnuje implementaci výše uvedených metod řízení a identifikaci. V rámci praktické časti práce jsou vytvořeny čtyři knihovny napsané v jazyce C++. Všechny knihovny jsou testovány na vývojovém programátoru Arduino UNO. Kvůli využití Arduino UNO musel být brán ohled na malou paměť SRAM, nelze alokovat velké matice a vektory s velkým horizontem řízení. Arduino UNO používá několik datových typů, ale pro potřeby těchto knihoven budou stačit datový typ float, který zabírá v paměti 4 B a datový typ int, který zabírá 2 B. Maximální počet proměnných je cca 400 s programem. Tohoto čísla lze velmi lehko dosáhnout používáním dvou rozměrných polí. Při programovaní muselo být zohledněno toto omezení.

```
void  initialize(parametry se liší);
void  showControllerData();
void  setSetpoint(float w, long current_time, long
start_time,long end_time);
void  printData(long current_time);
float  process(float last_sample);
```

Obr. 2.1 – Jednotné rozhraní knihoven

Pro knihovny s regulátory a identifikaci je vytvořeno jednotné rozhraní, které je využito v každé knihovně. Toto rozhraní pomůže s orientací v kódu v případě rozšíření knihoven. Knihovny obsahují prediktivní regulátor, odvozený prediktivní regulátor pro soustavy prvního řádu s dopravním zpožděním, manuální řízení, které obsahuje online MNČ a pro srovnání s prediktivním řízením poslední knihovna obsahuje PID regulátor v přírůstkovém tvaru. Funkce PID regulátoru zde není popsána, jsou zde uvedeny pouze výsledky měření. Jednotlivé kapitoly praktické části se věnují popisu metod tohoto rozhraní. Metody rozhraní a jejich návratové typy jsou zobrazené na obr. 2.1.

Rozhraní obsahuje pět různých metod, které se nazývají stejně pro každý typ regulátoru a ovládání. Metodu *initialize*, která se volá vždy při nastavování mikroprocesoru, protože nelze nastavovat některé parametry v čase volání konstruktoru knihovny. Metoda *initialize* nastavuje parametry regulátoru při běhu programu. Metoda *showControllerData*, která vypíše po sériové lince nastavené parametry regulátoru. Metodu *setSetpoint*, která nastavuje žádanou hodnotu do budoucnosti. Metoda *printData* vypisuje data řízení, nebo regulace po sériové lince a hlavní metoda *process*, která vykonává regulaci a vrací zpátky hodnotu, která se má nastavit na výstup.

Manuální řízení obsahuje ještě speciální metodu *identifyPlant*, která využívá online metody nejmenších čtverců pro identifikování soustavy.

Součástí knihoven je i ilustrační kód pro Arduino UNO. Dále je využito knihovny pro maticové operace, zjištění volné paměti v mikroprocesoru a autorem práce vytvořený převodník z analogového signálu o 1024 hodnotách na 5 voltů a z 5 voltů na 255 digitálních hodnot pro PWM signál. Ilustrační kód pro knihovny a převodník je připojen do přílohy.

Metody *showControllerData*, *setSetpoint* a *printData* jsou velmi podobné pro každý typ regulátoru.

Metoda showControllerData

Tato metoda je volána vždy po inicializaci regulátoru, protože vypíše po sériové lince parametry regulátoru. V případě prediktivního regulátoru vypíše první hodnotu v matici ***G,*** vektor ***k***, vynásobení mezi vektorem ***k*** a maticí ***Fp*** a první řádek matice ***Fp***. Prediktivní regulátor pro soustavy prvního řádu s dopravním zpožděním zase vypíše vypočtené parametry l_{y1}, l_{y2}, l_{y3} a zadanou soustavu.

```
G: 0.20
K: 0.11 0.23 0.32 0.38
Kfp: -2.67 1.75 -0.10 -0.17 1.80   a: -0.69 b: 0.31 d: 1 ZS: 0.99
Fp: 1.77 -0.81 0.05 0.08 -0.80     l1: -1.93 l2: 1.02 l3: 0.91
```

Obr. 2.2 – Příklad volání showControllerData

Metoda setSetpoint

```
    void PredictiveController::setSetpoint(float w, long
current_time, long start_time, long end_time) {
    if (_setPoint[0]!=w && (current_time >= start_time * 1000)
&& (current_time <= end_time * 1000)) {
        for (int i = 0; i < _N2; ++i) {
            _setPoint[i]=_setPoint[i+1];
        }
        _setPoint[_N2] = _alfa*_last_setPoint+(1-_alfa)*w;
        _last_setPoint=_setPoint[_N2];
    }
}
```

Obr. 2.3 – Implementace metody setSetpoint

Tato metoda, obr. 2.3, očekává parametry žádanou hodnotu, aktuální čas, odkdy má začít s nastavováním žádané hodnoty a do jakého času ji má nastavovat. Aby bylo možné mít několik volání této metody s jinými žádanými hodnotami, muselo být přidáno časové omezení

zadávání hodnot. Podmínka tedy vyhodnotí, jestli je správný čas pro zadání žádané hodnoty, a pokud ano, for cyklus posune hodnoty vektoru žádaných hodnot doleva a na poslední hodnotu pole nastaví žádanou hodnotu vypočítanou podle rovnice (1.2) a uloží si poslední žádanou hodnotu do paměti. Tento způsob nastavování žádané hodnoty je výhodný pouze pro prediktivní regulátor. Prediktivní regulátor pro soustavy prvního řádu s dopravním zpožděním počítá pouze s jednou žádanou hodnotou.

Metoda printData

Tato metoda vypisuje aktuální naměřené a vypočítané hodnoty po sériové lince, aby bylo možné dalšího zpracování. V prvním sloupci je aktuální čas, ve druhém je regulovaná veličina, ve třetím je akční veličina, ve čtvrtém je žádaná hodnota a v pátém sloupci je první predikovaná veličina. Při identifikaci soustavy jsou na místě čtvrtého a pátého sloupce vypisovány aktuální odhady parametrů *a* a *b*. Metoda je volána až na konci každého cyklu.

2.1 Prediktivní regulátor

Protože jazyk C++ nemá žádné nástroje pro práci s maticemi a vektory, v rámci práce jsou vymyšleny algoritmy pro naplnění matic ***Ap***, ***Bp***, ***Am***, ***Bm***, ***Cm*** tak, aby odpovídaly maticím uvedených v rovnici (1.11). Operace s maticemi a vypočítání potřebných hodnot pro regulační zákon obstarává metoda *initialize*, která je vždy volána v bloku nastavení Arduina. Metoda *proces* vypočítává regulační zákon dle rovnice (1.19) a volá se v každé periodě vzorkování.

2.1.1 Metoda initialize

Jak již bylo řečeno, všechny potřebné operace pro získání vektorů pro výpočet regulačního zákonu jsou uvnitř této metody.

```
    void initialize(float a[], float b[], float c[], int N2,
int Nu, float q,float alfa, int nA, int nB, int nC);
```

Obr. 2.4 – Definice volání metody

Dle obr. 2.4 metoda vyžaduje jako vstup parametry modelu *a* a *b*, polynom *c*, horizont žádané hodnoty N_2, horizont akční veličiny N_u, parametr penalizace akční veličiny *q*, parametr filtrování žádané hodnoty *alfa* a velikosti zadaných polí *nA*, *nB* a *nC*. Příklad volání metody je na obr. 2.5. Minimální horizont N_1 se nenastavuje a je automaticky nastaven na jedničku.

```
float a[3] = { 1.0000,   -0.7654,    0.0463};
float b[3] = {  0,    0.2030,    0.0756};
float c[2] = { 1.0000,-0.8};
int N2 = 4;
int Nu = 1;
float q = 0.8;
float alfa=0.7;
int nA = 3;
int nB = 3;
int nC = 2;
predictiveController.initialize(a, b, c, N2, Nu, q,alfa,
nA,nB, nC);
```

Obr. 2.5 – Příklad volání metody

Po úspěšném zavolání se začnou připravovat vektory pro vytvoření matic. Nejprve se provede vynásobení polynomu polynomem dle rovnice (1.10). Výsledkem je polynom $\tilde{A}$. Algoritmus pro výpočet součinu polynomů je poměrně jednoduchý, je využito součinu každého členu prvního polynomu s každým členem druhého polynomu dle obr. 2.6.

```
for (int i = 0; i < nA; ++i) {
     for (int j = 0; j < 2; ++j) {
          mat[i + j] = mat[i + j] + a[i] * delta[j];
     }
}
```

Obr. 2.6 – Algoritmus pro výpočet součinu polynomů

Dále se pokračuje plněním matic podle rovnice (1.11). Matice ***Ap*** obsahuje všechny prvky výsledku součinu polynomů, kde na hlavní diagonále je první prvek, tedy 1 a další prvky jsou pod hlavní diagonálou. Algoritmus plnění je na obr. 2.7. Pomocí dvou for cyklů se prochází matice ***Ap*** a do jednotlivých sloupců se zapisuje řádek výsledku součinu polynomů. Hlavní diagonála je při $i = j$. Lze napsat přístup do paměti matice tak, aby zápis řádku při každé další iteraci byl posunutý o jedničku. První podmínka chrání zápis čísel do jiných paměťových polí, než je matice ***Ap*** a druhá chrání před přístupem do paměti jinam, než je výsledek součinu polynomů.

```
for (int i = 0; i < N2; ++i) {
      for (int j = 0; j < N2; ++j) {
            if ((i + j) >= N2) {
                  break;
            }
            if (j < (nA + 1)) {
                  Ap[i + j][i] = mat[j];
            }
      }
}
```

Obr. 2.7 – Algoritmus plnění matice ***Ap***

Pro matici ***Bp*** je algoritmus velice podobný, akorát se matice plní pomocí vektoru ***b***.

U matic ***Am***, ***Bm*** a ***Cm*** je situace odlišná, prvky jsou totiž pouze nad vedlejší diagonálou. Byl použit algoritmus uvedený na obr. 2.8. I zde je použit for cyklus, který prochází všemi hodnotami matice po řádcích a do řádků vkládá hodnoty vektorů tak, aby při každé další iteraci byl zápis posunutý o jedničku doprava. Podmínka chrání před přístupem do jiné paměťové buňky, než je paměť alokovaná vektorem. Matice ***Bm*** je plněna podobným způsobem, ale je využit vektor ***b*** a je plněna pouze v případě, pokud je řád modelu soustavy vyšší než jedna. Matice ***Cm*** je plněna vektorem ***c*** a je plněna pouze pokud vektor ***c*** má velikost 2 a vyšší.

```
for (int i = 0; i < N2; ++i) {
      for (int j = 0; j < nA; ++j) {
            if ((i + j) >= (nA + 1)) {
                  break;
            }
            Am[i][j] = -mat[j + i + 1];
      }
}
```

Obr. 2.8 – Algoritmus plnění matice ***Am***

Nakonec přípravy matic je vytvořena matice ***Fpo***, která obsahuje všechny prvky matic ***Am***, ***Bm*** a ***Cm***.

Dále je využito knihoven Matrix Math, které obsahují základní operace pro práci s maticemi. Každá maticová operace, kromě inverze, vyžaduje vytvoření nové matice, což způsobuje největší problémy s pamětí, ale po testování kódu je dospěno k omezení neboli k hodnotám horizontu žádané hodnoty a horizontu řízení, aby nedocházelo k žádným nepředvídatelným událostem při plné paměti Arduina. Maximální hodnota horizontu žádané hodnoty je při druhém řádu modelu soustavy nastaven na šest, při horizontu akční veličiny nastavené na jedna. Pokud horizont žádané hodnoty je nastavený na čtyři, lze horizont akční veličiny nastavit také na čtyři. Při vyšším řádu soustavy se tyto hodnoty zmenšují a naopak. Při

prvním řádu modelu soustavy je maximální horizont žádané veličiny sedm, při horizontu akční veličiny nastaveném na jedničku. Při nastavení větších hodnot dochází k nepředvídatelným jevům s pamětí Arduina a program se nevykonává správně.

```
      Matrix.Invert((mtx_type*) Ap, N2);
      Matrix.Multiply((mtx_type*) Ap, (mtx_type*) Bp, N2, N2,
N2, (mtx_type*) Go);
      Matrix.Transpose((mtx_type*) G, N2, Nu, (mtx_type*) Gt);
      Matrix.Add((mtx_type*) GtRG, (mtx_type*) Q, Nu, Nu,
(mtx_type*) GtRGQ);
      Matrix.Print((mtx_type*)L, Nu, N2, "L")
```

Obr. 2.9 – Příklad použitých metod knihovny Matrix Math

Knihovna Matrix math je pod licencí GPL2, je tedy možné ji svobodně využít. Metody, které jsou využívány, jsou zobrazeny na obr. 2.9. Metoda pro inverzi matice využívá Gauss-Jordanovu eliminační metodu s částečným pivotováním. V rámci řešení praktické části práce je otestováno mnoho různých knihoven pro výpočet inverzí matice, ale žádná z nich nebyla tak stabilní a přesná ve výpočtu iverzní matice, jako tato.

Konec metody *initialize* obsahuje uložení do paměti potřebných vektorů pro výpočet regulačního zákonu. Jedná se o první řádek matice ***L***, první řádek matice ***Fp***, první hodnotu v prvním sloupci matice ***G*** a první řádek výsledku násobení matic ***L*** a ***Fp***. Ostatní proměnné jsou vymazány a paměť je uvolněna pro další výpočty. Průměrný čas zpracování metody *initialize* je 60 ms.

2.1.2 Metoda process

V této metodě jsou vykonávány všechny operace potřebné k výpočtu regulačního zákonu. Jedná se o přípravu vektoru ***xp***, samotný výpočet regulačního zákonu a výpočet první predikované hodnoty. Zde již není využito žádné knihovny, operace jsou jednoduché a není potřeba využívat a mít paměť obsazenou knihovnou Matrix math.

Ve vektoru ***xp*** je vždy uložena aktuální změřená akční veličina a podle řádu soustavy jsou uloženy historické hodnoty akční veličiny. V každém kroku výpočtu regulačního zákonu se musí vektor ***xp*** posouvat doprava, do minulosti. Podobný algoritmus, který je použit v případě posouvání žádané hodnoty, lze využít i zde, algoritmus je zobrazen na obr. 2.10.

```
for (int i = _nA; i > 0; --i) {
      _Xp[i] = _Xp[i - 1];
}
      _Xp[0] = last_sample;
```

Obr. 2.10 – Algoritmus posouvání hodnot do minulosti

Algoritmus se liší akorát ve směru přesouvání hodnot a místo na poslední prvek pole je poslední naměřená akční veličina uložena na první pozici pole. Další hodnoty jsou do vektoru ***xp*** vloženy pouze v případě, že řád modelu soustavy je vyšší než 1 a pokud je řád filtračního polynomu vyšší nebo roven 1. Stejným způsobem dojde k posunutí doleva a naplnění hodnotami. I zde je omezení kvůli ušetření paměti. Maximální řád modelu soustavy je tři a maximální řád filtračního polynomu je dva.

Dále zde proběhne násobení vektorů, dle obr. 2.11, a pokračuje se výpočtem regulačního zákonu a prediktoru.

```
for (int i = 0; i < _nA + _nB + _nC - 3; ++i) {
     KFpXp += _Kfp[i] * _Xp[i];
}

for (int i = 0; i < _nA + _nB + _nC - 3; ++i) {
     FpXp += _Fp[i] * _Xp[i];
}
```

Obr. 2.11 – Součin vektorů ***KFp*** s ***Xp*** a součin prvního řádku matice ***Fp*** s ***Xp***

Jelikož nejsou potřeba odhady budoucích přírůstků akčních veličin, je možné použít pouze první řádek matice ***L***, který je zde označen písmenem ***K***. Také není potřeba znát budoucí predikované hodnoty a stačí pro výpočty pouze následující predikovanou regulovanou veličinu a vzhledem k povaze matice ***G***, která obsahuje hodnoty pouze pod hlavní diagonálou, nemuselo být ukládáno do paměti víc než první hodnota matice ***G*** a první řádek matice ***Fp***. Dle obr. 2.12 následuje pouze výpočet regulačního zákonu a prediktoru, uložení do paměti potřebných hodnot pro další iteraci a vrácení hodnoty akční veličiny.

```
for (int i = 0; i < _N2; ++i) {
     deltaControllValue += _K[i] * _setPoint[i];
          }
deltaControllValue += KFpXp;
_controll_value += deltaControllValue;
_last_controll_value = deltaControllValue;
_predict_value = _G * deltaControllValue + FpXp;
return _controll_value
```

Obr. 2.12 – Výpočet regulačního zákonu a prediktoru

2.2 Prediktivní regulátor pro soustavy prvního řádu s dopravním zpožděním

Pro tento typ prediktivního regulátoru není potřeba žádných maticových operací, protože všechny potřebné parametry pro regulační zákon a prediktor lze vypočítat podle vztahů uvedených v kapitole 1.2.

2.2.1 Metoda initialize

```
void initialize(float a[], float b[], int d,float q,float
alfa, int future);
```

Obr. 2.13 – Příklad volání metody initialize

Stejně jako u prediktivního regulátoru, i zde je zapotřebí modelu soustavy, který je nastaven v polích *a* a *b*. Dalším povinným parametrem je dopravní zpoždění soustavy *d*, parametr penalizace akční veličiny *q* a filtrovací parametr žádané hodnoty *alfa*. Posledním parametrem je *future,* který způsobí posunutí vektoru žádané hodnoty o uvedenou hodnotu. Příklad volání je na obr. 2.13.

Po zavolání metody je nejprve vypočteno zesílení soustavy, které je potřebné pro výpočet regulačního zákonu a podle rovnic (1.27) jsou vypočteny všechny parametry tak, jak je ukázáno na obr. 2.14.

```
q2 = pow(q,2);
k11 = -exp(0.3598-0.9127*q+0.3165*q2);
k21 = -exp(0.0875-1.2309*q+0.5086*q2);
k31 = 1.05;
k12 = exp(-1.7383-0.40403*q);
k22 = exp(-0.32157-0.8192*q+0.3109*q2);
k32 = 1.045;

_l1 = k11-k21*_a/(k31+_a);
_l2 = k12-k22*_a/(k32+_a);
_l3 = -_l1-_l2;
```

Obr. 2.14 – Výpočet parametrů regulátoru

Hodnoty parametrů l1, l2 a l3 jsou uloženy do paměti. Průměrná doba zpracování metody je do 15 ms. Pokud je získán model soustavy před regulací, je možné tuto metodu zavolat pouze jednou, a to v čase nastavování Arduina.

2.2.2 Metoda process

Stejně tak, jako u předchozího regulátoru, i zde je počítán prediktor a regulační zákon podle vztahů uvedených v rovnicích (1.24) a (1.26).

```
_y[1]=last_sample;

for (int i = 2; i <= _d+1; ++i) {
      _y[i] = (1 - _a) * _y[i-1]  + _a * _y[i-2] + _b *
(_u[_d+1-i] - _u[_d+2-i]);
}
```

Obr. 2.15 – Výpočet prediktoru

Nejdříve je uložena poslední naměřená regulovaná veličina a podle vztahu (1.24) jsou for cyklem, jak je ukázáno na obr. 2.15., vypočteny predikované hodnoty.

Regulační zákon je vypočten podle rovnice (1.26), výpočet je na obr. 2.16. Nakonec je provedeno posunutí hodnot do minulosti, jak predikovaných, tak akčních veličin. Posunutí je provedeno stejným algoritmem jako v případě vektoru ***xp***.

```
_u[0] = _u[1] + (_l1*_y[_d+1]+_l2*_y[_d]+_l3*_setPoint[0 +
future])/_ZS;
```

Obr. 2.16 – Výpočet regulačního zákonu

První hodnota v poli *_u* je akční veličina, která se nastaví na výstup. Jak výpočet parametrů, tak výpočet regulačního zákonu s predikovanou regulační veličinou, jsou mnohonásobně snazší a rychlejší, než „plná“ implementace prediktivního regulátoru. Díky tomu je možné tento typ regulátoru použít tam, kde by byl normálně použit PID regulátor, implementační čas je totiž srovnatelný. Tento regulátor se nemusí nijak nastavovat, pouze je nutná znalost modelu soustavy, který je možný získat pomocí metody nejmenších čtverců, nebo aproximací uvedenou v kapitole 1.2.1. Pokud je možné použít některou z online metod nejmenších čtverců, lze volat metodu *initialize* v každé periodě vzorkování spolu s voláním metody *process*.

2.3 Online metoda nejmenších čtverců

Oba dva regulátory, které jsou naprogramovány, potřebují pro řízení model soustavy. V rámci práce muselo být vymyšleno a naprogramována knihovna, která měří a identifikuje zapojenou soustavu. V knihovně je využito stejného rozhraní jako v předchozích dvou knihovnách. Měřicí knihovna pouze nastavovala žádanou hodnotu na vstup soustavy, poté je

nutné veškerá data převést do programového vybavení MATLAB a pomocí skriptů získat model soustavy. Tento způsob je velmi zdlouhavý, a tak je využito znalostí z teoretické části práce online metody nejmenších čtverců, která je naimplementována do měřicí knihovny.

Online MNČ je součástí knihovny manuálního ovládání soustavy, zabírá nejméně místa v paměti mikroprocesoru, a tak zde byl prostor pro další metodu. Před voláním metody pro identifikaci je zapotřebí pomocí metody initialize nastavit žádanou hodnotu *setPoint* a odhadovaný řád soustavy *n*, dle obr. 2.17. I zde je parametr filtrování *alfa* a parametr *future*, oba parametry fungují stejně jako v předchozích případech.

```
void ManualControll::initialize(float setPoint, float
alfa,int future,int n)
```

Obr. 2.17 – Příklad volání metody initialize

2.3.1 Metoda identifyPlant

Hlavní metodou knihovny v tomto případě není metoda process, ale identifyPlant. Metoda se volá každou periodu vzorkování, volání metody je uvedeno na obr. 2.18. Metoda očekává na vstupu poslední vzorek a poslední akční veličinu, která byla poslána do soustavy.

```
void identifyPlant(float sample,float controll_value);
```

Obr. 2.18 – Volání metody identifyPlant

Je vytvořen algoritmus pro výpočet rovnice (1.48), pomocí rovnice (1.49) a následně výpočet rovnice (1.50) a plnění vektoru $\boldsymbol{f}$. Výpočty s maticemi a vektory jsou řešeny zase pomocí for cyklů. Některé výpočty potřebovaly mezi výpočty a pomocné proměnné, které jsou inicializovány na počátku metody.

```
for (int i = 0; i < n; ++i) {
      for (int j = 0; j < n; ++j) {
            mk[i] = mk[i] + _P[i][j]*_f[j];
            denom = denom + _f[j]*_P[j][i]*_f[i];
      }
}
```

Obr. 2.19 – Výpočet rovnice (1.49)

Obr. 2.19 zobrazuje algoritmus pro výpočet rovnice (1.49). Zde je zvlášť vypočítán jmenovatel a čitatel rovnice. Dalším for cyklem obr. 2.20 je dopočítán vektor $\boldsymbol{m}_k$ a připraven další mezi výpočet, který vytvoří matici $\boldsymbol{M}_{kf}$, jenž obsahuje prvky násobení vektoru $\boldsymbol{m}_k$ s vektorem $\boldsymbol{f}$. Je zde taky připraven výpočet odhadu regulované veličiny *yp*.

```
for (int i = 0; i < n; ++i) {
      yp=yp + _f[i]*_X[i];
      mk[i]=mk[i]/(1+denom);
      for (int j = 0; j < n; ++j) {
            mkf[i][j]=mk[i]*_f[j];
      }
}
```

Obr. 2.20 – Výpočet ***mk***, a mezi výpočety ***M***kf a *yp*

Následně je proveden výpočet rovnice (1.48) pomocí maticového součinu a odečtením od minulé hodnoty matice ***P***, obr. 2.22.

```
for (int i = 0; i < n; ++i) {
      for (int j = 0; j < n; ++j) {
            for (int k = 0; k < n; ++k) {
                  sum=sum +mkf[i][k]*_P[k][j];
            }
            pk[i][j]=_P[i][j]-sum;
            sum=0.0;
      }
}
```

Obr. 2.21 – Výpočet rovnice (1.48)

Nakonec je pomocí algoritmu vypočtena rovnice (1.50), obr. 2.23.

```
      for (int i = 0; i < n; ++i) {
            xk[i]=_X[i] + mk[i]*(sample-yp);
      }
```

Obr. 2.22 – Výpočet rovnice (1.50)

Dále je v metodě uložení aktuálních hodnot vektoru ***x*** a matice ***P***. Pro posunutí vektoru ***f*** do minulosti je využit stejný algoritmus, jako v případě posouvání ***xp*** a nahrání aktuálních hodnot akční a regulované veličiny na správná místa ve vektoru ***f***.

Posloupnost zpracování Online MNČ je následující:

1. Výpočet rovnice (1.49)
2. Výpočet rovnice (1.48)
3. Výpočet rovnice (1.50)
4. Naplnění vektoru ***f*** aktuálními hodnotami
5. Posunutí vektoru ***f*** do minulosti
6. Uložení vektoru $\boldsymbol{x}_k$ a matice $\boldsymbol{P}_k$ do paměti

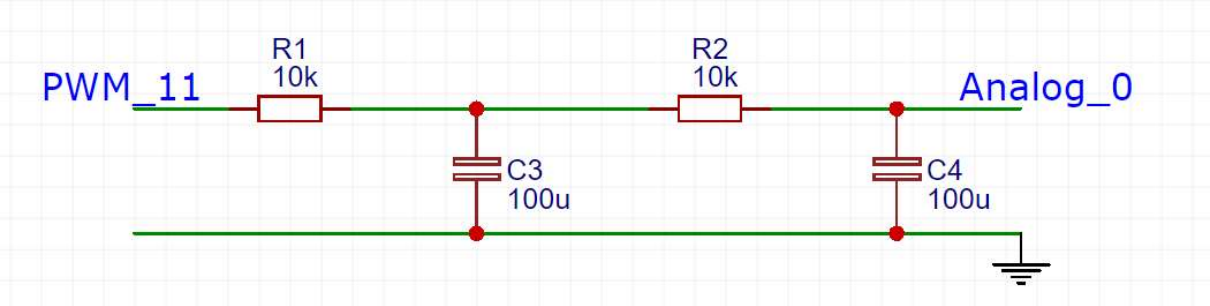
Metodu lze využít pro identifikaci soustav maximálně pátého řádu.

2.4 Reálné experimenty na soustavě s pomocí knihoven

Funkce knihoven je otestována na soustavě dvou sériově zapojených integračních článků, dle obr. 2.24, RC články se takto chovají jako soustava druhého řádu s přenosem

$$F(s) = \frac{1}{(RCs + 1)(RCs + 1)}, \tag{2.1}$$

Hodnota obou odporů a obou kapacit je totožná, kde R = 10KΩ a C = 100μF, soustava má jednu dvojnásobnou konstantu rovnou jedné.

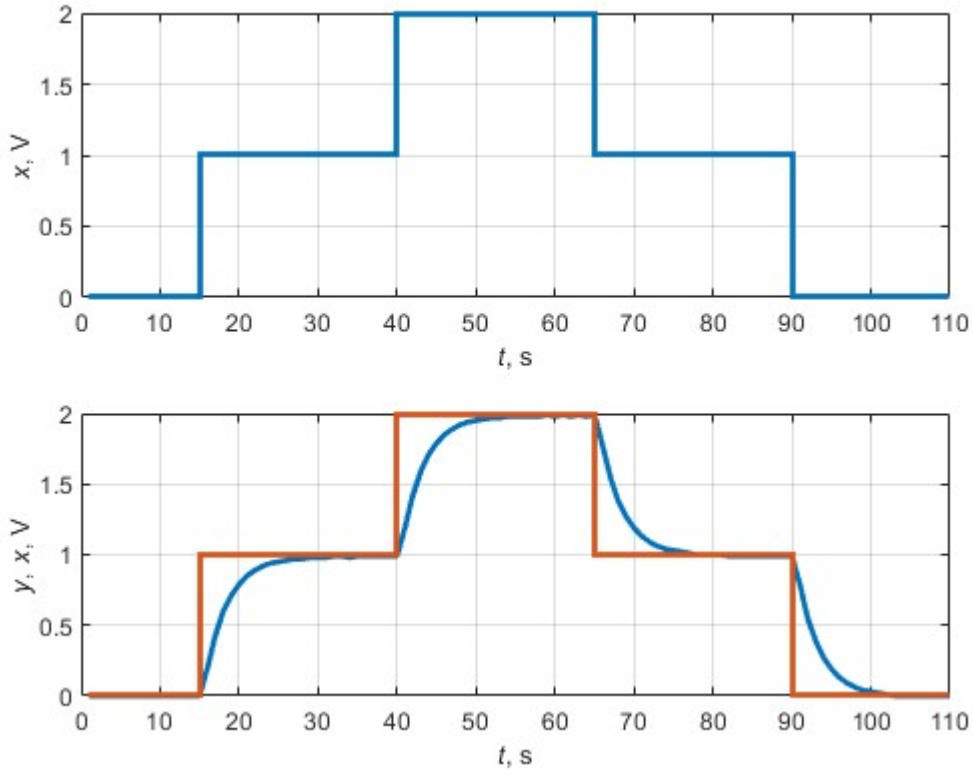


Obr. 2.23 – Zapojení soustavy dvou RC článků

Na vstup RC článku je zapojen PWM výstup Arduina UNO a na výstup soustavy je zapojen analogový vstup. Pro správnou funkci je propojena zem RC článku s Arduinem.

2.4.1 Měření a identifikace reálné soustavy

Knihovna měření je použita spolu s ukázkovým programem, který je přiložen do přílohy. Tento program je nahrán do ARDUINA. Vstupním parametrem pro identifikaci je zadána soustava druhého řádu, vzorkovací perioda je nastavena na jednu sekundu a během prvních 110 sekund je provedeno několik změn akční veličiny. Výsledky měření jsou uvedeny na obr. 2.25.



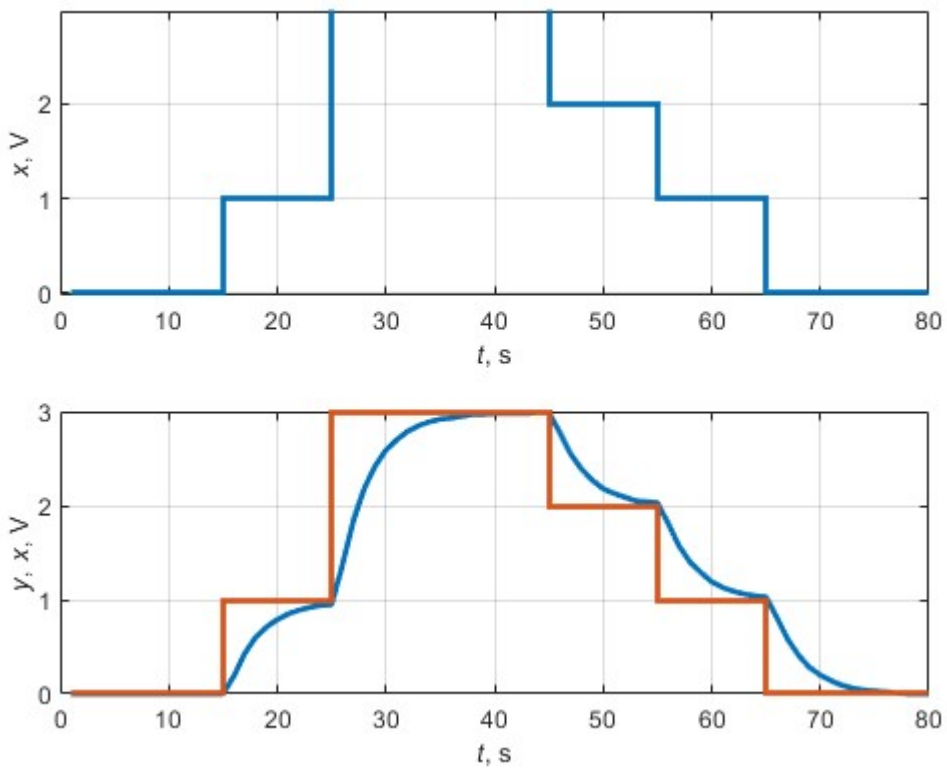
Obr. 2.24 – Měření odezvy na skokové změny akční veličiny

První graf obr. 2.25 ukazuje vstup do soustavy, druhý graf porovnává tento vstup (oranžový) s naměřeným výstupem (modrým). Stejně jsou zobrazeny všechny průběhy ovládání.

Výsledný model, který ARDUINO pomocí metody *identifyPlant* získalo, je ve tvaru

$$F(z^{-1}) = \frac{0{,}1955z^{-1} + 0{,}0890z^{-2}}{1 - 0{,}7452z^{-1} + 0{,}0321z^{-2}}, \tag{2.2}$$

Pro ověření modelu jsou provedeny další dva experimenty se soustavou. Výsledky měření jsou uvedeny na obr. 2.26 a obr. 2.27.

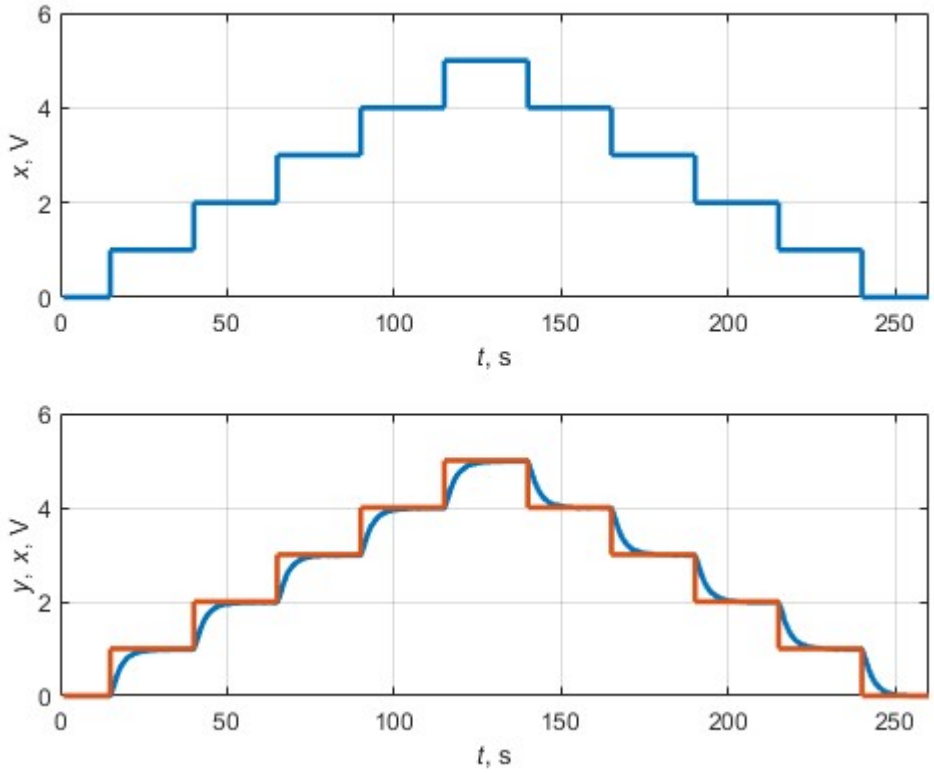


Obr. 2.25 – Druhé měření odezvy na skokové změny akční veličiny

Výsledný model identifikovaný online metodou nejmenších čtverců je

$$F(z^{-1}) = \frac{0{,}1955z^{-1} + 0{,}1049z^{-2}}{1 - 0{,}6847z^{-1} - 0{,}0131z^{-2}}. \tag{2.3}$$

Poslední měření je provedeno v celém měřicím rozsahu hodnot.



Obr. 2.26 – Měření odezvy soustavy v celém měřicím rozsahu

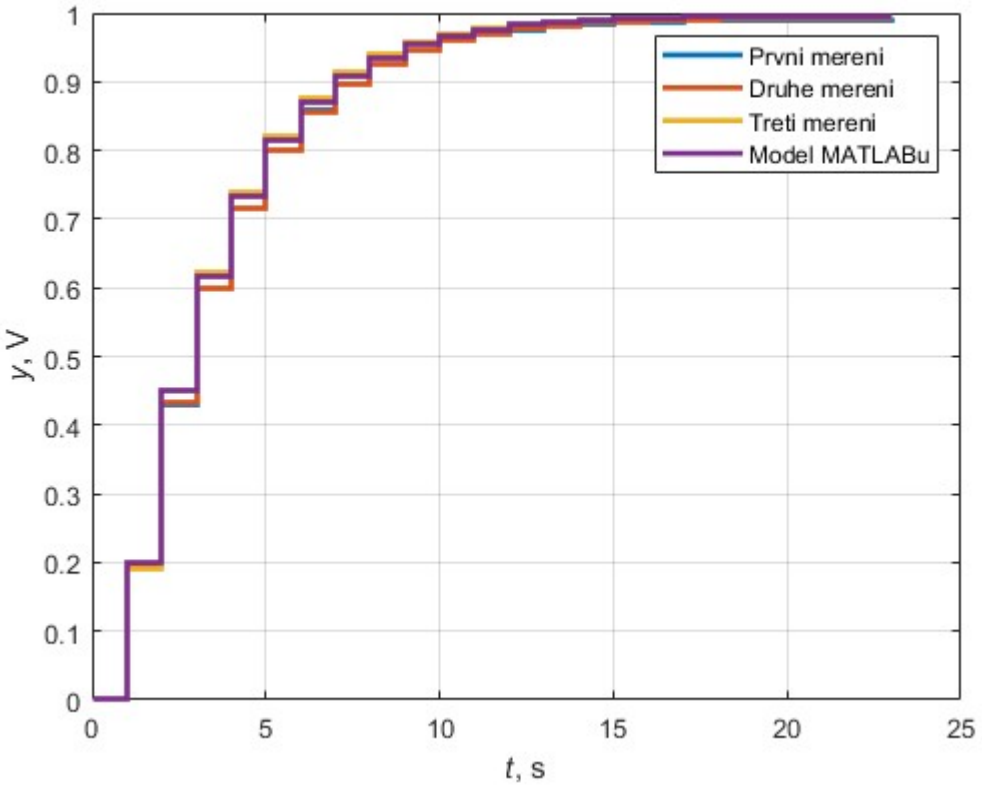
Výsledný model

$$F(z^{-1}) = \frac{0{,}1906z^{-1} + 0{,}1408z^{-2}}{1 - 0{,}6268z^{-1} - 0{,}0407z^{-2}}. \tag{2.4}$$

Naměřené hodnoty z posledního měření jsou zadány do MATLABu, kde je využita offline metoda nejmenších čtverců pro získání modelu (Dušek, 2005)

$$F(z^{-1}) = \frac{0{,}2004z^{-1} + 0{,}1242z^{-2}}{1 - 0{,}6294z^{-1} - 0{,}0450z^{-2}}. \tag{2.5}$$

Dále jsou porovnány modely pomocí přechodových charakteristik, které jsou zobrazeny na obr. 2.28.



Obr. 2.27 – Porovnání naměřených modelů pomocí přechodové charakteristiky

Většina modelů se překrývá, nebo se liší jen minimálně. Knihovna měření a identifikace je považována za funkční.

2.4.2 Regulace pomocí knihovny prediktivního regulátoru

První věcí, která musela být zkontrolována, jsou správně vypočtené hodnoty vektoru ***k***, součinu vektoru ***k*** s maticí ***Fp*** a první řádek matice ***Fp***. Tyto proměnné jsou důležité pro regulaci. Jako výchozí model soustavy je použita rovnice (2.4) ze třetího měření. Filtrační polynom je nastavený na [1; -0,8]. Parametr N_2 by měl být nastaven na čas, za který se soustava dokáže z 10 % žádané hodnoty dostat na 90 %, zde to jsou čtyři sekundy. Parametr N_2 je tedy nastaven na čtyři. Parametr N_u je prozatím nechán nastaven na jedničku. Parametr penalizace akční veličiny je nastaven na 0,8 a vzorkovací perioda je jedna sekunda. Výsledné vypočítané hodnoty Arduinem jsou uvedeny na obr. 2.29.

```
G: 0.19
K: 0.10 0.23 0.32 0.37
Kfp: -2.33 1.23 0.08 -0.28 1.60
Fp: 1.63 -0.59 -0.04 0.14 -0.80
```

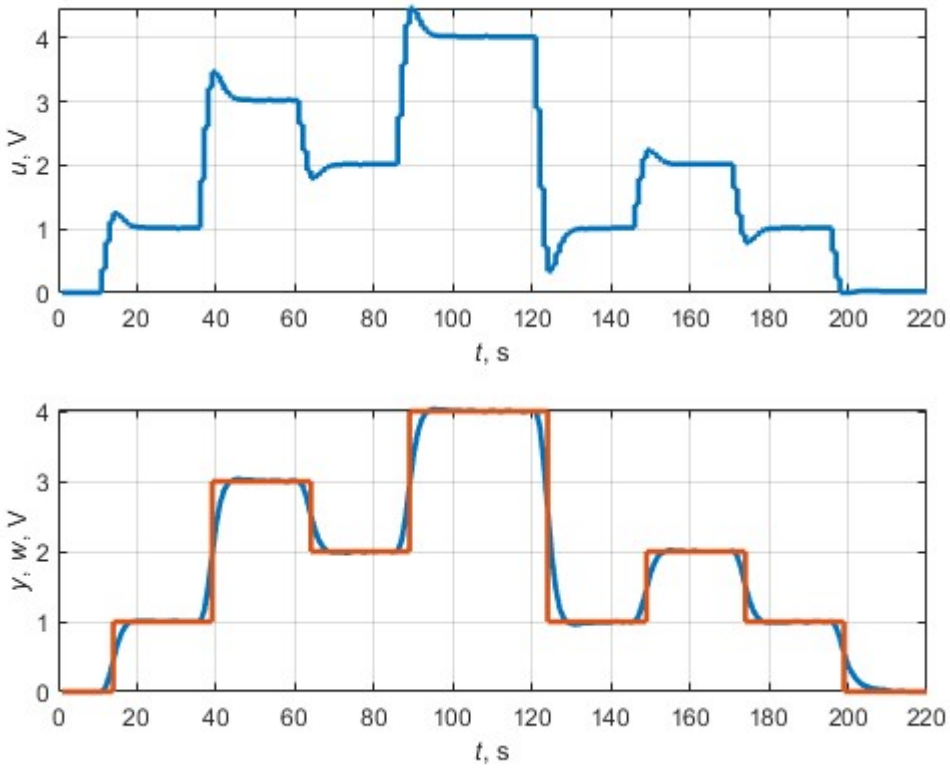
Obr. 2.28 – Vypočtené hodnoty regulátoru pomocí knihovny prediktivního regulátoru

Pro porovnání správnosti těchto hodnot byly vloženy stejně nastavené parametry do skriptu v programu MATLAB. Výsledné hodnoty jsou zobrazeny v tab. 2.1.

Tab. 2.1 – Hodnoty vypočtené pomocí MATLABu

Parametr	Hodnoty				
G	0,1906				
K	0,0966	0,2285	0,3151	0,3748	
Kfp	-2,331	1,2345	0,0815	-0,2818	1,601
Fp	1,6268	-0,5861	-0,0407	0,1408	-0,08

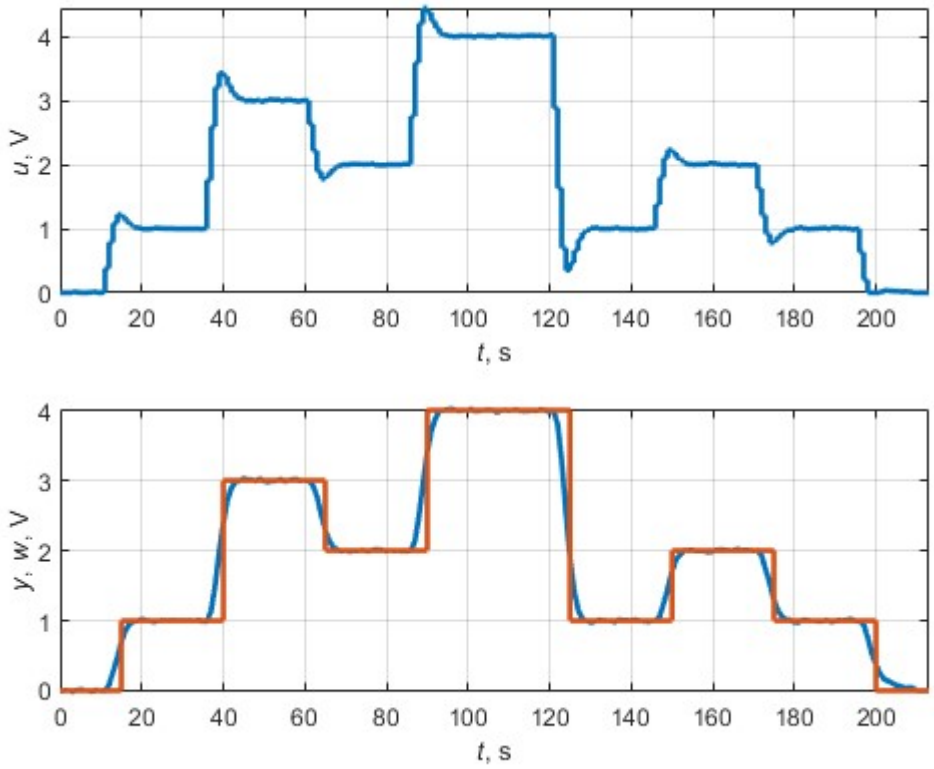
Hodnoty se liší pouze v zaokrouhlení, metoda *initialize* je také funkční. Následně je testován samotný prediktivní regulátor. Jsou navrhnuty dva regulační pochody, rychlý a pomalý. Dále je na těchto průbězích otestován parametr alfa, který filtruje žádanou hodnotu tak, aby nevytvářela prudké skoky.



Obr. 2.29 – Regulační pochod s pomalými změnami žádané hodnoty

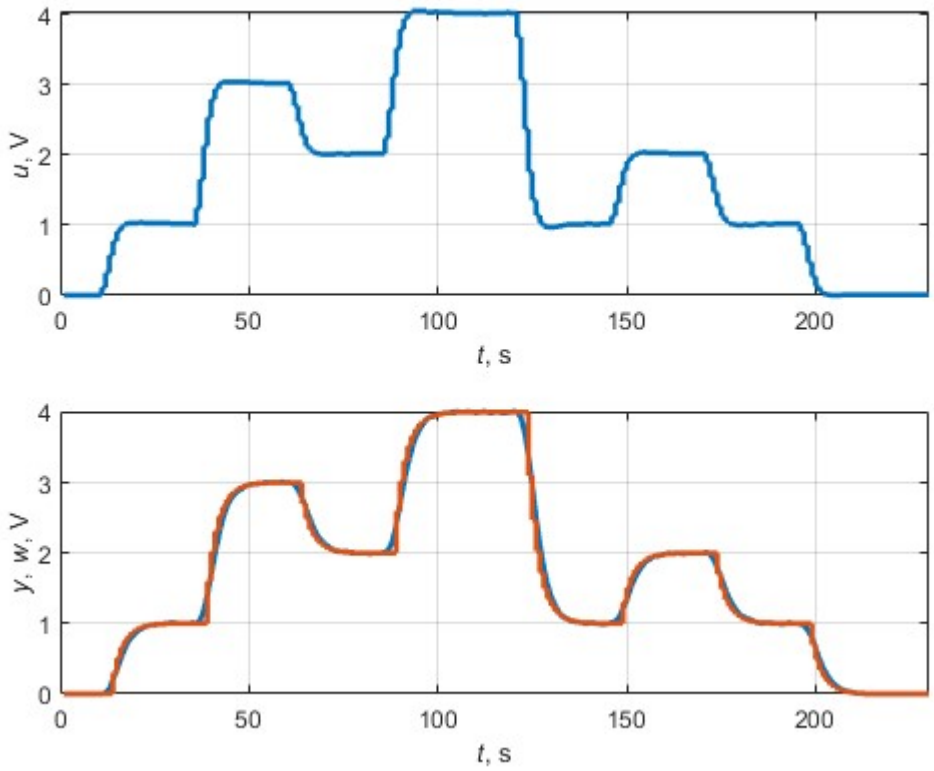
První graf obr. 2.31 zobrazuje průběh akční veličiny, druhý graf zobrazuje porovnání průběhu žádané hodnoty (oranžové) a regulované veličiny (modré). Všechny další grafy budou zobrazeny ve stejném duchu. Na obr. 2.31 je zobrazen regulační pochod s pomalými změnami žádané hodnoty, kdy parametr alfa je rovný nule. Dále je odsimulován stejný regulační průběh

v MATLABu, aby bylo porovnání správnosti funkce prediktivního regulátoru, obr. 2.32. Metoda *process* a všechny přídavné metody u prediktivního regulátoru fungují správně, protože odsimulovaný a reálný průběh jsou si velmi podobní.



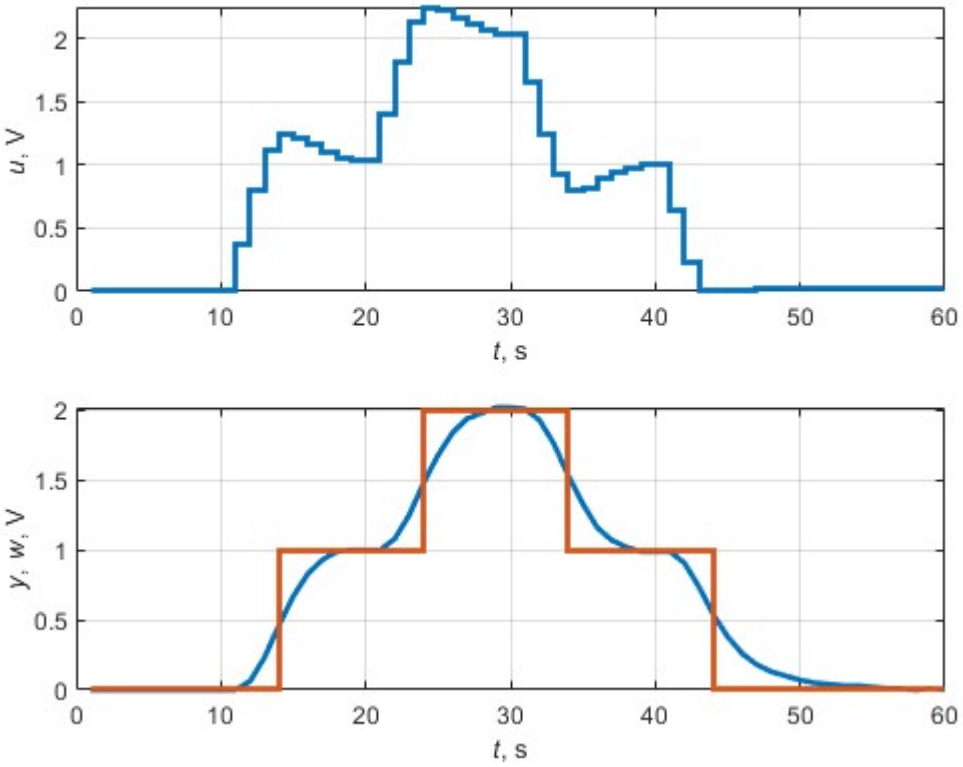
Obr. 2.30 – Simulovaný regulační pochod

Experimenty jsou dále pouze s reálnou soustavou, nic není simulováno. Funkčnost prediktivního regulátoru je správná. Poslední neotestovaná společná funkce knihoven je parametr *alfa*, který filtruje žádanou hodnotu tak, aby nevznikala prudká skoková změna. Filtrační parametr *alfa* je nastaven v tomto případě nastaven na hodnotu 0,7. Výsledek regulačního experimentu je na obr. 2.33. Parametr *alfa* vyfiltroval průběh žádané hodnoty, i tato funkce je správná.

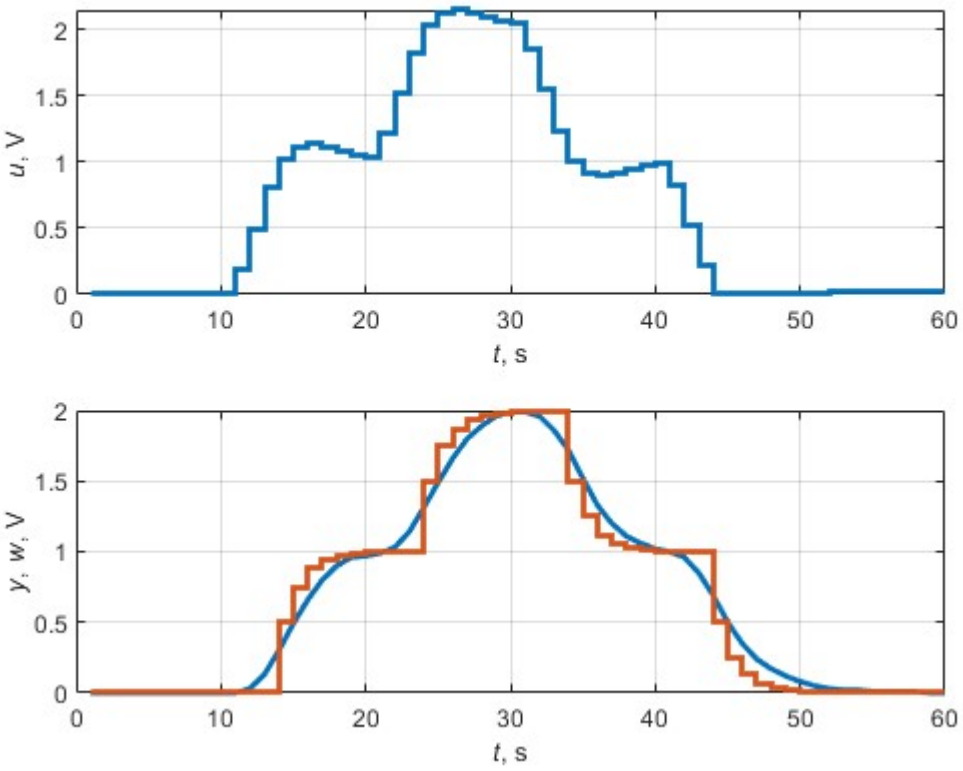


Obr. 2.31 – Regulační pochod s vyhlazenou žádanou hodnotou

Pro budoucí srovnání regulátorů je navrhnut další průběh žádané hodnoty. Tentokrát se bude měnit pouze třikrát každých 10 sekund. Výsledek měření je na obr. 2.34 a s parametrem alfa nastaveným na 0,5 na obr. 2.35.



Obr. 2.32 – Rychlý regulační pochod



Obr. 2.33 – Rychlý regulační pochod s parametrem alfa nastaveným na 0,5

2.4.3 Regulace pomocí knihovny prediktivního regulátoru pro soustavy prvního řádu s dopravním zpožděním

Aproximovaný model soustavy prvního řádu s dopravním zpožděním je získán stejným způsobem uvedeným v teoretické části v kapitole 1.2.1. Vstupním modelem je model získaný ze třetího měření pomocí knihovny měření a identifikace. Model soustavy je

$$F(z^{-1}) = \frac{0{,}3244z^{-1}}{1 - 0{,}6756z^{-1}} z^{-1}. \tag{2.6}$$

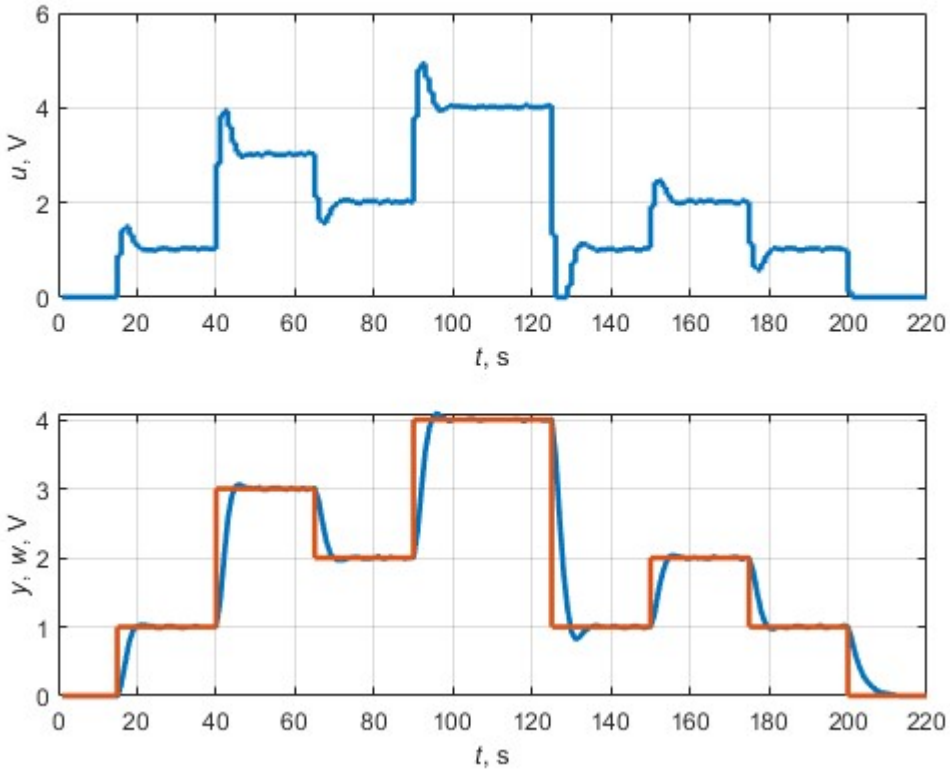
Dopravní zpoždění soustavy je shodné s periodou vzorkování, je tedy možné použít model (2.6). Tento model a parametr penalizace akční veličiny nastavený na 0,8 je zadán do knihovny prediktivního regulátoru pro soustavy prvního řádu s dopravním zpožděním. Je spuštěna metoda pro inicializaci regulátoru. Vypočtené parametry jsou na obr. 2.36, vypočtené parametry z MATLABu jsou v tab. 2.2. Parametry vypočtené Arduinem jsou zaokrouhlené, ale jinak se shodují. Metoda *initialize* je funkční a je možno přejít k měření regulačních pochodů, pro srovnání je použito stejných žádaných hodnot jako v případě prediktivního řízení.

```
l1: -1.86 l2: 0.97 l3: 0.90
```

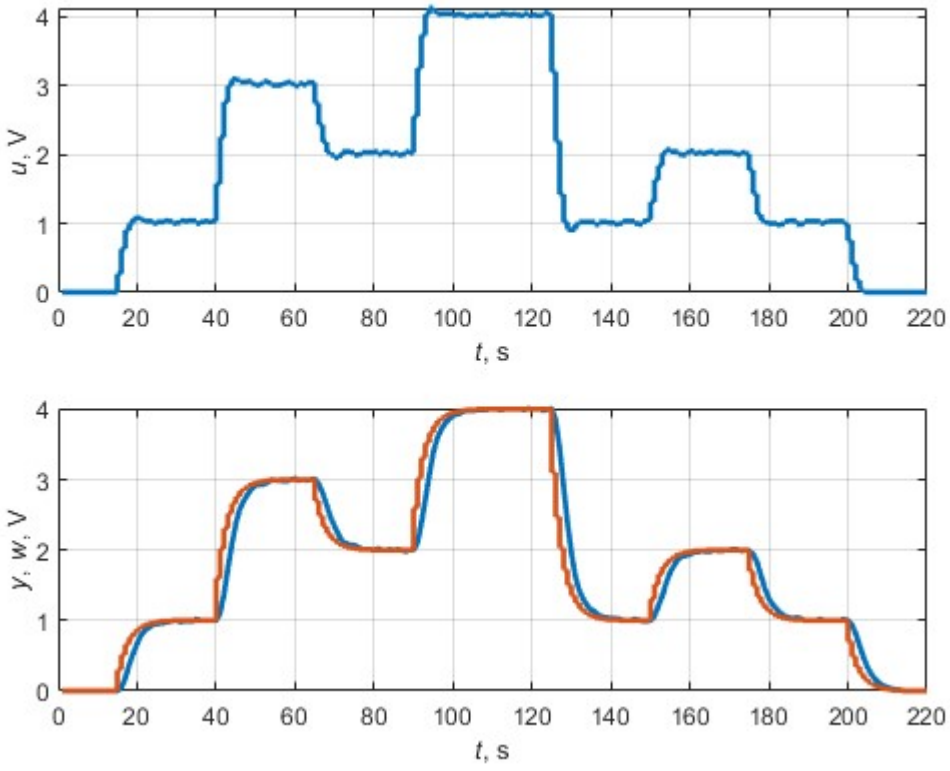
Obr. 2.34 – Hodnoty vypočtené knihovnou v Arduinu

Tab. 2.2 – Hodnoty vypočtené MATLABem

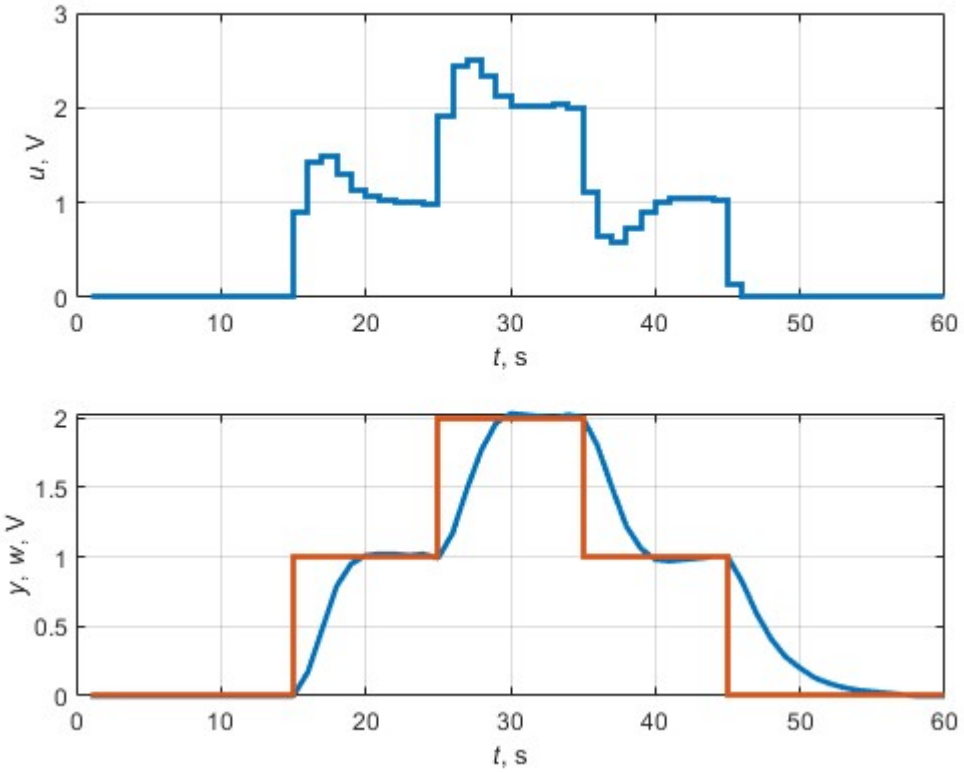
Parametr	Hodnota
l1	-1,8642
l2	0,9674
l3	0,8969



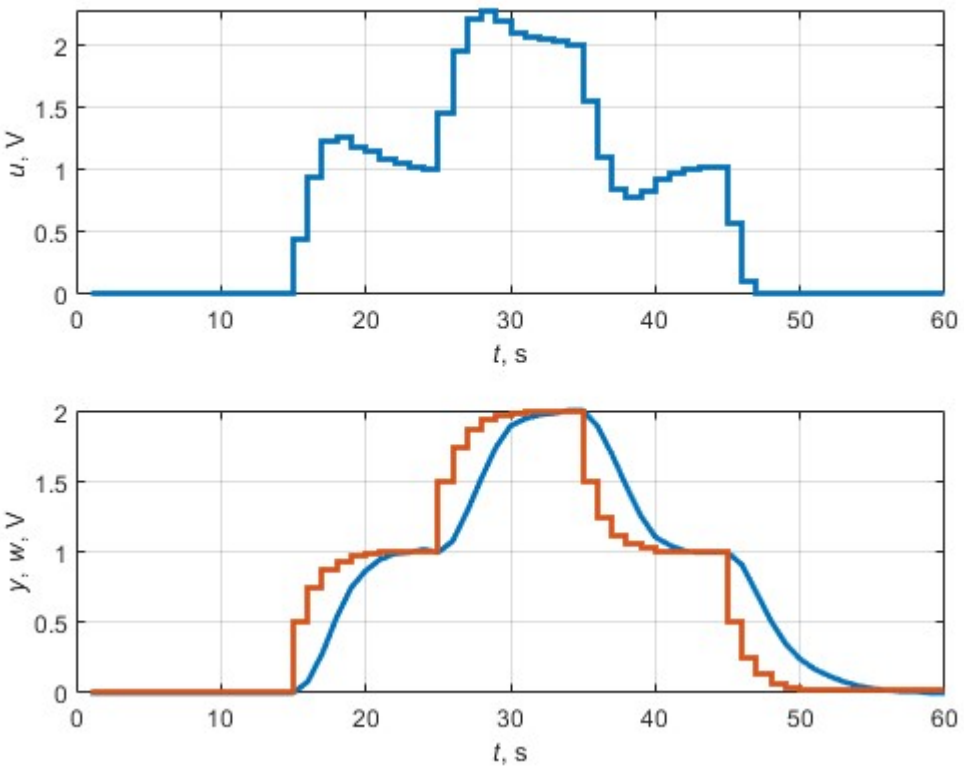
Obr. 2.35 – Regulační pochod s prediktivním regulátorem pro s. prvního řádu s dopr. zp.



Obr. 2.36 – Regulační pochod s parametrem alfa nastaveným na 0,8



Obr. 2.37 – Regulační pochod s rychlými změnami žádané hodnoty

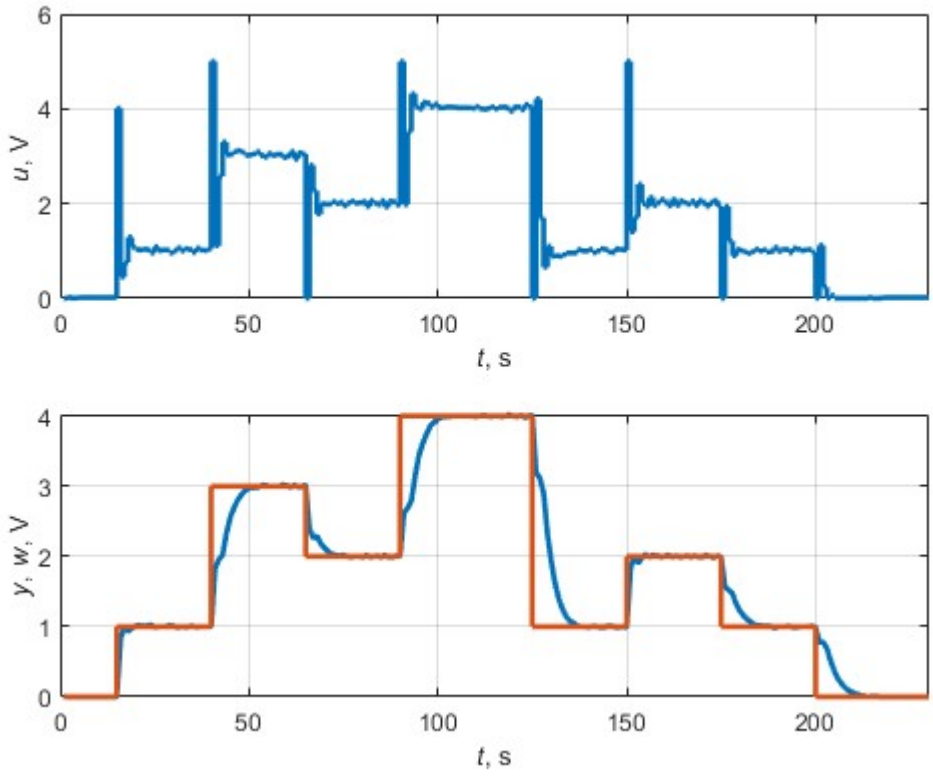


Obr. 2.38 – Regulační pochod s rychlými změnami a parametrem alfa nastaveným na 0,5

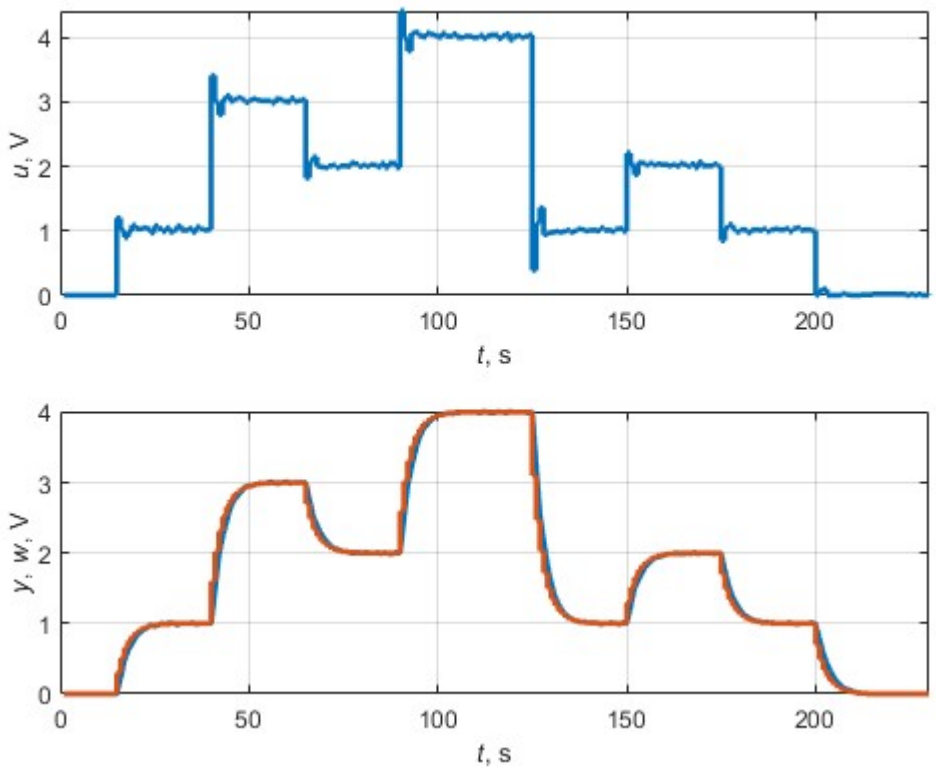
Výše uvedené regulační pochody, obr. 2.38 až obr. 2.41, jsou naměřené na reálné soustavě s knihovnou prediktivního regulátoru pro soustavu prvního řádu s dopravním zpožděním. Tím je ověřena funkce metody *process*, která je také funkční.

2.4.4 Regulace pomocí knihovny PID regulátoru

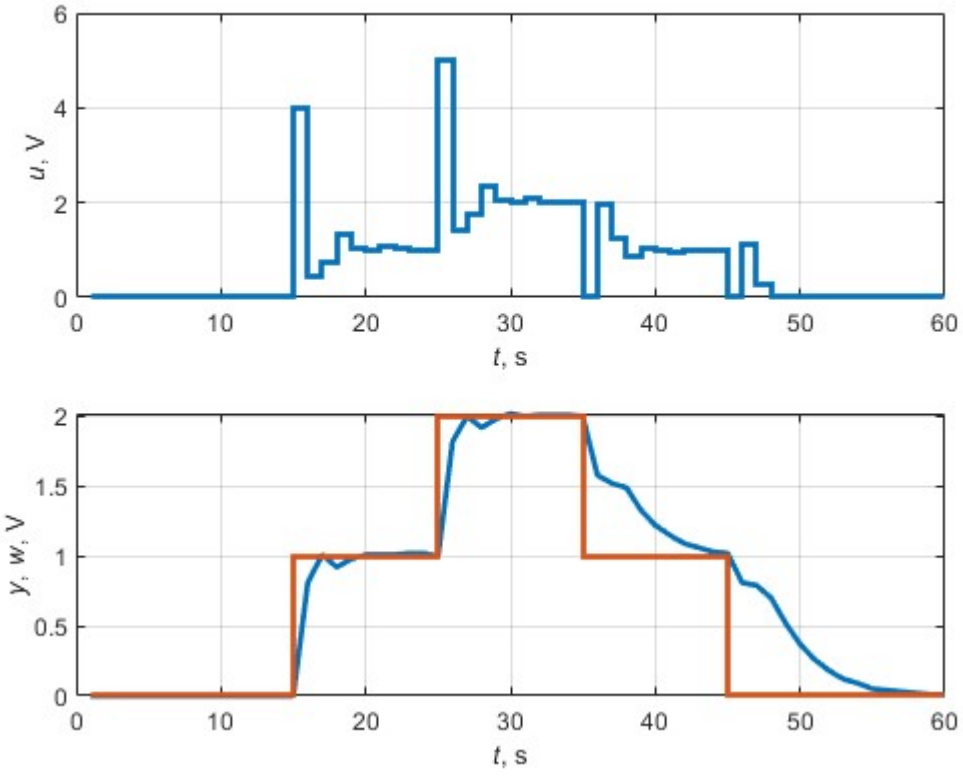
Poslední knihovnou je přírůstkový PID regulátor, který je nastaven pomocí Kuhnovy metody pro „rychlý“ regulátor. Průběhy žádané hodnoty jsou použity stejné jako v případě předchozích dvou regulátorů, aby bylo možné provést srovnání kvality regulace. Výsledky měření jsou na obr. 2.42 až obr. 2.45.



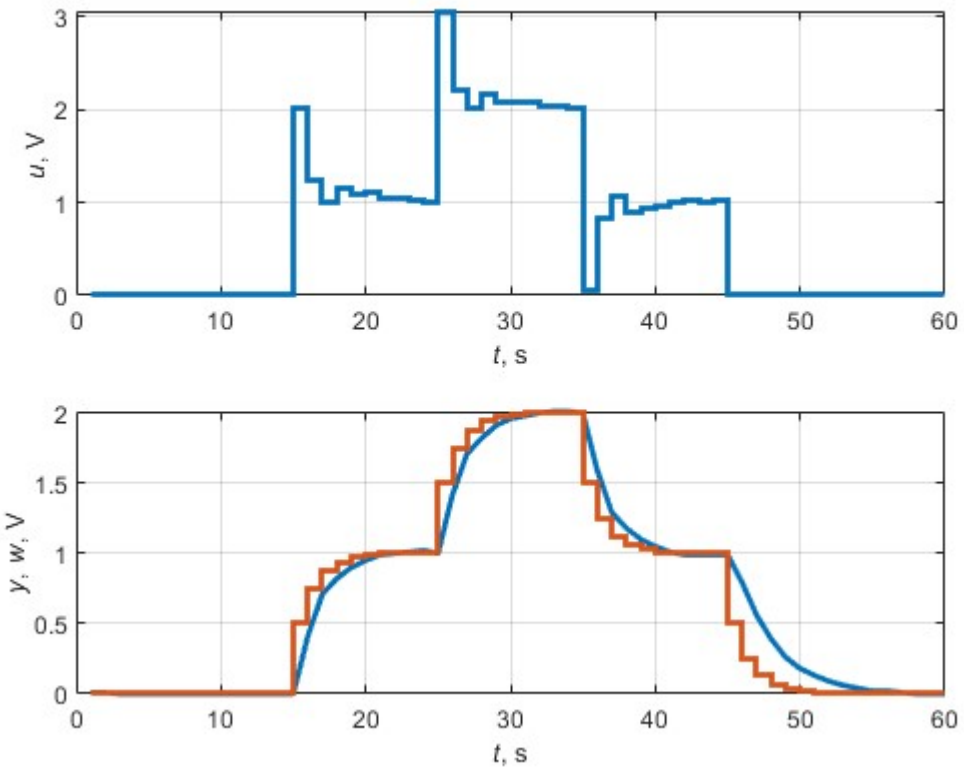
Obr. 2.39 – Regulační pochod PID regulátorem



Obr. 2.40 – Regulační pochod s PID regulátorem a parametrem alfa nastaveným na 0,7



Obr. 2.41 – Regulační pochod s rychlými změnami žádané hodnoty



Obr. 2.42 – Regulační pochod s rychlými změnami a parametrem alfa nastaveným na 0,5

2.5 Porovnání metod regulace

Pro kritérium vyhodnocení kvality regulačního pochodu je zvoleno kritérium ITAE, které je definováno rovnicí

$$\text{ITAE} = \int_0^{\infty} |e| \cdot t \cdot \mathrm{d}t, \tag{2.7}$$

kde e je regulační odchylka a

t – čas (Dušek, 2017).

Jednotlivá vypočtená kritéria kvality pro výše uvedené regulační pochody jsou v tab. 2.3 a tab. 2.4. Srovnání časů přípravy a náročnost implementace v tab. 2.5.

Tab. 2.3 – Vypočtená kritéria pro jednotlivé regulační pochody

	Regulační pochody s pomalou změnou žádaně veličiny	Regulační pochody s rychlou změnou žádaně veličiny
Simulace prediktivního regulátoru	17,6	6,12
Prediktivní regulátor	18,6	6,53
Simulace prediktivního regulátoru pro soustavy prvního řádu s dopr. zp.	34.28	11,6
Prediktivní regulátor se soustavou prvního řádu s dopr. zp.	34,47	11,73
Simulace PID regulátoru	35,86	12,3
PID regulátor	42,69	11,19
Bez regulace	44,98	14,4

Tab. 2.4 – Vypočtená kritéria pro jednotlivé regulační pochody s parametrem alfa

	Regulační pochody s pomalou změnou žádaně veličiny a parametrem alfa nastaveným na 0,7	Regulační pochody s rychlou změnou žádaně veličiny a parametrem alfa nastaveným na 0,5
Simulace prediktivního regulátoru	6,21	3,22
Prediktivní regulátor	11,5	4,8
Simulace prediktivního regulátoru pro soustavy prvního řádu s dopr. zp.	31,02	10,79
Prediktivní regulátor se soustavou prvního řádu s dopr. zp.	30,71	10,83
Simulace PID regulátoru	16,17	6,47
PID regulátor	14,95	6,36
Bez regulace	45,24	14,27

Tab. 2.5 – Srovnání náročnosti implementace regulátorů

	Doba výpočtu parametrů	Implementace	Nastavení regulátoru
Prediktivní regulátor	61 ms	Složitá	Lehké, pokud je znám model
Prediktivní regulátor pro soustavy prvního řádu s dopr. zp.	16 ms	Lehká	Lehké, pokud je znám model
PID Regulátor	1 ms	Lehká	Lehké až složité podle metody

3 ZÁVĚR

Cílem této diplomové práce bylo aplikovat metody prediktivního řízení do jednoduchého řídicího systému. Praktická část práce je otestována na vývojovém kitu Arduino UNO, který lze programovat v jazyce C++, a má tedy velmi mnoho možností, jak vytvořit požadované knihovny.

Prediktivní řízení je popsáno v teoretické části práce. Dále je zde uvedena praktičtější metoda prediktivního řízení, která je založená na modelu soustavy prvního řádu s dopravním zpožděním. Teoretická část se dále zabývá získáním modelu soustavy, který je stěžejní pro prediktivní regulaci. Na konci teoretické části je popsán „jednoduchý“ řídící systém s popisem vývojového kitu Arduino UNO.

Praktická část je věnována specifickým problémům, které musely být vyřešeny při implementaci prediktivního regulátoru, jako je například malá paměť Arduina UNO. Toto omezení bylo vyřešeno nastavením malých hodnot horizontu sledování žádané hodnoty a horizontu akční veličiny. Pro složitější soustavy a delší horizonty by bylo nutné použít mikroprocesor s větší pamětí. Dále musel být navržen způsob, jakým se bude do prediktivního regulátoru zadávat žádaná hodnota. Toto bylo vyřešeno jednoduchým algoritmem, který posouvá hodnoty uvnitř pole podle potřeby. Tento algoritmus je v práci využit několikrát.

Praktická část práce ukazuje, že i jednoduchý mikroprocesor, jaký má Arduino UNO, je schopný realizovat moderní metodu řízení, jakou je prediktivní regulace. Stačí pouze nastavit model soustavy, horizonty žádané hodnoty a akční veličiny a parametr penalizace. Knihovna poté provede všechny potřebné maticové operace k získání regulačního zákonu a provádí samotnou regulaci. Jelikož prediktivní řízení vychází z modelu soustavy, bylo nutné získat model soustavy. Z tohoto důvodu byla vytvořena knihovna pro online metodu nejmenších čtverců. Byly vytvořeny čtyři různé knihovny, pro prediktivní řízení, prediktivní řízení pro soustavy prvního řádu, přírůstkový PID regulátor a knihovna pro měření a identifikaci soustav. Knihovny byly otestovány na jednoduché soustavě druhého řádu – dva RC články v sérii.

Kvalita regulace byla posuzována na základě hodnoty kritéria ITAE. Prediktivní regulace je nejkvalitnější ve všech regulačních pochodech. Druhého nejlepšího výsledku bylo dosaženo pomocí prediktivního řízení pro soustavu prvního řádu s dopravním zpožděním a pomocí PID regulace. Prediktivní řízení pro soustavy prvního řádu s dopravním zpožděním není tak agresivní na akční veličinu, jako PID regulátor. Výsledkem srovnání je označení tohoto řízení za úspornější, protože hodnota akční veličiny se tolik nemění, jako v případě PID regulátoru nastaveného Kuhnovou metodou. PID regulátor má navrch v případech, kdy je

žádaná hodnota filtrována parametrem *alfa*. Kvalita regulace pomocí PID regulátoru bude záležet na metodě nastavení regulátoru, u prediktivního řízení bude záležet na přesnosti získaného modelu systému a na hodnotách penalizačních koeficientů.

Prediktivní řízení bylo nejsložitější na implementaci[1], zejména příprava matic, protože v jazyce C není žádná ochrana paměti a vše musí obstarat programátor. Při testování knihoven muselo být ošetřeno chování, při kterém se bylo přistupováno do jiných paměťových polí, než byla alokována matice. Prediktivní řízení pro soustavy prvního řádu s dopravním zpožděním bylo odvozeno autory Camacho a Bordons (2007), takže samotná implementace byla stejně jednoduchá, jako implementace PID regulátoru.

Regulovat jednoduché soustavy prediktivním regulátorem nedává smysl. Zkušený inženýr umí nastavit PID regulátor „od oka“ tak, aby regulace jednoduchého systému byla kvalitní. Aplikace prediktivních regulátorů jsou v průmyslu omezené na několik oblastí, například rafinérie a papírenský průmysl. Prediktivní řízení má smysl u složitějších procesů, ale i tam je pod vrstvou prediktivního řízení klasická PID regulace.

Dalším rozšířením práce by byl přesun knihoven na mikroprocesor s větší pamětí, tak aby mohly být použity delší horizonty. Knihovny jsou strukturované a měly by toto umožnit lidem pokračujícím v této práci.

[1] Celý projekt se všemi knihovnami je dostupný na adrese: https://bitbucket.org/st46514/thesisproject/src/master/

LITERATURA

ASTRÖM, K. J. a WITTENMARK, B. 1995 *Adaptive control*. 2nd ed. Reading: Addison-Wesley, ISBN 0-201-55866-1.

BALÁTĚ, J. 2004. *Automatické řízení*. 2., přeprac. vyd. Praha: BEN – technická literatura. ISBN 80-7300-148-9.

BERTSEKAS, D. P. c1999. *Nonlinear programming*. 2nd ed. Belmont: Athena Scientific. ISBN 1-886529-00-0.

CAMACHO, E. F. a BORDONS, C. 2007. *Model Predictive Control* (*Second Edition*). London: Springer-Verlag London Limited. 405 s. ISBN 1-85233-694-3.

DORF, R. C. a BISHOP, R. H. 1998. *Modern control systems*. 8th ed. Menlo Park: Addison-Wesley. ISBN 0-201-30864-9.

DUŠEK, F. 2017. *Teorie automatického řízení I*. Přednášky Fakulta elektrotechniky a informatiky. Univerzita Pardubice: Osobní sdělení. [cit. 2019-03-30].

DUŠEK, F. a HONC, D. 2005. *Matlab a Simulink: úvod do používání*. Pardubice: Univerzita Pardubice, ISBN 80-7194-776-8.

HONC, D. 2018a. *Teorie automatického řízení III*. Přednášky Fakulta elektrotechniky a informatiky. Univerzita Pardubice: Osobní sdělení. [cit. 2019-03-30].

HONC, D. 2018b. *Identifikace a modelování dynam. systémů*. Přednášky Fakulta elektrotechniky a informatiky. Univerzita Pardubice: Osobní sdělení. [cit. 2019-03-30].

KUPKA, L. 2018. *Teorie automatického řízení II.* Přednášky Fakulta elektrotechniky a informatiky. Univerzita Pardubice: Osobní sdělení. [cit. 2019-03-30].

MIKLEŠ, J. a FIKAR, M. *Modelovanie, identifikácia a riadenie procesov 2*. STU Bratislava, 2004. ISBN 80-227-2134-4

PEKAŘ, J. a JECH, J. 2007. *Prediktivní řízení průmyslových procesů*. Automa. 5(2), 1. ISSN 1210-9592. Dostupné také z: http://automa.cz/cz/casopis-clanky/prediktivni-rizeni-prumyslovych-procesu-2007_02_34170_1546/

ROSSITER, J. A. c2003. *Model-based predictive control: a practical approach*. Boca Raton: CRC Press. CRC Press control series. ISBN 0-8493-1291-4.

ŠTECHA, J. a HAVLENA, V. 1999. *Teorie dynamických systémů*. Vyd. 2. Praha: Vydavatelství ČVUT. ISBN 80-01-01971-3.

VODA, Z. 2015. *Průvodce světem Arduina*. Bučovice: Martin Stříž, ISBN 978-80-87106-90-7.

WELLSTEAD, P. E. a. ZARROP, B. M. c1991. *Self-tuning systems: control and signal processing*. New York: Wiley. ISBN 0-471-92883-6.

PŘÍLOHY

A – CD

B – Příklady programů pro použití knihoven

Příloha A

Příloha k diplomové práci

Aplikace prediktivního regulátoru v jednoduchém řídicím systému

Jiří Vinduška

CD

OBSAH

Příloha B

Příloha k diplomové práci

Aplikace prediktivního regulátoru v jednoduchém řídicím systému

Jiří Vinduška

Příklady programů pro použití knihoven

OBSAH

1 UKÁZKA PROGRAMU PRO VYUŽITÍ KNIHOVNY PREDIKTIVNÍHO REGULÁTORU

```
#include "Arduino.h"
#include "PredictiveController.h"
#include "InputOutput.h"
#include "MemoryFree.h"

#define ANALOG_INPUT_PIN 0
#define PWM_OUTPUT_PIN 11

PredictiveController predictiveController;
InputOutput inputOutput;
/*This lines are set by user.*/
/*Model of plant in CARIMA model, maximum order is 3 if N2=4*/
float a[3] = { 1.0000,   -0.6268, -0.0407};
float b[3] = {  0,    0.1906, 0.1408};
/*Filter of first order*/
float c[2] = { 1.0000,-0.8};
/*Setpoint horizont(Sh). Maximal is 6 for Arduino Uno.*/
int N2 = 4;
/*Control horizont. Maximal is 1 if SH is 6 otherwise if SH=4
CH is maximal of 4. Nu<=N2*/
int Nu = 1;
/*Penalty for control value*/
float q = 0.8;
/*This parameter is used for smooth setPoint curve. 0=unit
jump,0.99=smooth curve of first order*/
float alfa=0.0;
/*Sizes of previously filled vectors*/
int nA = 3;
int nB = 3;
int nC = 2;
/*sample time*/
int Ts = 1;
////////////////////////////////
long previous_millis = 0;
const long interval = Ts * 1000;
float last_sample = 0.0;
float controll_value = 0.0;
int input_value = 0;
int output_value = 0;

void setup() {

      Serial.begin(9600);
      Serial.flush();
      pinMode(ANALOG_INPUT_PIN, INPUT);
      pinMode(PWM_OUTPUT_PIN, OUTPUT);
```

```
      Serial.println("Starting");
      /*This method should be called first*/
      predictiveController.initialize(a, b, c, N2, Nu, q,alfa,
nA, nB, nC);

      /* Here I have written information about free memory in
UNO, controller data and computing time*/
      Serial.println(freeMemory());
      Serial.println();
      predictiveController.showControllerData();
      Serial.println();
      Serial.println(millis());

      Serial.println("t, y, u, w, y(t+1)");
}
void loop() {
      long current_millis = millis();
      /*Every sample time is processed this condition*/
      if (current_millis - previous_millis >= interval) {
            previous_millis = current_millis;

            input_value = analogRead(ANALOG_INPUT_PIN);
            last_sample =
inputOutput.inputToVoltage(input_value);

            /*Here you can set the set points in future. Method
parameters are (set point,current_time,start_time,end_time).
End time>N2 and end time>start time.*/
            predictiveController.setSetpoint(1.0, current_millis,
10, 35);
            predictiveController.setSetpoint(2.0, current_millis,
35, 60);
            predictiveController.setSetpoint(1.0, current_millis,
60, 85);
            predictiveController.setSetpoint(0.0, current_millis,
85, 110);

            /*Computing of control law*/
            controll_value =
predictiveController.process(last_sample);

            output_value =
inputOutput.voltageToOutput(controll_value);
            analogWrite(PWM_OUTPUT_PIN, output_value);
            /* This method prints all measurements*/
            predictiveController.printData(current_millis);

      }
}
```

2 UKÁZKA PROGRAMU PRO VYUŽITÍ KNIHOVNY PRO PREDIKTIVNÍ REGULÁTOR PRO SOUSTAVY PRVNÍHO ŘÁDU S DOPRAVNÍM ZPOŽDĚNÍM

```
#include "Arduino.h"
#include "PredictiveDTController.h"
#include "InputOutput.h"
#include "MemoryFree.h"

#define ANALOG_INPUT_PIN 0
#define PWM_OUTPUT_PIN 11


PredictiveDTController predictiveDTController;
InputOutput inputOutput;
/*This lines are set by user.*/
/*Model of plant in CARIMA model, maximum order is 1*/
float a[2] = {1.0, -0.6905};
float b[2] = {0.0, 0.3079};
/*dead time*/
int d = 1;
/*Penalty for control value*/
float q = 0.8;
/*This parameter is used for smooth setPoint curve. 0=unit
jump,0.99=smooth curve of first order*/
float alfa= 0.7;
/*Parameter for shifting set point tothe future*/
int future = 0;
/*dead time*/
int Ts = 1;
////////////////////////////////

long previous_millis = 0;
const long interval = Ts * 1000;
float last_sample = 0.0;
float controll_value = 0.0;
int input_value = 0;
int output_value = 0;

void setup()
{
// Add your initialization code here
      Serial.begin(9600);
      Serial.flush();
      pinMode(ANALOG_INPUT_PIN, INPUT);
      pinMode(PWM_OUTPUT_PIN, OUTPUT);

      Serial.println("Starting");
```

```
      /*This method should be called first*/
      predictiveDTController.initialize(a, b, d, q, alfa,
future);
      /* Here I have written information about free memory in
UNO, controller data and computing time*/
      Serial.println(freeMemory());
      Serial.println();
      predictiveDTController.showControllerData();
      Serial.println();
      Serial.println(millis());
      Serial.println("t, y, u, w, y(t+1)");
}

// The loop function is called in an endless loop
void loop()
{
      long current_millis = millis();
      /*Every sample time is processed this condition*/
      if (current_millis - previous_millis >= interval) {
            previous_millis = current_millis;

            input_value = analogRead(ANALOG_INPUT_PIN);
            last_sample =
inputOutput.inputToVoltage(input_value);

            /*Here you can set the set points in future. Method
parameters are (set point,current_time,start_time,end_time).
End time>start time.*/
            predictiveDTController.setSetpoint(1.0,
current_millis, 10, 35);
            predictiveDTController.setSetpoint(2.0,
current_millis, 35, 60);
            predictiveDTController.setSetpoint(1.0,
current_millis, 60, 85);
            predictiveDTController.setSetpoint(0.0,
current_millis, 85, 110);

            /*Computing of control law*/
            controll_value =
predictiveDTController.process(last_sample);

            output_value =
inputOutput.voltageToOutput(controll_value);
            analogWrite(PWM_OUTPUT_PIN, output_value);
            /* This method prints all measurements*/
            predictiveDTController.printData(current_millis);

      }
}
```

3 UKÁZKA PROGRAMU PRO VYUŽITÍ KNIHOVNY PID REGULÁTORU

```
#include "Arduino.h"
#include "PIDController.h"
#include "InputOutput.h"
#include "MemoryFree.h"

#define ANALOG_INPUT_PIN 0
#define PWM_OUTPUT_PIN 11

PIDController pidController;
InputOutput inputOutput;
/*This lines are set by user.*/
float r0 = 2.0;
float Ti = 2.3599;
float Td = 0.5723;
float setpoint = 0.0;
float alfa = 0.0;
int future = 0;
int Ts = 1;
////////////////////////////////

long previous_millis = 0;
const long interval = Ts * 1000;
float last_sample = 0.0;
float controll_value = 0.0;
int input_value = 0;
int output_value = 0;
void setup() {
      Serial.begin(9600);
      Serial.flush();
      pinMode(ANALOG_INPUT_PIN, INPUT);
      pinMode(PWM_OUTPUT_PIN, OUTPUT);

      Serial.println("Starting");
      /*This method should be called first*/
      pidController.initialize(r0, Ti, Td, Ts, setpoint, alfa,
future);
      /* Here I have written information about free memory in
UNO, controller data and computing time*/
      Serial.println(freeMemory());
      Serial.println();
      pidController.showControllerData();
      Serial.println();
      Serial.println(millis());
      Serial.println("t, y, u, w");

}
```

```
void loop() {
      long current_millis = millis();
      /*Every sample time is processed this condition*/
      if (current_millis - previous_millis >= interval) {
            previous_millis = current_millis;

            input_value = analogRead(ANALOG_INPUT_PIN);
            last_sample =
inputOutput.inputToVoltage(input_value);

            /*Here you can set the set points in future. Method
parameters are (set point,current_time,start_time,end_time).
End time>start time.*/
            pidController.setSetpoint(1.0, current_millis, 10,
35);
            pidController.setSetpoint(2.0, current_millis, 35,
60);
            pidController.setSetpoint(1.0, current_millis, 60,
85);
            pidController.setSetpoint(0.0, current_millis, 85,
110);

            /*Computing of control law*/
            controll_value = pidController.process(last_sample);

            output_value =
inputOutput.voltageToOutput(controll_value);
            analogWrite(PWM_OUTPUT_PIN, output_value);
            /* This method prints all measurements*/
            pidController.printData(current_millis);

      }
}
```

4 UKÁZKA PROGRAMU PRO VYUŽITÍ KNIHOVNY PRO MĚŘENÍ A IDENTIFIKACI

```
#include "Arduino.h"
#include "MemoryFree.h"
#include "ManualControll.h"
#include "InputOutput.h"

#define ANALOG_INPUT_PIN 0
#define PWM_OUTPUT_PIN 11
ManualControll manualControl;
InputOutput inputOutput;

/*This lines are set by user.*/
float setpoint = 0.0;
/*set point smoothening parameter.*/
float alfa=0.0;
int future = 0;
int Ts = 1;
/*Order of plant to identify.*/
int n = 2;
////////////////////////////////
long previous_millis = 0;
const long interval = Ts * 1000;
float last_sample = 0.0;
float controll_value = 0.0;
int input_value = 0;
int output_value = 0;

void setup() {
      Serial.begin(9600);
      Serial.flush();
      pinMode(ANALOG_INPUT_PIN, INPUT);
      pinMode(PWM_OUTPUT_PIN, OUTPUT);

      Serial.println("Starting");
      /*This method should be called first*/
      manualControl.initialize(setpoint, alfa, future, n);
      /* Here I have written information about free memory in
UNO, controller data and computing time*/
      Serial.println(freeMemory());
      Serial.println();
      manualControl.showControllerData();
      Serial.println();
      Serial.println(millis());
      Serial.println("t, y, u");
}

// The loop function is called in an endless loop
```

```
void loop() {
      long current_millis = millis();
      /*Every sample time is processed this condition*/
      if (current_millis - previous_millis >= interval) {
            previous_millis = current_millis;

            input_value = analogRead(ANALOG_INPUT_PIN);
            last_sample =
inputOutput.inputToVoltage(input_value);

            /*Here you can set the set points in future. Method
parameters are (set point,current_time,start_time,end_time).
End time>start time.*/
            manualControl.setSetpoint(1.0, current_millis, 10,
35);
            manualControl.setSetpoint(2.0, current_millis, 35,
60);
            manualControl.setSetpoint(1.0, current_millis, 60,
85);
            manualControl.setSetpoint(0.0, current_millis, 85,
110);

            /*Setting set point to control value.*/
            controll_value = manualControl.process(last_sample);
            /*Identify plant method expects sample and
control_value(y and u).*/
            manualControl.identifyPlant(last_sample,
controll_value);

            output_value =
inputOutput.voltageToOutput(controll_value);
            analogWrite(PWM_OUTPUT_PIN, output_value);
            /* This method prints all measurements*/
            manualControl.printData(current_millis);

      }
}
```

5 KNIHOVNA PRO PŘEVOD ANALOGOVÝCH HODNOT NA DIGITÁLNÍ A NAOPAK

```
#include "InputOutput.h"

InputOutput::InputOutput() {
	// TODO Auto-generated constructor stub

}

float InputOutput::inputToVoltage(int _input_value) {
	float voltage;
	//prevod z rozsahu 0-1023 na 0-5V s desetinným místem
	voltage = _input_value * (5.0 / 1023.0);

	return voltage;
}

int InputOutput::voltageToOutput(float _output_value) {
	int output;
	//Dolní a horní omezení
	if (_output_value < 0.0) {
		_output_value = 0.0;
	}
	if (_output_value > 5.0) {
		_output_value = 5.0;
	}
	// prevod z 0-5V na 0-255 bez desetinneho mista
	output = _output_value * 51;
	return output;
}

InputOutput::~InputOutput() {
	// TODO Auto-generated destructor stub
}
#ifndef INPUTOUTPUT_H_
#define INPUTOUTPUT_H_

class InputOutput {
public:
	InputOutput();
	float inputToVoltage(int _input_value);
	int voltageToOutput(float _output_value);
	virtual ~InputOutput();
};

#endif /* INPUTOUTPUT_H_ */
```