VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

Adaptivní řízení elektromechanických aktuátorů s využitím dopředného kompenzátoru založeného na více-modelovém přístupu

Adaptive control of electromechanical actuators using multiple model adaptive feed forward compensator

DISERTAČNÍ PRÁCE
DOCTORAL THESIS

AUTOR PRÁCE — Ing. Václav Sova
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing.Robert Grepl, Ph.D.
SUPERVISOR

BRNO 2018

Abstrakt

Tato dizertační práce se zabývá odvozením nového adaptivního dopředného kompenzátoru, který bude použit pro řízení elektromechanických aktuátorů používaných v automobilovém průmyslu. Jedná se o elektronickou škrticí klapku a EGR ventil. Představený adaptivní kompenzátor je odvozen od již existující metody více-modelového zpětnovazebního řízení. V této práci je uvedeno odvození této metody a simulační a experimentální ověření.

Dále jsou v této práci představeny a shrnuty nejznámější filtry pro získávání derivací digitálního signálu, protože je dopředný kompenzátor vyžaduje pro svou funkci. Z těchto filtrů je vybrán jeden konkrétní, jehož koeficienty vedou při vhodném nastavení na celočíselné násobení a celočíselnou implementaci filtrace. Toho využijeme při implementaci tohoto filtru na FPGA, kde ukážeme, že tento filtr zabere mimimum zdrojů FPGA vůči implementaci filtrace pomocí aritmetiky s pevnou či plovoucí desetinnou čárkou.

Abstract

This thesis deals with the derivation of novel adaptive feed forward compensator, which will be used for the control of the electromechanical actuators used in automotive industry. The electromechnical actuators are an electronic throttle valve and an EGR valve. The introduced adaptive compensator is derived from an existing multiple model feedback control method. This work describes the derivation of this method and simulation and experimental verification.

In addition, the most well known digital filter differentiators are presented and summarized in this paper because the feed forward compensator needs them for its operation. From these filters, one specific is chosen, whoose coefficients for the specific setting leads to integer multiplication and an integer implementation of the filter. This will be used to implement this filter to the FPGA and then we prove, that this implementation saves a lot of FPGA resources compared to filters implemented using fixed or floating-point arithmetic.

Bibliografická citace

SOVA, V. Adaptivní řízení elektromechanických aktuátorů s využitím dopředného kompenzátoru založeného na více-modelovém přístupu. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2018. 96 s. Vedoucí dizertační práce doc. Ing. Robert Grepl, Ph.D..

Čestné prohlášení

Prohlašuji, že jsem svou disertační práci vypracoval samostatně pod vedením doc. Ing. Roberta Grepla, PhD. s využitím vlastních znalostí získaných během studia a na základě použité a důsledně citované odborné literatury

V Brně, Václav Sova, 2018

PODĚKOVÁNÍ

Tímto bych chtěl poděkovat těm, kteří mi pomáhali při vytváření této práce.
Především děkuji vedoucímu dizertační práce, panu doc. Ing. Robertu Greplovi, Ph.D. za cenné rady a věnovaný čas.
Dále děkuji kolegům z mechatronické laboratoře za příjemné a inspirující pracovní prostředí.
V neposlední řadě bych chtěl poděkovat rodině za podporu.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Mnoho současných výrobků má mechatronický charakter. Jedná se o takové spojení mechanické, elektrické, elektronické a řídicí části, aby výsledný produkt představoval funkčně vyvážený celek a bylo dosaženo synergického efektu. Získáváme tak výrobky, které mají lepší vlastnosti než předchozí generace a je umožněn i vznik zcela nových zařízení.

Ve velké většině mechatronických výrobků existují řídicí struktury s uzavřenou regulační smyčkou. Hlavním úkolem regulačního mechanismu je zajistit, aby se řízená soustava chovala dle požadavků, s uvažováním všech vnějších vlivů a s co nejmenším zásahem člověka. V případě, že řízená soustava je lineární, časově invariantní, existuje mnoho zavedených, spolehlivých a stabilních metod pro řízení. V případě, že řízená soustava není lineární, mění se v čase, uvažujeme různé poruchy, atd., jedná se řádově o složitější problém.

Obecně je řízení nelineárních soustav stále otevřený vědecký problém. Existuje mnoho různých metod řízení těchto soustav, ale žádná není univerzální a každá metoda je vhodná na určitou třídu problémů. U reálných soustav navíc může docházet k tomu, že neznáme přesný matematický model řízené soustavy, případně soustava nebo její okolí podléhá pozvolným či náhlým změnám. Takové případy vedou k použití algoritmů z oborů adaptivního nebo robustního řízení.

V této práci se zaměříme na oblast adaptivního řízení s využitím vícemodelového přístupu. Budeme se věnovat adaptivnímu polohovému řízení elektromechanických aktuátorů, které můžeme ve velké míře nalézt u spalovacích motorů moderních automobilů. Zaměříme se na elektronickou škrticí klapku a EGR ventil. V dopravních prostředcích se tyto aktuátory sice vyskytují již několik desetiletí, ale jejich řízení stále představuje poměrně zajímavý vědecký problém. Tyto aktuátory vykazují silnou nelinearitu a jejich vlastnosti se mění s časem. Z hlediska řízení a regulace je neustále prostor na

zlepšení a můžeme se také zaměřit na tyto vlastnosti: kvalita regulace (statická odchylka, zpoždění, časová konstanta), energetická náročnost, adaptivita v případě změn parametrů soustavy, akční zásah s ohledem na životnost soustavy a jiné.

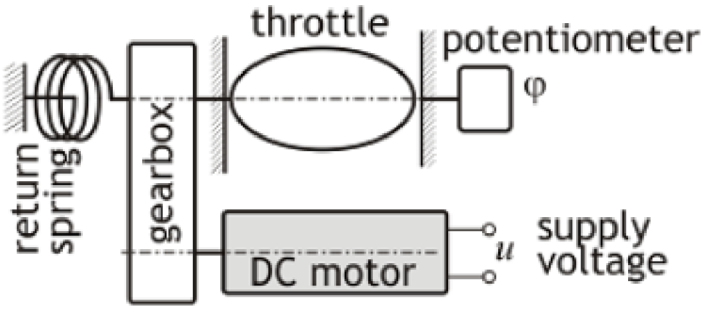
Při práci na adaptivním řízení se objevila další, poměrně ohraničená oblast, která byla nezbytná pro správnou funkci testovaného řízení a kterou jsme se poměrně důkladně zabývali. Touto oblastí bylo získávání derivací digitálního signálu, jež spadá do kategorie digitálního zpracování signálu.

Kapitola 2

Rešerše

2.1 Elektronická škrticí klapka a EGR ventil

Jako typického zástupce elektromechanického aktuátoru jsme zvolili elektronickou škrticí klapku a elektronický EGR(*Exhaust Gas Recirculation*) ventil. Elektronická škrticí klapka slouží k regulaci přívodu vzduchu do spalovacích motorů a poprvé se objevuje v sériových automobilech u výrobce BMW již v roce 1988. V současné době je elektronická škrticí klapka součástí téměř každého spalovacího motoru, který musí splňovat přísné emisní limity. Na obrázku 2.1 je základní schéma elektronické škrticí klapky.



Obrázek 2.1: Schéma elektronické škrticí klapky (převzato z [1])

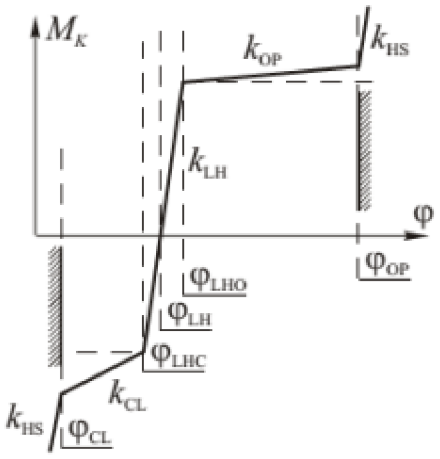
Systém elektronické škrticí klapky se skládá především z:

- stejnosměrného motoru
- převodovky
- motýlkového ventilu

- nelineární pružiny
- polohového senzoru

Elektronická škrticí klapka představuje polohový servomechanismus a je používána již mnoho let ve velkých objemech, přesto je její řízení stále otevřeným vědeckým problémem. Je to především z těchto důvodů:

- Charakteristika vratné pružiny je silně nelineární (obrázek 2.2).
- Mechanická část se vyrábí s důrazem na nízké výrobní náklady, a proto je přítomno velmi výrazné suché tření a parametry jednotlivých vyrobených kusů se mohou lišit.
- Jsou kladeny poměrně vysoké nároky na kvalitu regulace.



Obrázek 2.2: Nelineární charakteristika vratné pružiny (převzato z [1])

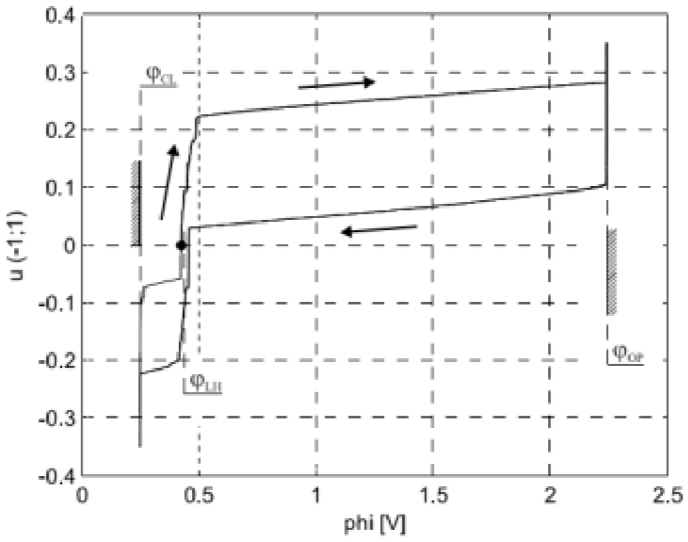
V grafu můžeme odlišit tři oblasti podle polohy φ:

- $\varphi_{CL} - \varphi_{LHC}$: oblast přivírání ventilu
- $\varphi_{LHC} - \varphi_{LHD}$: klidová oblast, s polohou φ_{LH}, což je klidová poloha ventilu bez napájení motoru
- $\varphi_{LHD} - \varphi_{OP}$: oblast otevírání ventilu

Oblasti pod φ_{CL} a nad φ_{OP} představují dorazy, kterých by při běžném chodu nemělo být dosáhnuto. Charakteristika pružiny je volena takto proto, aby při poruše na elektronické části škrticí klapky nebo při poruše senzoru

polohy zůstal bez napájení stejnosměrného motoru ventil částečně otevřen a umožnil nouzový chod motoru se sníženým výkonem. Tato poloha se nazývá *limp-home* a v textu bude dále označována zkratkou LH.

Vliv tření je patrný z kvazi statické charakteristiky na obrázku 2.3, kterou získáme, pokud budeme velmi pomalu (otevření – zavření v řádu desítek sekund) měnit vstupní napětí a projdeme celý rozsah otevření ventilu. Chování vykazuje hysterezi – vstupní napětí pro danou polohu je závislé na směru, kterým se pohybujeme. Rozdíl tohoto napětí představuje dvojnásobek hodnoty statického tření.



Obrázek 2.3: Kvazi statická charakteristika elektronické škrticí klapky (převzato z [1])

Zanedbáme-li dynamiku elektrické části stejnosměrného motoru, dorazy, vůli v převodovce a vliv proudícího vzduchu, můžeme elektronickou škrticí klapku vyjádřit pomocí následující rovnice:

$$I\ddot{\varphi} + b\dot{\varphi} + k(\varphi)\varphi + \tau(\dot{\varphi}) = u \tag{2.1}$$

kde u je vstupní napětí, I je zobecněný moment setrvačnosti, b viskózní tření, $k(\varphi)$ tuhost pružiny a $\tau(\dot{\varphi})$ je třecí moment. Všechny veličiny jsou vztaženy na hřídel ventilu.

EGR ventil je principiálně shodný s elektronickou škrticí klapkou, avšak slouží k regulaci průchodu spalin spalovacího motoru zpět do sání. Používá se opět z důvodu snižování emisí. Tato práce se nevztahuje na EGR ventily, které jsou založeny na jiném než elektronickém způsobu regulace pomocí stejnosměrného motoru.

2.2 Získávání derivací digitálního signálu

Algoritmus dopředného kompenzátoru vyžaduje ke své funkci informaci o poloze, rychlosti a zrychlení. Obecně můžeme říci, že algoritmus dopředného kompenzátoru vyžaduje minimálně informaci o regulované veličině a její derivace až do řádu, který odpovídá řádu modelu uvnitř kompenzátoru. Při reálné aplikaci je soustava vybavena většinou pouze sensorem regulované veličiny (v našem případě polohový sensor - potenciometr) a ostatní stavy musí být odhadovány.

Pokud bychom znali model soustavy, nabízí se jako přirozená možnost využít stavového pozorovatale, případně Kálmánova filtru. Pro náš případ adaptivního kompenzátoru toto ale nemůžeme použít, protože model soustavy nám není znám (model soustavy není znám na začátku regulace, ale procesem adaptace tento model získáváme). Proto musíme získávat derivace regulované veličiny přímou derivací digitálního signálu.

Frekvenční odezva ideálního derivace je:

$$H_{FB}(\omega) = j\omega, |\omega| < \pi \tag{2.2}$$

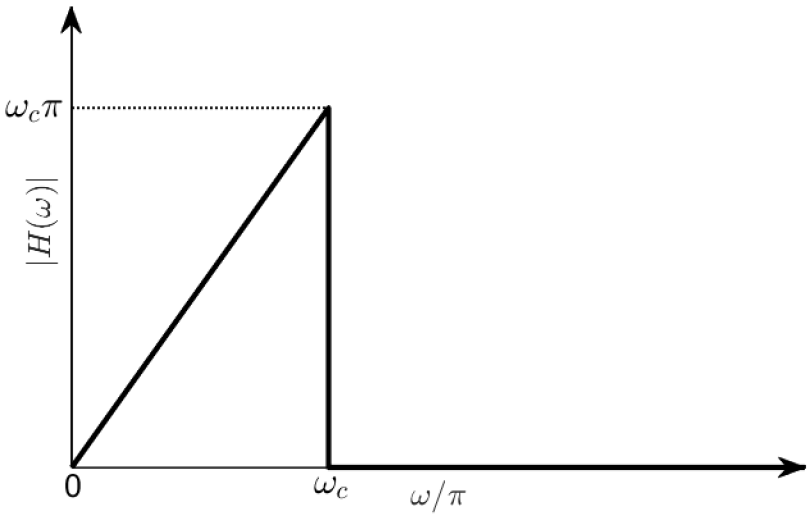
Zesílení se lineárně zvyšuje s frekvencí a maxima dosahuje pro Nyquistovu frekvenci (polovina vzorkovací frekvence, $\omega = \pi$). Měřený signál se skládá z užitečného signálu, často na nízkých frekvencích, a nežádoucího rušení a šumu, které jsou obecně přítomny v celém frekvenčním spektru. A právě vysoké zesílení derivace pro vyšší frekvence způsobuje, že zesílený šum výrazně znehodnotí původní signál. Kvůli tomuto důvodu je použití ideálního derivátoru (i kdyby byl realizovatelný) nežádoucí.

Pro tyto případy se používá takzvaný *low-pass differentiator*, jehož frekvenční odezva v ideálním případě je:

$$H_{LP}(\omega) = \begin{cases} j\omega, & |\omega| \leq \omega_c \\ 0, & \omega_c < |\omega| < \pi \end{cases} \tag{2.3}$$

kde ω_C je cutoff frekvence a užitečný signál by měl ležet v pásmu $\omega < \omega_C$. Ve skutečnosti není možno takto strmé charakteristiky (obrázek 2.4) dosáhnout, ale této ideální charakteristice se můžeme více či méně přiblížit. Existuje mnoho metod pro získávání derivací digitálního signálu a my se zaměříme na ty z našeho pohledu nejdůležitější, jež jsou:

- low-pass filtr a centrální diference
- Savitzky-Golay filtr
- Equiripple low-pass differentiator
- Maximally Flat Low-pass differentiator



Obrázek 2.4: A-F charakteristika ideální low-pass derivace

2.2.1 Low-pass filtr a centrální diference

Jako první možnost uvádíme kombinaci numerické derivace (centrální diference) a low-pass filtru. Tato metoda nevyžaduje žádné speciální znalosti metod pro digitální derivování signálu, přesto může poskytovat uspokojivé výsledky. Zvláště pokud je low-pass filtr navrhnut s ohledem na nejvyšší frekvenci užitečného signálu.

Signál je získán vzorkováním analogového signálu a je tak definován pomocí hodnot v ekvidistantních uzlových bodech. Nahradíme hodnotu derivace signálu hodnotou derivace interpolačního polynomu. V nejjednodušším případě budeme vždy mezi dvěma body interpolovat polynomem prvního řádu (přímkou) a získáme vztah:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} \tag{2.4}$$

který je také znám jako dopředná diference a kde h představuje krok metody. Dále můžeme použít vzorec pro centrální diferenci:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} \tag{2.5}$$

V případě digitálního zpracování signálu bývá krok h definován jako převrácená hodnota vzorkovací frekvence a vstup budeme označovat jako x_i. Potom

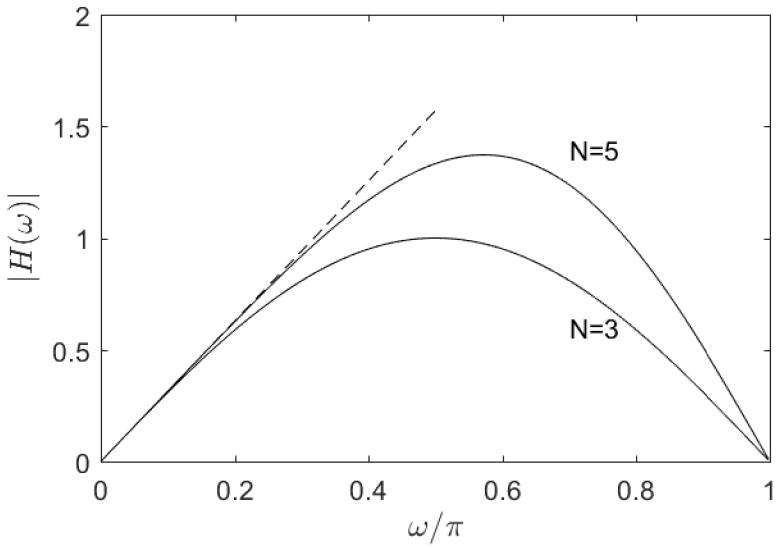
můžeme psát centrální diferenci pomocí obecného vzorce pro FIR filtr jako:

$$y(n) = \sum_{k=1}^{(N-1)/2} c_k(x_{n+k} - u_{x-k}) \tag{2.6}$$

kde N je řád metody a c_k jsou koeficienty (impulsní odezva) filtru. Frekvenční odezva tohoto filtru je dána:

$$H(\omega) = 2i \sum_{k=1}^{(N-1)/2} w_k \sin(k\omega)) \tag{2.7}$$

Z grafu frekvenční odezvy (obrázek 2.5) vidíme, že v oblasti nízkých frekvencí je frekvenční odezva centrální diference shodná s ideální derivací (čárkovaně), ale v oblasti vyšších frekvencí dochází k útlumu. V případě zpracování reálného signálu, který obsahuje šum, je ale tento útlum žádoucí, jinak by byl šum příliš zesilován. Při reálné implementaci je často i tento útlum na vysokých frekvencích nedostatečný a zesílený šum většinou znehodnotí vlastní signál. Proto obvykle bývá centrální diference součástí kaskády, ve které je dále přítomen low-pass filtr.

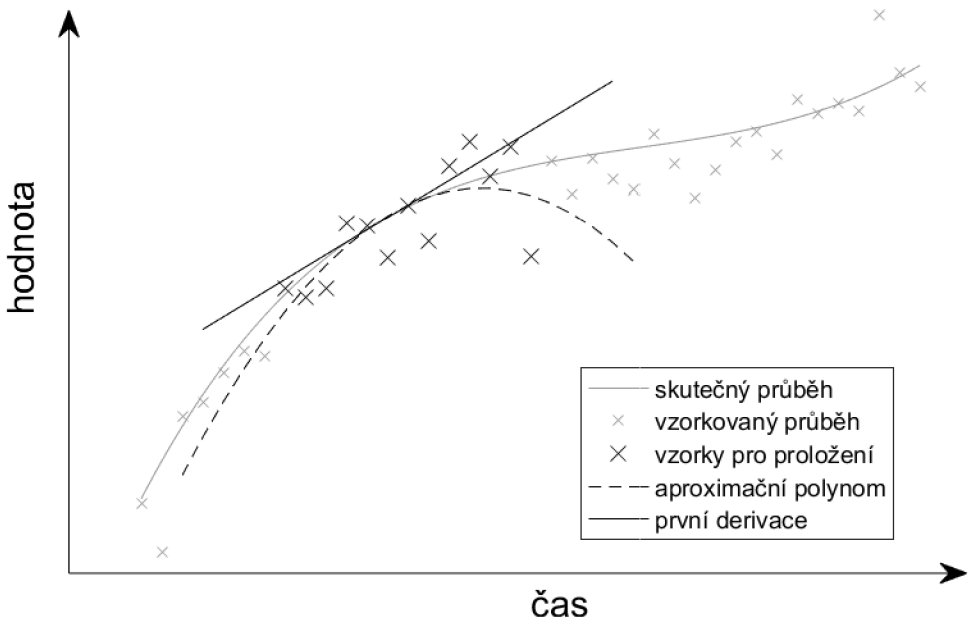


Obrázek 2.5: A-F charakteristika centrální diference pro 3 a 5 bodů

Připomínáme, že centrální diference a filtrace jsou lineární operace a tedy nezáleží na jejich pořadí. Konvolucí impulsní odezvy centrální diference a impulsní odezvy filtru získáme koeficienty nového filtru, která provede obě operace.

2.2.2 Savitzky-Golay filtr

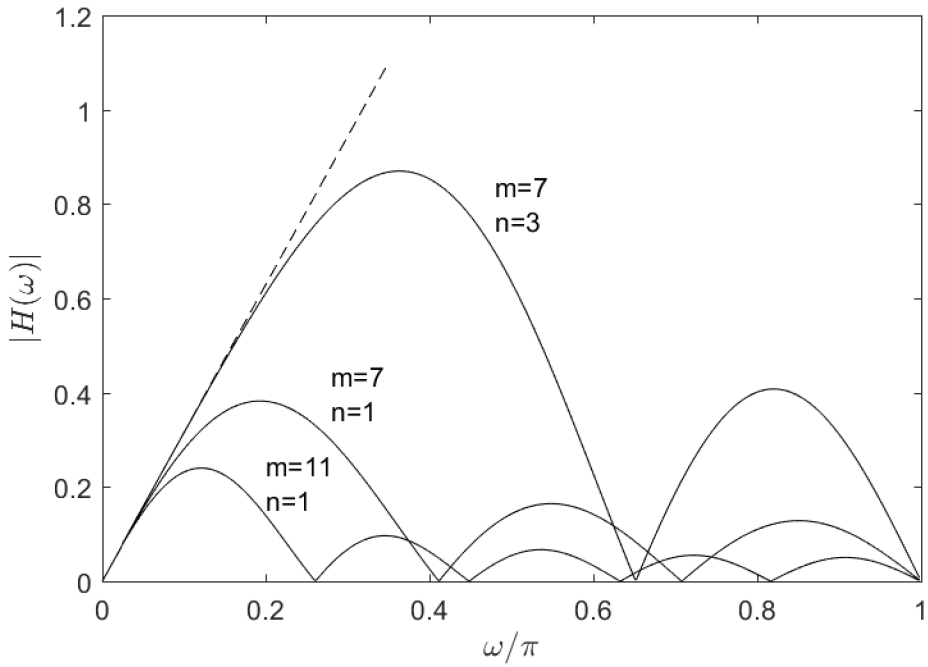
Savitzky-Golay filtr byl poprvé popsán chemiky Abrahamem Savitzky a Marcelem Golay již v roce 1965 [2]. Tento filtr je založen na polynomiální regresi s využitím metody nejmenších čtverců. Idea spočívá v proložení několika posledních uzlových prvků signálu polynomem určitého řádu a odhad derivace poté určíme pomocí derivace tohoto polynomu. Princip této metody je znázorněn při aproximaci polynomem prvního řádu na obrázku 2.6.



Obrázek 2.6: Princip Savitzky-Golay filtru (převzato z [3])

Důležité je, že nemusíme v každém bodě znovu počítat metodou nejmenších čtverců koeficienty aproximačního polynomu, protože koeficienty filtru jsou konstantní, pokud je konstantní počet prvků filtru, řád aproximačního polynomu a poloha bodu ve kterém zjišťujeme derivaci vůči aproximačnímu oknu. Filtrace tedy probíhá pomocí konvoluce jako pro obecný FIR filtr. Ve většině případů volíme polohu bodu ve kterém zjišťujeme derivaci v polovině aproximačního okna a tím získáme filtr s lineární fází (impulsní odezva je symetrická).

Na obrázku 2.7 je znázorněna frekvenční odezva Savitzky-Golay filtru pro několik variant filtru lišících se počtem prvků filtru a řádem aproximačního polynomu. Ve všech případech se jedná o filtr s lineární fází.



Obrázek 2.7: A-F charakteristika Savitzky-Golay filtru (převzato z [3])

2.2.3 Equiripple filtr

Název této kategorie filtrů vyplývá z faktu, že tyto filtry vykazují stejné amplitudy zvlnění (*equal-ripples*) v pásmu propouštění (*passband*) i zadržení (*stopband*) signálu. Občas se také tyto filtry nazývají Parks-McClellan filtr podle autorů, kteří poprvé publikovali efektivní algoritmus pro návrh těchto filtrů [4].

Můžeme vyjádřit rozdíl mezi A-F odezvou ideálního filtru a jeho praktickou realizací jako chybovou funkci:

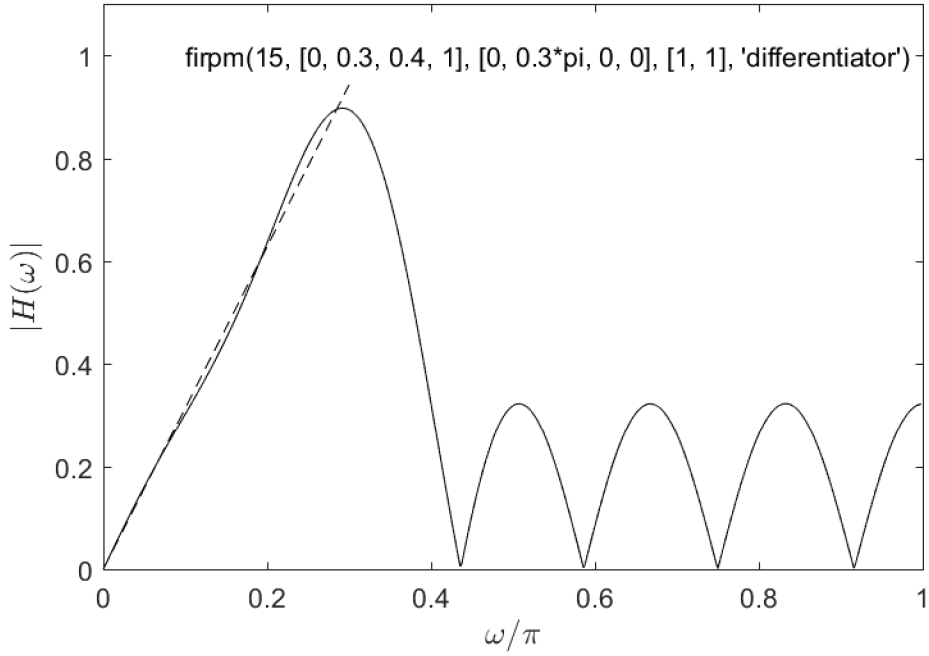
$$|E(\omega)| = W(\omega)[H_D(\omega) - H(\omega)] \tag{2.8}$$

kde $H_D(\omega)$ je žádaná amplitudo-frekvenční (A-F) odezva, $H(\omega)$ je skutečná A-F odezva a $W(\omega)$ je váhová funkce, která může být různá pro rozdílná frekvenční pásma. Návrhová metoda pro Equiripple filtr hledá během optimalizace takovou frekvenční odezvu, která minimalizuje maximální hodnotu chybové funkce 2.8 přes *passband* a *stopband*. Matematicky můžeme vyjádřit jako:

$$\min[\underset{\omega\in\phi}{max}|E(\omega)|] \tag{2.9}$$

kde ϕ značí *passband* a *stopband*. Bylo dokázáno, že minimalizace 2.9 vede na stejné amplitudy zvlnění [5].

Ukázka A-F charakteristiky je zobrazena na obrázku 2.8. Tento filtr byl navrhnut pomocí funkce *firpm* v prostředí MATLAB s parametry zobrazenými v obrázku. Za poznámku stojí, že na rozdíl od Savitzky-Golay filtru a Maximally Flat filtru vykazuje frekvenční odezva zvlnění i v oblasti *passband*.



Obrázek 2.8: A-F charakteristika equiripple derivátoru (převzato z [3])

2.2.4 Maximally flat filtr

Kategorie maximally flat filtrů byla poprvé představena O. Herrmannem[6] na začátku sedmdesátých let dvacátého století. Jak název napovídá, A-F charakteristika těchto filtrů je hladká v oblasti *passband* i *stopband* a nevykazuje typické zvlnění přítomné u většiny FIR filtrů. Koeficienty tohoto filtru lze nalézt v uzavřené podobě.

Pro derivaci prvního řádu použijeme implementaci [7], která uvažuje tato omezení na amplitudo frekvenční odezvu:

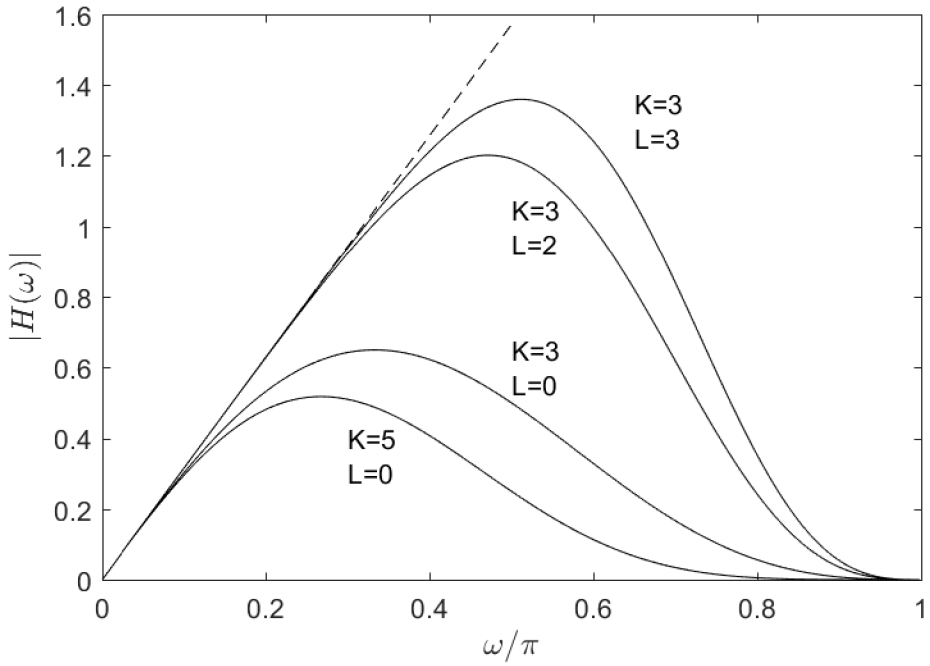
$$|H_{MF}(\omega)| = 0, \omega = 0 \tag{2.10}$$

$$\frac{\mathrm{d}}{\mathrm{d}\omega}|H_{MF}(\omega)| = 1, \omega = 0 \tag{2.11}$$

$$\frac{\mathrm{d}^k}{\mathrm{d}\omega^k}|H_{MF}(\omega)| = 0, \omega = 0, 2 \leq k \leq 2L \tag{2.12}$$

$$\frac{\mathrm{d}^k}{\mathrm{d}\omega^k}|H_{MF}(\omega)| = \pi, \omega = 0, 2 \leq k \leq K \tag{2.13}$$

kde $H_{MF}(\omega)$ je frekvenční odezva, $2L$ a K vyjadřují až do kterého řádu jsou derivace shodné s ideální charakteristikou v okolí nulové frekvence a v okolí Nyquistovy frekvence ($\omega = \pi$). Ideální charakteristika má v okolí nulové frekvence první derivaci rovnou jedné, ostatní derivace včetně nulté jsou rovny nule. V oblasti Nyquistovy frekvence požadujeme, aby všechny derivace včetně nulté byly rovny nule.



Obrázek 2.9: A-F charakteristika maximally flat derivátoru (převzato z [3])

Pro výpočet koeficientů tohoto filtru použijeme funkci *lowdiff* [8]. Tato funkce očekává dva parametry K a L a vrací koeficienty filtru. Délka impulsní odezvy tohoto filtru je $K + 2L + 2$. Pro liché K získáme FIR filtr typu III a pro sudé K získáme FIR filtr typu IV.

Při návrhu filtru touto metodou nemůžeme přesně specifikovat cutoff frekvenci, místo toho musíme zvolit konstanty K a L, spočítat koeficienty filtru

a vykreslit A-F charakteristiku. Tento postup musíme opakovat do té doby, než získáme žádanou A-F charakteristiku. Na obrázku 2.9 vidíme A-F charakteristiky pro různé konstanty K a L.

Za zmínku stojí, že pro $L = 0$ mají koeficienty filtru společného dělitele, který je převrácenou hodnotou mocniny dvou. Příklad pro $K = 5$ a $L = 0$:

$$c(n) = [\frac{1}{32}, \frac{4}{32}, \frac{5}{32}, 0, -\frac{5}{32}, -\frac{4}{32}, -\frac{1}{32}] = \frac{1}{2^5}[1, 4, 5, 0, -5, -4, -1] \quad (2.14)$$

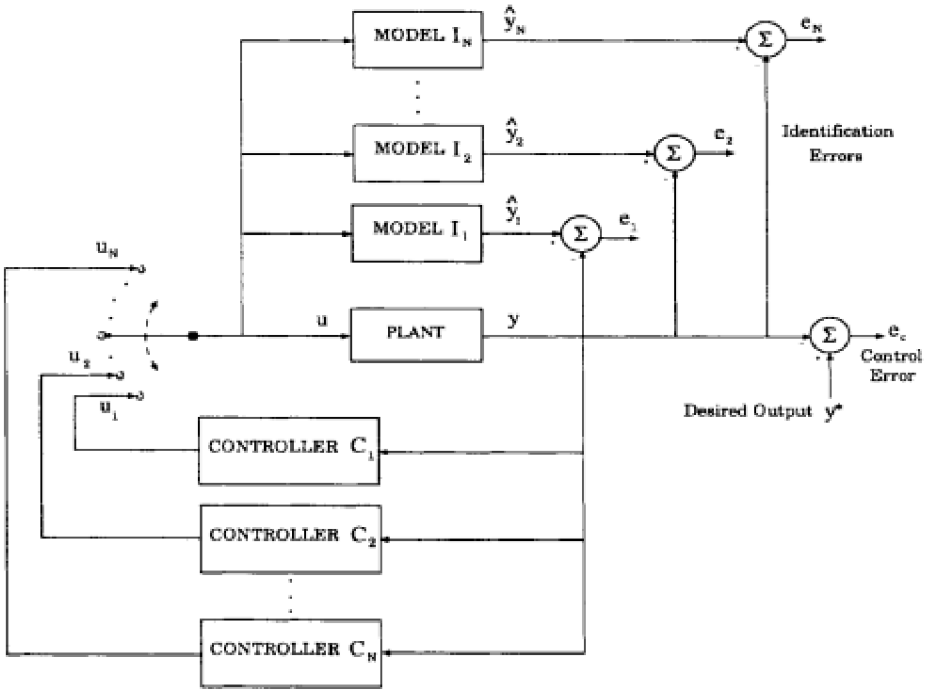
Z toho vyplývá, že při praktické implementaci v HW můžeme nahradit obecné násobení konstantou bitovým posunem a tím bude tento algoritmus velmi málo náročný na výpočet, z čehož můžeme těžit hlavně při implementaci na různé embedded systémy (založené na mikrokontroleru případně FPGA). Také se vyhneme zaokrouhlovacím chybám při výpočtu.

2.3 Více-modelový přístup pro regulaci

První zmínky o využití více-modelového přístupu pro řízení se objevují v 70-tých letech 20. století [9][10]. Od té doby se tento přístup rozvíjel a vzniklo mnoho praktických implementací [11][12][13].

Základní myšlenkou více-modelového přístupu je, že model daného systému je neznámý, ale náleží známé konečné množině modelů. Jinými slovy, každý člen množiny modelů může reprezentovat dynamiku daného systému, ale pouze jeden z nich je ten platný. Problém se tak redukuje na úkol najít, který z dané množiny modelů nejlépe reprezentuje aktuální systém. Konkrétní provedení jednotlivých implementací více-modelového řízení se mohou lišit tím, jak jsou vyjádřeny jednotlivé modely a také algoritmem, který přepíná mezi jednotlivými modely a regulátory.

Obecné schéma pro zpětnovazební řízení můžeme vidět na obrázku 2.10. Zde slouží N paralelních modelů pro identifikaci parametrů soustavy a výstup identifikační chyba slouží k tomu, abychom mohli určit, který model je aktuálně platný. Pro vlastní řízení slouží N regulátorů, jejichž parametry jsou závislé na odpovídajících identifikačních modelech.



Obrázek 2.10: Obecné schéma více-modelového řízení (převzato z [14])

2.4 Multiple model adaptive control dle [15]

Pro hlubší zkoumání jsme si vybrali více-modelové adaptivní řízení dle [15], protože by mělo mít tyto klíčové vlastnosti:

- je plně adaptivní (adaptivní modely a adaptivní rozhodovací algoritmus)
- je v souladu se stochastickým přístupem
- využívá duální řízení
- jednotlivé modely jsou vyjádřeny pomocí lineárních modelů

Nyní podrobněji popíšeme klíčové části:

2.4.1 Lokální modely

Lokální modely jsou uvažovány pomocí diferenční rovnice ve tvaru:

$$y(k) = \mathbf{a}_{m(k)}^T \mathbf{x}(k-1) + b_{m(k)} u(k-1) + d_{m(k)} + e(k) \qquad (2.15)$$

kde $y(k)$ je výstup, $u(k)$ je vstup, $\mathbf{x}(k) = [y(k), ..., y(k-n+1), u(k-1), ..., u(k-p)]^T$ je $n+p$ dimenzionální stavový vektor a $e(k)$ je aditivní bílý šum. Parametry $\mathbf{a}_i := [a_1, a_2, ..., a_{n+p}]^T$, b_i a d_i jsou definovány pro každý model $i \in \{1, ..., H\}$, kde H je počet modelů.

Spojení všech lokálních modelů můžeme vyjádřit jako sumu lokálních modelů s využitím indikační proměnné λ_i:

$$y(k) = \sum_{i=1}^{H} \lambda_i[u(k-1), \mathbf{x}(k-1)]\mathbf{w}_i^{*T}\Phi(k-1) + e(k) \tag{2.16}$$

kde:

$$\Phi(k) = [u(k), \mathbf{x}^T(k), 1] \tag{2.17}$$

$$\mathbf{w}_i^* = [b_i, \mathbf{a}_i^T, d_i]^T \tag{2.18}$$

$$\lambda_i[u(k), \mathbf{x}(k-1)] = \begin{cases} 1, & \text{jestliže } m(k) = i \\ 0, & \text{jinak} \end{cases} \tag{2.19}$$

Skutečné parametry jednotlivých lokálních modelů $\{\mathbf{w}_i^*\}_{i=1}^H$ a také indikační proměnná $\{\lambda_i\}_{i=1}^H$ nejsou známy a budou nahrazeny jejich odhadem.

V případě lokálních modelů budeme jejich odhad značit $\hat{\mathbf{w}}_i^* = [\hat{b_i}, \hat{\mathbf{a}}_i^T, \hat{d_i}]^T$ a indikátorová proměnná bude nahrazena odhadem pomocí tzv. hodnoticí funkce $\hat{\lambda}_i[u(k-1), \mathbf{x}(k-1)]$, pro kterou platí:

$$0 < \hat{\lambda}_i < 1, \sum_{i=1}^{H} \hat{\lambda}_i = 1 \tag{2.20}$$

Tím vznikne modulární síť, která představuje náš identifikační model:

$$\hat{y}(k) = \sum_{i=1}^{H} \hat{\lambda}_i[u(k-1), \mathbf{x}(k-1)]\hat{y}_i(k) \tag{2.21}$$

kde:

$$\hat{y}_i(k) = \hat{\mathbf{w}}_i^T\Phi(k-1) \tag{2.22}$$

Protože se jedná o část adaptivního řízení, budou parametry lokálních modelů $\hat{\mathbf{w}}_i$ a hodnotící funkce $\hat{\lambda}_i$ rekurzivně odhadovány on-line během provozu. V případě parametrů lokálních modelů, budou tyto odhadovány pomocí Kálmánova filtru.

2.4.2 Adaptivita lokálních modelů

Během aktivity konkrétního lokálního modelu i můžeme rovnici 2.16 přepsat pomocí stavového zápisu jako:

$$\begin{aligned} \mathbf{w}_i^*(k+1) &= \mathbf{w}_i^*(k) \\ y(k) &= \mathbf{w}_i^{*T}\Phi(k-1) + e(k) \end{aligned} \tag{2.23}$$

Jelikož je tato rovnice lineární v parametrech a u šumu $e(k)$ předpokládáme Gaussovo rozdělení, můžeme využít Kálmánův filtr na odhad parametrů konkrétního modelu:

$$\begin{aligned} \hat{\mathbf{w}}_i(k|k-1) &= \hat{\mathbf{w}}_i(k-1|k-1) \\ \mathbf{P}_i(k|k-1) &= \mathbf{P}_i(k-1|k-1) \end{aligned} \tag{2.24}$$

a

$$\begin{aligned} S_i &= \mathbf{\Phi}^T(k-1)\mathbf{P}_i(k|k-1)\mathbf{\Phi}(k-1) + R \\ \mathbf{K}_i(k) &= \mathbf{P}_i(k|k-1)\mathbf{\Phi}(k-1)/S_i \\ \hat{\mathbf{w}}_i(k|k) &= \hat{\mathbf{w}}_i(k|k-1) + \mathbf{K}_i(k)(y(k) - (\hat{y})_i(k)) \\ \mathbf{P}_i(k|k) &= \{\mathbf{I} - \mathbf{K}_i\mathbf{\Phi}^T(k-1)\}\mathbf{P}_i(k|k-1) \end{aligned} \tag{2.25}$$

Rovnice 2.24 představuje predikci Kálmánova filtru a rovnice 2.25 korekci. Za pozornost stojí, že v predikční části Kálmánova filtru se model ani kovarianční matice nemění a tím se stává tento odhad ekvivalentní s metodou rekurzivních nejmenších čtverců [16].

2.4.3 Hodnotící funkce

Hodnotící funkce vyjadřují pravděpodobnost, že daný mód je ten odpovídající skutečné soustavě na základě předchozích vstupů a stavu. Hodnotící funkce představuje nelineární mapování z prostoru $[u(k-1), \mathbf{x}(k-1)]$ na interval $\langle 0, 1\rangle$. Toto mapování je neznámé a bude odhadováno on-line jako součást adaptačního procesu s využitím tzv. *gating network*. V publikaci [15] je uvažováno použití dvou *gating network*:

Softmax gating network

Tato sít je převzata z *Mixture of Experts* architektury představené v [17] a je založena na parametrizování *softmax* funkce. Hodnoticí funkce potom bude vypadat takto:

$$\hat{\lambda}_i[u(k-1), \mathbf{x}(k-1)] = \frac{e^{\mathbf{h}_i^T\phi(k-1)}}{\sum_{j=1}^{H} e^{\mathbf{h}_j^T\phi(k-1)}} \tag{2.26}$$

kde $\{\mathbf{h}\}_{i=1}^{H}$ je vektor, jímž je parametrizována hodnotící funkce a podléhá on-line odhadu během procesu adaptace.

Gaussian mixture kernel gating network

Je založena na směsi Gaussových rozdělení pravděpodobnosti (*Gaussian Mixture Model*). Hodnotící funkce je dána vztahem:

$$\hat{\lambda}_i[u(k-1), \mathbf{x}(k-1)] = \frac{\mathbf{g}^T \theta_i}{\sum_{j=1}^{H} \mathbf{g}^T \theta_j} \tag{2.27}$$

kde $\mathbf{g} := [g_1, g_2, \cdots, g_\zeta]^T$ a $\theta_i := [\theta_{i_1}, \theta_{i_2}, \cdots, \theta_{i_\zeta}]^T$. Vektor $\mathbf{q}$ představuje ζ Gaussových jader modelu směsi. Tato jádra jsou definována pomocí střední hodnoty μ_j a rozptylu σ_j, které jsou pevné a musí být definovány dopředu tak, aby Gaussova jádra vhodně pokrývala celý prostor, kde se může ϕ nalézat. Podobně jako je tomu u neuronových sítí typu RBF.

$$g_j = \frac{exp(-\frac{(\tilde{\phi}(k-1)-\mu_j)^T(\tilde{\phi}(k-1)-\mu_j)}{2\sigma_j^2})}{(2\pi\sigma_j^2)^{0,5(n+p+1)}} \tag{2.28}$$

Vektor θ_i obsahuje koeficienty směsí, které jsou definovány ve smyslu *softmax* funkce ve tvaru:

$$\theta_{i_l} = \frac{e^{(h_{i_l})}}{\sum_{l=1}^{\zeta} e^{(h_{i_l})}} \tag{2.29}$$

Vektor $\{\mathbf{h}\}_{i=1}^{H}$ potom podléhá procesu adaptace.

2.4.4 Výpočet akční veličiny

Vyjdeme-li z předpokladů, že model soustavy je znám (odhad), model soustavy je lineární a hodnotící funkce je také známa (její odhad), byl by přirozenou volbou výpočet akčního zásahu pro lokální mód i dle:

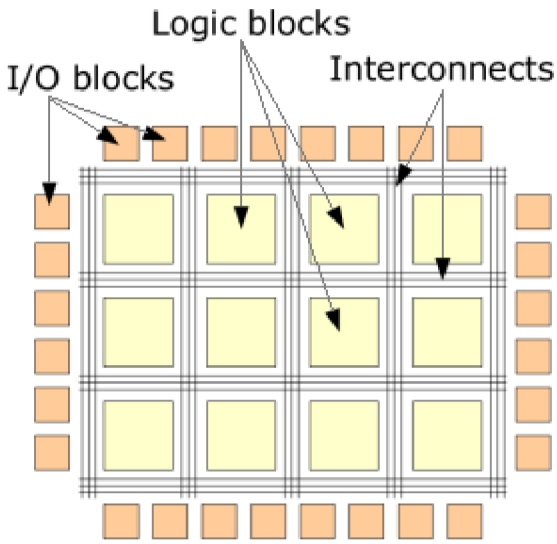
$$\hat{u}_i(k) = \frac{y_d(k+1) - \hat{\mathbf{a}}_{m(k+1)}^T \mathbf{x}(k) - \hat{d}_{m(k+1)}}{\hat{b}_{m(k+1)}} \tag{2.30}$$

V publikaci [15] je také uveden výpočet akční veličiny pomocí duálního řízení, ale nabízí se i jiné možnosti. Při výpočtu celkového akčního zásahu pomocí akčních zásahů pro jednotlivé módy je možno postupovat dvěma způsoby. Celkový akční zásah je brán jako suma jednotlivých akčních zásahů lokálních módů vážených hodnoticími funkcemi anebo se pro celkový akční zásah zvolí pouze ten lokální mód, který má nejvyšší hodnotu hodnotící funkce.

2.5 Programovatelná hradlová pole - FPGA

Programovatalná hradlová pole, anglicky **F**ield **P**rogrammable **G**ate **A**rray - FPGA jsou polovodičové čipy založené na matici konfigurovatelných logických bloků vzájemně propojovaných konfigurovatelnými propoji tak, aby čip prováděl danou funkci. Anglická zkratka se skládá se dvou sousloví:

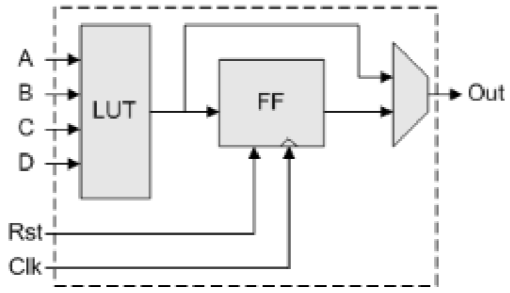
- *Field Programmable* - programovatelné v poli (koncovým uživatelem)
- *Gate Array* - matice hradel



Obrázek 2.11: Struktura FPGA (převzato z[18])

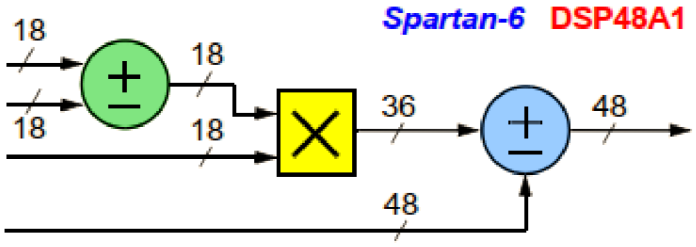
Základním stavebním blokem FPGA jsou konfigurovatelné logické bloky, které se skládají z jedné nebo více tzv. *slice*. Každá *slice* (obr.: 2.12) obsahuje jednu nebo několik:

- programovatelnou n-bitovou look-up tabulku (LUT) - umožňuje tvořit kombinační logiku
- klopný obvod typu D - jednobitová paměť - umožňuje tvořit sekvenční logiku
- programovatelný multiplexer - přemostění datových signálů
- na obrázku 2.12 nejsou zobrazeny, ale každá slice může dále obsahovat například multiplexery pro routování hodinového signálu, dodatečné signály a hradla pro efektivní tvorbu sčítaček a násobiček, atd.



Obrázek 2.12: Zjednodušená struktura slice (převzato z[18])

Skrz celou strukturu FPGA vede síť, která obstarává propojování signálů mezi jednotlivými konfigurovatelnými logickými bloky - globální propojovací matice. Před každým konfigurovatelným logickým blokem je lokální propojovací matice, která slouží k mapování signálů z globalní propojovací matice na jednotlivé vstupy konfigurovatelných logických bloků a *slice*. Obě tyto propojovací matice jsou plně konfigurovatelné.



Obrázek 2.13: DSP48 jednotka - vestavěná násobička (převzato z[18])

V průběhy let se ve struktuře FPGA objevily ještě dva prvky, které jsou součástí každého moderního FPGA:

- vestavěné násobičky - v terminologii firmy Xilinx jsou nazývány *DSP48 slice*. Operace násobení je poměrně často používaná a její implementace vyžaduje velké množství LUT. Množství obsazených LUT narůstá přibližně s druhou mocninou bitové šíře operandů. Proto byly vytvořeny tyto bloky, které v závislosti na třídě FPGA mohou kromě násobičky obsahovat také akumulátor nebo před-sčítačku.

  Na obrázku 2.13 je znázorněno schéma DSP48 bloku včetně znázornění bitových šíří jednotlivých signálů pro FPGA Spartan-6. Podobných DSP48 bloků mohou být ve struktuře FPGA desítky až tisíce.

- bloky RAM pamětí - kromě násobení se ukázalo, že další často používaná funkce je uložení dat v paměti. Pokud by byla RAM (**R**andom **A**ccess **M**emory) paměť realizována pomocí jednobitových registrů, bylo by jich potřeba velmi mnoho. Z toho důvodu se v FPGA objevují bloky RAM pamětí.

Výsledná funkcionalita FPGA je daná konfigurací globální propojovací matice, lokálních propojovacích matic, nastavení LUT a vnitřních multiplexerů. Základem pro konfiguraci FPGA jsou jednoduché grafické editory, které obsahují základní logická hradla nebo pokročilejší logické bloky (např.: čítač, demultiplexer, akumulátor). Nejběžnějším nástrojem pro konfiguraci (programování) FPGA jsou tzv. HDL jazyky (***H**ardware **D**esriction **L**anguage*), z nichž nejznámější jsou VHDL a Verilog. Dále existují pro programování FPGA vysokoúrovňové nástroje, mezi které patři:

- **System Generator for DSP - Xilinx** [19] - Knihovna pro Simulink, která obsahuje velké množství základních funkcí, ze kterých je možno v prostředí Simulinku generovat konfiguraci pro většinu FPGA od výrobce Xilinx.
- **LabVIEW FPGA Module - National Instruments** [20] - tvorba kódu pro HW založený na *RIO*[21] technologii.
- **HDL Coder - MathWorks** [22] - umožňuje generovat VHDL nebo Verilog kód z funkcí MATLABu, Simulink modelů a StateFlow diagramů.

Kapitola 3

Formulace problémů a cílů řešení

V pojednání k této disertační práci byly stanoveny tyto cíle:

1. Přeformulování metody více-modelového řízení uvedené v publikaci [15] pro použití jako dopředný kompenzátor

2. Úprava a optimalizace algoritmu této metody pro běh na *real-time* HW a experimentální ověření

3. Implementace filtrace a předzpracování signálu senzoru na FPGA

Počátečním impulsem pro vznik této práce byla zahraniční stáž na University of Malta pod vedením Prof. Ing. Simon G Fabri. Cílem stáže bylo testovat nové metody pro adaptivní řízení elektromechanických aktuátorů. Zaměřili jsme se na metodu *Multiple Model Adaptive Control of Spatial Multimodal Systems* z publikace[15], která slibovala zajímavé výsledky.

Ovšem během implementace této metody pro řízení elektromechanických aktuátorů tato metoda selhávala a nebyla úspěšná. Konkretní důvody jsou uvedeny v kapitole 4. Hlavním cílem práce je přeformulování a úprava této metody tak, aby byla vhodná pro řízení elektromechanických aktuátorů včetně experimentálního ověření. V kapitole 4 jsou popsány provedené modifikace, prezentovány různé poznatky a simulační a experimentální ověření.

V pojednání k této dizertační práci jsme si stanovili dílčí cíl porovnat filtraci a derivování digitálního signálu na FPGA s ohledem na využití pro dopředný kompenzátor. Objevili jsme tzv. *Smooth noise robust diffrentiator* [23], který sliboval zajímavé parametry ve srovnání s běžným přístupem získávání derivací digitálního signálu. Za běžný způsob získávání derivací digitálního signálu považujeme použití low-pass filtru a centrální diference,

případně pokročilejší způsob pomocí Savitzky-Golay filtru [2]. Významná výhoda *Smooth noise robust diffrentiatoru*, především ve spojitosti s implementací na FPGA je ta, že koeficienty filtru (impulsní odezva) obsahují pouze celočíselné koeficienty. Z toho plyne jednoduchá implementace na FPGA.

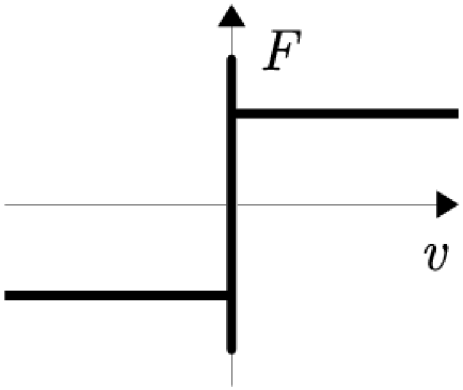
Při hlubším studiu oblasti digitálního zpracování signálu - digitálních derivátorů jsme narazili na mnoho algoritmů návrhu filtrů/derivátorů a zjistili jsme, že se jedná o poměrně zajímavou oblast, která je často z pohledu lidí věnujících se návrhu regulačních algoritmů přehlížena. Avšak vyžaduje-li regulační algoritmus znalost derivací signálu, má tato vlastnost klíčový vliv na kvalitu regulace a funkci celého algoritmu. Jako výsledek studia této oblasti vznikl článek, který shrnuje dle nás nejznámější způsoby návrhu filtrů/derivátorů, stručně popisuje způsob jejich návrhu a snaží se je porovnat. Článek [3] je v současné době připravován pro publikaci.

Při studiu této oblasti jsme zjistili, že filtr *Smooth noise robust diffrentiator* je vlastně již dlouho známý pod kategorií *maximally-flat* filtrů, která vznikla již v roce 1971[6]. Při implementaci na FPGA se tedy nevěnujeme tomuto filtru, ale kategorii *maximally-flat* filtrů, z nichž implementace dle [7] generuje při určitých vstupních parametrech stejné koeficienty jako *Smooth noise robust diffrentiator*.

Kapitola 4

Dopředný adaptivní kompenzátor založený na vícemodelovém přístupu

Při aplikaci vícemodelového zpětnovazebního řízení dle [15] pro regulaci elektromechanických aktuátorů jsme narazili na zásadní problém, a tím je suché tření. Suché tření představuje dominantní pasivní odpor v námi testovaných aktuátorech a můžeme si ho přiblížit pomocí grafu na obrázku 4.1.



Obrázek 4.1: Coulombův model suchého tření s efektem ulpívání

Vodorovná osa představuje rychlost vzájemného pohybu dvou povrchů a na svislé ose vidíme sílu, která mezi těmito povrchy působí. V oblasti ulpívání, okolí nulové rychlosti je tato závislost silně nelineární. Pokud chceme dva povrchy, mezi nimiž působí suché tření rozpohybovat, musíme působit silou, která je větší než statické tření. Jakmile překonáme tuto hranici, povrchy se

od sebe odtrhnou a velikost síly, která musí při pohybu působit, se sníží na úroveň kinematického tření. Zpětnovazební řízení dle [15] nemá jak tuto silně nelineární závislost postihnou a v okolí nulové rychlosti selhává jak regulace, tak adaptivita jednotlivých modelů.

Dalším, neméně závažným problémem je to, že jednotlivé lokální modely jsou vyjádřeny ve formě podobné ARX modelu. Polohový signál z řízené soustavy však obsahuje poměrně mnoho aditivního šumu, což způsobuje vychýlení odhadu parametrů modelu [24] a posléze selhání celé regulace.

Tyto problémy nás vedly k úvaze použít výše zmíněnou strukturu ne přímo pro zpětnovazební regulaci, ale jako dopředný kompenzátor, kde by výše zmíněné problémy nemusely tolik ovlivňovat kvalitu řízení. Drobné nepřesnosti modelu pokryje „slabý“ PID regulátor.

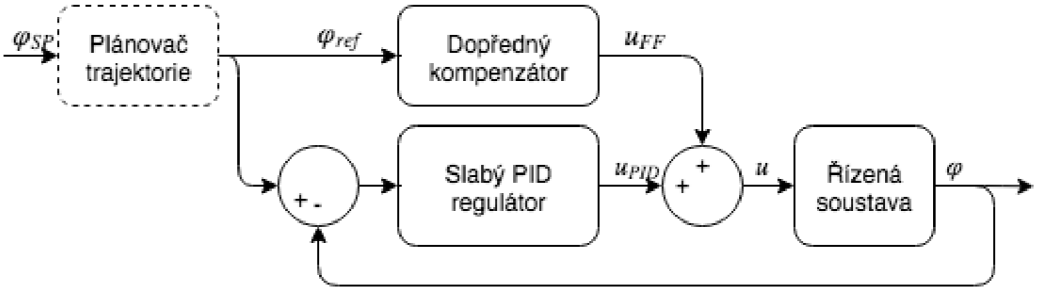
4.1 Přeformulování metody [15] pro použití jako dopředný kompenzátor

Schéma kompozitního řízení se zpětnovazební mregulátorem a dopřednou kompenzací je na obrázku 4.2. Čárkovaně je zobrazen blok *plánovač trajektorie*, který vhodným způsobem modifikuje vstupní signál - žádanou veličinu. Tento blok není principiálně nutný, ale je často vhodný. Situace, kdy je tento blok potřebný, může představovat skok, případně rychlé změny v signálu žádané veličiny. Typická řízená soustava je dynamický systém[1] a je vhodné již s ohledem na řízenou soustavu a maximální hodnotu akční veličiny vhodně omezit maximální rychlost změny žádaného signálu. V nejjednodušším případě může být tohoto dosaženo pomocí filtru typu dolní propust.

Chceme-li použít řízení [15] pro dopřednou kompenzaci, bude hlavní rozdíl spočívat v tom, že jednotlivé lokální modely nebudou sloužit pro výpočet akčního zásahu pomocí regulátoru, ale pro výpočet dopředné kompenzace na základě referenčních vstupů. Dojte tedy ke změně, zjednodušení poslední etapy celého algoritmu. Další drobná změna bude ve vyjádření jednotlivých modelů. V původní publikaci bylo vyjádření viz rovnice 2.15. Pro dopředný kompenzátor přeformulujeme rovnici tak, že výstupem z modelu bude vstup do soustavy a ostatní členy budou vstupy (rovnice 4.1).

$$u(k) = \mathbf{a}_{m(k)}^T \mathbf{x}(k-1) + d_{m(k)} \tag{4.1}$$

[1] Představme si setrvačník, jehož rychlost otáčení je řízená veličina a na setrvačník působíme silou. Žádáme-li skokovou změnu rychlosti, museli bychom působit nekonečnou silou, což přirozeně není možné. Stejný problém nastává při sice spojité změně rychlosti, ale velmi rychlé. Museli bychom působit sice konečnou, ovšem velkou silou, která může být za možnostmi aktuátoru



Obrázek 4.2: Blokové schéma kompozitního řízení s dopřednou kompenzací

kde $u(k)$ je vstup do soustavy, $\mathbf{x}(k) = [y(k), ..., y(k-n+1), u(k), ..., u(k-p+1)]^T$ je $n+p$ dimenzionální stavový vektor a $\mathbf{a}_{m(k)}^T$ a $d_{m(k)}$ jsou parametry modelu. Ostatní části, které se týkají lokálních modelů, jejich adaptivity a hodnoticí funkce, která vyhodnocuje aktuálně platný model, mohou zůstat stejné.

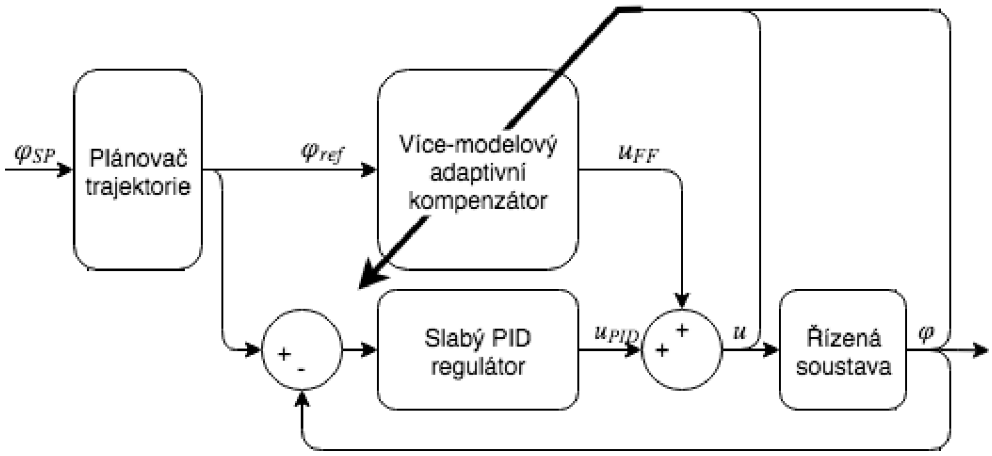
V publikaci [15] jsou jednotlivé lokální modely vyjádřeny pomocí diskrétního popisu. Obecně ale můžeme lokální modely také vyjádřit pomocí spojitého popisu. Popíšeme obě dvě možnosti:

Diskrétní vyjádření lokálních modelů

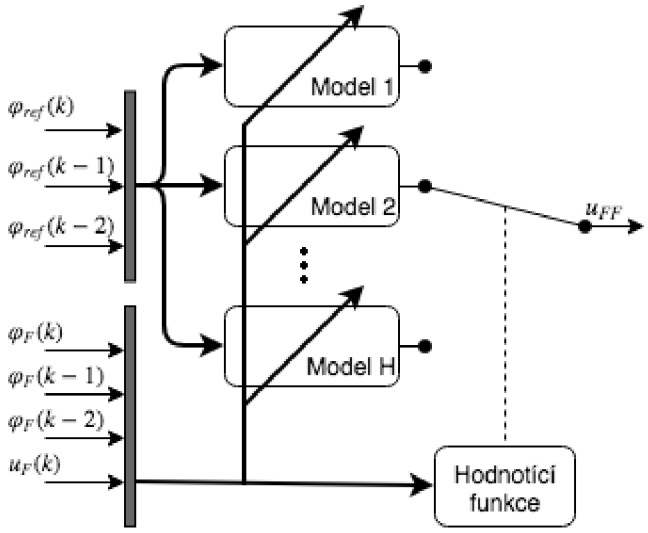
Budeme-li se držet původní publikace, použijeme lokální modely, které budou vyjádřeny v diskrétní podobě (rovnice 4.1). Kompletní blokové schéma řízení bude vypadat dle obrázku 4.3 a detailnější blokové schéma vlastního algoritmu je na obrázku 4.4. Na tomto obrázku je vstup vzorku napětí pouze jeden $u_{FF}(k)$, obecně ale mohou být pro výpočet použity i předcházející vzorky. Záleží na modelu soustavy a použité metodě diskretizace.

V této práci se ale budeme hlavně věnovat spojitému vyjádření lokálních modelů, protože přináší některé výhody. Především je to větší názornost a lepší interpretovatelnost jednotlivých konstant a funkcí[2], což nám pomůže především při ladění algoritmu. Hlubší pojednání a porovnání diskrétních a spojitých modelů pro adaptivní dopřednou kompenzaci jsme představili v publikaci [25].

[2]Obecně se domníváme, že člověk je více zvyklý a lépe si dokáže interpretovat modely vyjádřené ve spojité podobě.



Obrázek 4.3: Blokové schéma více-modelového adaptivního kompenzátoru s diskrétním vyjádřením lokálních modelů



Obrázek 4.4: Vnitřní struktura adaptivního dopředného kompenzátoru založeného na více-modelovém přístupu s diskrétním vyjádřením lokálních modelů

Spojité vyjádření spojitých modelů

Lokální model ve spojité podobě představuje například rovnice 4.2.

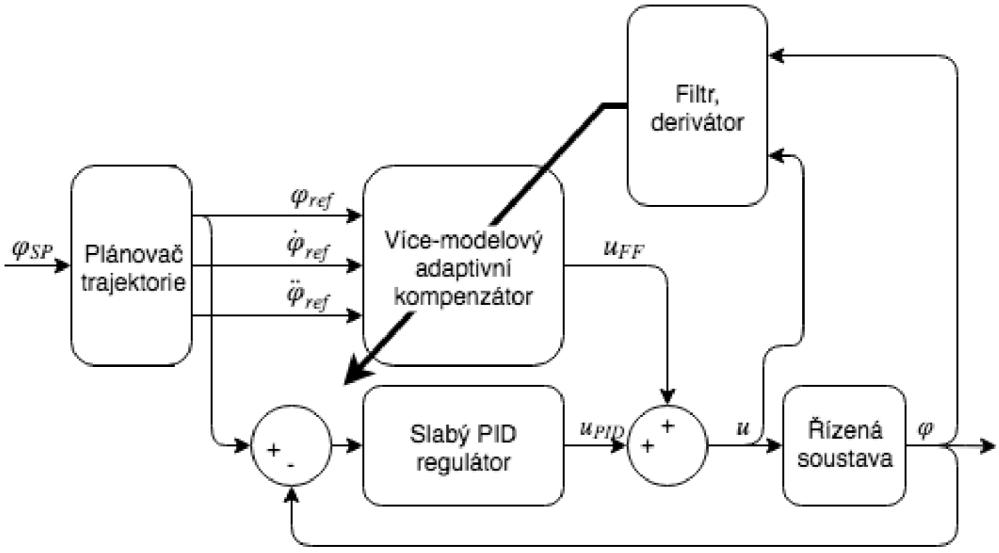
$$u = b_1\ddot{\varphi} + b_2\dot{\varphi} + b_3\varphi + b_4 \tag{4.2}$$

Jedná se o diferenciální rovnici druhého řádu, kde člen $b_1\ddot{\varphi}$ zahrnuje vliv setrvačnosti, $b_2\dot{\varphi}$ představuje viskózní tření (tlumení), $b_2\dot{\varphi}$ sílu závislou na

poloze (v našem případě statickou charakteristiku pružiny) a b_4 je konstantní člen, který v našem případě může zahrnovat předpětí pružiny, případně vliv suchého tření. V tomto modelu je zahrnut vliv suchého tření pouze pokud se jedná o lokální model, kde by pro každý směr pohybu byla jiná konstanta b_4. Za poznámku stojí, že diferenciální rovnice 4.2 nepředstavuje žádný výpočetní problém, jedná se o jednoduchý vztah, protože vstupem jsou poloha a její derivace a výstupem je napětí.

Pro případ spojitého vyjádření lokálních modelů je kompletní schéma řízení na obrázku 4.5. Od diskrétního případu se toto schéma liší především v tom, že vstupem do více-modelového adaptivního kompenzátoru musí být kromě polohy také její derivace až do řádu, který se rovná řádu modelu. Tyto derivace počítá blok *plánovač trajektorie*, jehož vstupem je žádaný signál. Pro učení kompenzátoru je nezbytné znát vstupní napětí soustavy, výstupní signál polohy a její derivace.

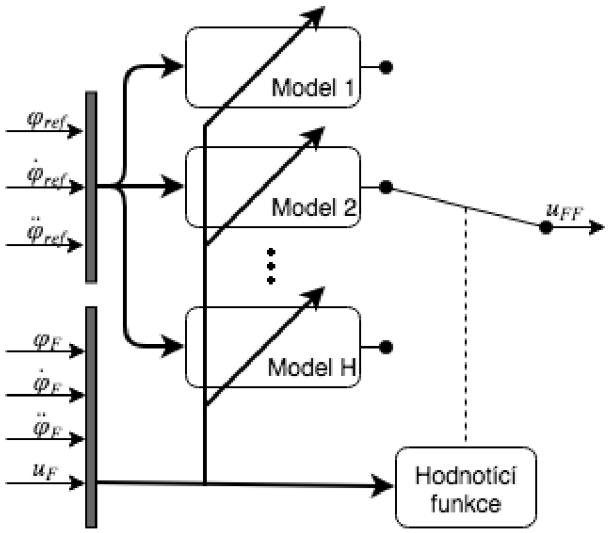
Zde je získání derivací signálu řádově složitější než v plánovači trajektorie, protože signál z polohového sensoru obsahuje šum. Problematice odhadu derivací digitálního signálu obsahujícího šum je věnována celá kapitola 2.2 a 5.



Obrázek 4.5: Blokové schéma více-modelového adaptivního kompenzátoru se spojitým vyjádřením lokálních modelů

Na obrázku 4.6 je blokové schéma kompenzátoru. Vidíme, že vstupem do lokálních modelů pro výpočet kompenzačního napětí je referenční poloha a její derivace $(\ddot{\varphi}_{ref}, \dot{\varphi}_{ref}, \varphi_{ref})$. Pro učení lokální modelů a hodnoticí

funkce slouží skutečné a filtrované napětí a poloha a její odhadované derivace $(u_F, \ddot{\varphi}_F, \dot{\varphi}_F, \varphi_F)$.



Obrázek 4.6: Vnitřní struktura adaptivního dopředného kompenzátoru založeného na více-modelovém přístupu se spojitým vyjádřením lokálních modelů

4.1.1 Volba počtu a vyjádření lokálních modelů

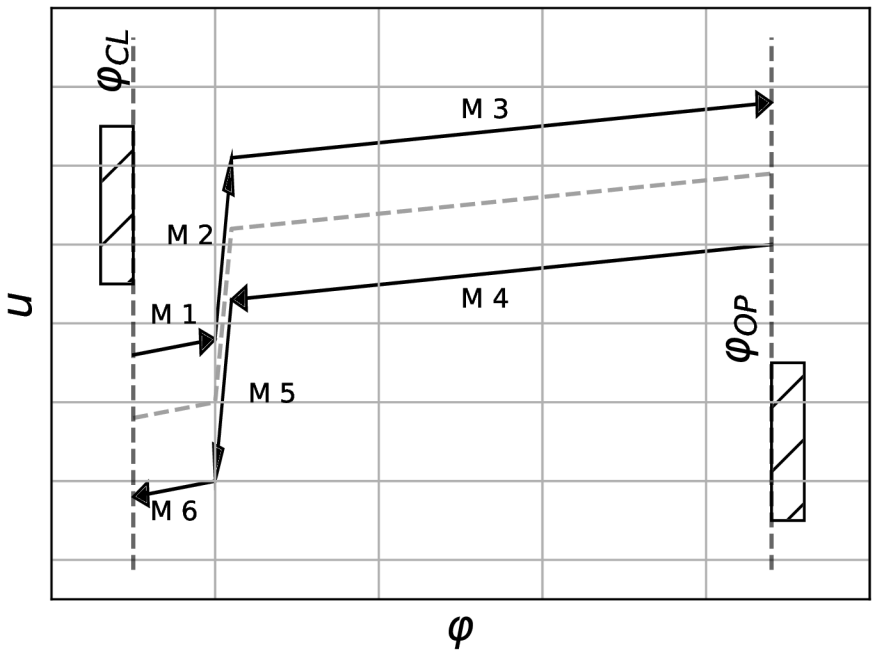
V této kapitole ukážeme pouze několik z velmi mnoha možností, jak vyjádřit jednotlivé modely pro více-modelový adaptivní kompenzátor a s tím související počet těchto modelů. Tyto možnosti budou představeny v pořadí, v jaké si myslíme, že nás napadnou nejdříve, přirozeně až po ty méně zřejmé.

Šest lokálních modelů

Podíváme-li se na kvazi statickou charakteristiku elektronické škrticí klapky (obrázek 2.3), pravděpobně nás jako první možnost napadne rozdělit tuto charakteristku na šest sekcí, kterým bude odpovídat šest lineárních modelů (obrázek 4.7). Každý model můžeme vyjádřit pomocí rovnice:

$$u = b_1\ddot{\varphi} + b_2\dot{\varphi} + b_3\varphi + b_4 \tag{4.3}$$

kde b_4 zahrnuje vliv předpětí pružiny i třecí síly.



Obrázek 4.7: Kvazistatická charakteristika škrticí klapky, zvýrazněno 6 lokálních modelů

Šest lokálních modelů jako sada dvou bank po třech modelech

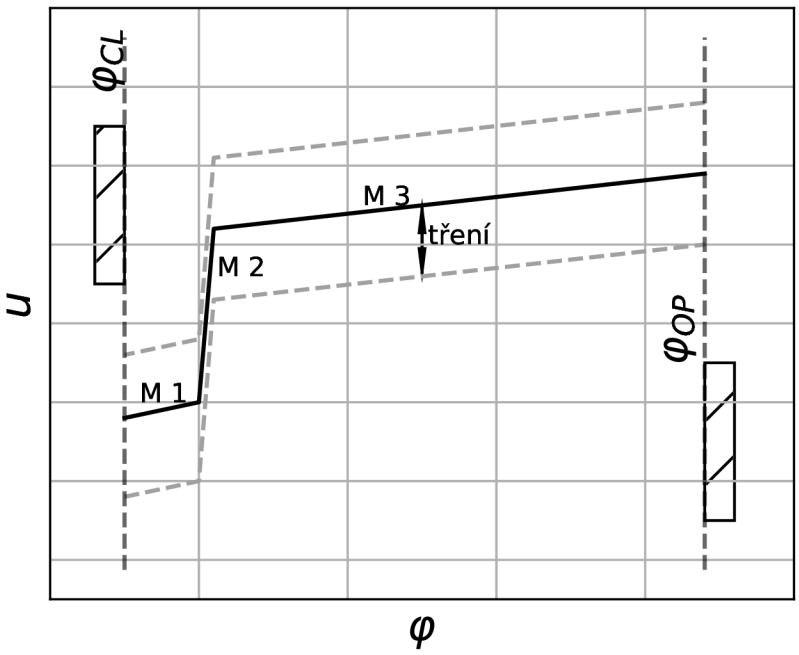
Členění na lokální modely bude stejné jako v předchozím případě, rozdíl však bude v implementaci hodnoticí funkce. Vliv tření, přepínání modelů dle směru rychlosti můžeme předpokládat jako předem dané a v čase neměnné. Budeme tedy manuálně dle směru rychlosti přepínat mezi dvěma bankami tří lokálních modelů a hodnoticí funkce bude provádět mapování pouze dle polohy. Je třeba zvlášť ošetřit situaci, kdy by byla rychlost nulová.

Tři lokální modely

Další možností je v lokálních modelech postihnout vliv tření včetně změny směru rychlosti. Lokální modely můžeme vyjádřit ve tvaru:

$$u = b_1\ddot{\varphi} + b_2\dot{\varphi} + b_3\varphi + b_4 + b_5 sng(\dot{\varphi}) \tag{4.4}$$

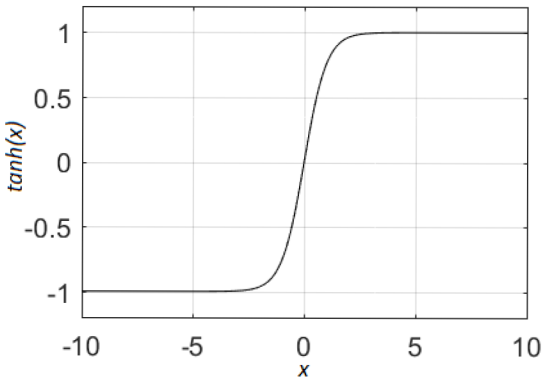
kde pomocí funkce *signum* zachytíme jednoduše vliv suchého tření. Budeme mít pouze tři lokální modely, jak je znázorněno na obrázku 4.8. Nevýhodou



Obrázek 4.8: Kvazistatická charakteristika škrticí klapky, zvýrazněny tři lokální modely

takovéto kompenzace suchého tření pomocí funkce *signum* je to, že v okolí nulové rychlosti dochází ke skokové změně a tím ke skokové změně napětí. Navíc pro nulový vstup je výstup funkce *signum* nula a tuto situaci je nutno zvlášť ošetřit. Toto chování je problém hlavně při pomalém pohybu nebo pokud škrticí klapka drží konstantní polohu. Vlivem okolí a šumu v signálu dochází k neustálé změně směru rychlosti a tím k neustálé skokové změně napětí. Tento problém se stejně projevuje i při implementaci pomocí šesti lokálních modelů, rozdíl je pouze v tom, že dochází k přepínání modelů.

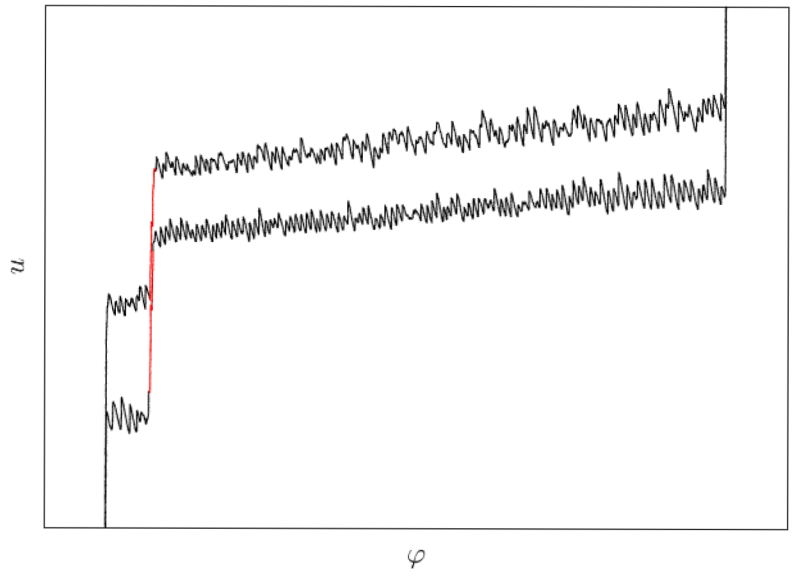
Částečně se dá tento problém odstranit, pokud funkci *signum* nahradíme funkcí hyperbolický tangens, jejíž graf je na obrázku 4.9. Dosáhneme toho, že v okolí nulové rychlosti není skoková změna a průběh napětí bude plynulejší. Sklon v okolí nulové rychlosti můžeme měnit tím, že vstup do funkce hyperbolický tangens budeme násobit vhodnou konstantou.



Obrázek 4.9: Průbeh funkce hyperbolický tangens

Pět lokálních modelů

Tato možnost nemá na první pohled ve statické charakteristice škrticí klapky opodstatnění, ovšem při experimentech se ukázalo, že si můžeme dovolit toto zjednodušení. V okolí *limp-home* pozice je tuhost pružiny tak velká, že charakteristika pro kladný i záporný směr téměř splývá a můžeme ji nahradit jedním modelem (obrázek 4.10). Na toto zjednodušení jsme přišli v okamžiku,



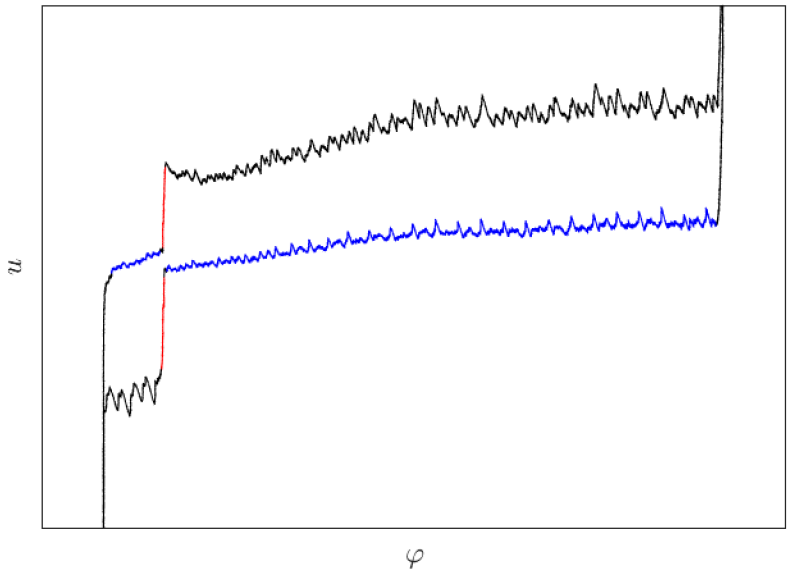
Obrázek 4.10: Skutečná kvazistatická charakteristika škrticí klapky

kdy jsme vyhodnocovali výsledky simulací a z naučeného mapování hodnoticí

funkce bylo zřejmé, že v oblasti *limp-home* pozice použila hodnoticí funkce jeden model pro oba směry rychlosti.

Čtyři lokální modely

Na toto zjednodušení nás opět navedly výsledky simulací. Podíváme-li se na skutečnou kvazi statickou charakteristiku EGR ventilu, můžeme vidět, že oblast otevírání klapky v oblasti zavřeno téměř navazuje na oblast při přivírání klapky v oblasti otevřeno (obrázek 4.11).



Obrázek 4.11: Skutečná kvazistatická charakteristika EGR ventilu

Je třeba ale konstatovat, že toto zjednodušení může mít negativní vliv na chování adaptivního kompenzátoru, pokud by se v průběhu životnosti aktuátoru jeho charakteristika příliš odchýlila od uvažované.

4.1.2 Dimenze mapování hodnoticí funkce

V původní implementaci (kapitola 2.4) byla dimenze mapování hodnoticí funkce shodná s rozměrem stavového prostoru. V našem případě by hodnoticí funkce mapovala z prostoru dimenze tři:

$$\hat{\lambda}_i[\ddot{\varphi}, \dot{\varphi}, \varphi] \mapsto (0, 1) \tag{4.5}$$

a můžeme říci, že výpočetní náročnost je obecně $O(n^3)$, pokud bychom chtěli všechny dimenze pokrýt Gaussovými jádry rovnoměrně. Pro náš konkrétní případ elektromechanických aktuátorů si ale můžeme situaci značně zjednodušit.

Například na základě znalosti aktuátorů můžeme říci, že podle $\ddot{\varphi}$ nedochází k žádnému přepínání mezi lokálními modely, konstanta u $\ddot{\varphi}$ by měla být shodná pro všechny lokální modely. Tím můžeme tuto dimenzi odstranit z mapování hodnoticí funkce a mapování bude probíhat pouze z prostoru $[\dot{\varphi}, \varphi]$ a výpočetní náročnost bude obecně $O(n^2)$.

V našem případě si můžeme dovolit ještě další zjednodušení. Víme, že přepínání modelu dle členu $\dot{\varphi}$ slouží pouze a jedině k rozeznání směru rychlosti, což se využívá ke kompenzaci tření. Po hodnoticí funkci tedy jednoduše řečeno chceme, aby se v dimenzi $\dot{\varphi}$ „naučila“ funkci *signum*. Nepředpokládáme, že by zde mohlo dojít v průběhu života aktuátoru k nějaké změně a můžeme toto mapování považovat za známé a fixní a vyjmout ho z adaptivní hodnoticí funkce. Budeme tedy manuálně přepínat mezi dvěma sadami tří modelů na základě směru $\dot{\varphi}$.

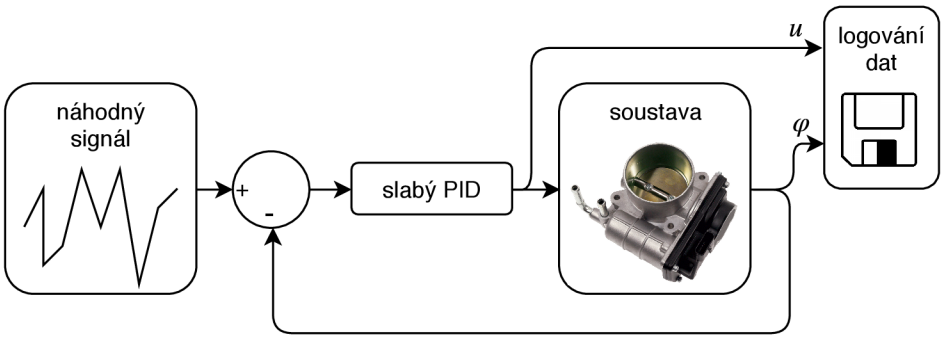
Tím nám zbývá mapování pouze z dimenze φ a výpočetní náročnost bude jenom $O(n^1)$. Značně se nám tedy sníží výpočetní náročnost a zjednoduší se nám práce s hodnoticí funkcí během vývoje a ladění, protože budeme pracovat pouze v jedné dimenzi. Také můžeme předpokládát, že proces učení hodnoticí funkce bude rychlejší a stabilnější.

4.2 Simulační ověření

Simulační ověření navrhovaných algoritmů probíhalo na skutečnýchých datech naměřených na obou aktuátorech. Snažili jsme se získat reálná data pro pohyb aktuátoru v celé pracovní oblasti pro různé rychlosti a sekvence pohybu. Budili jsme aktuátory pomalým trojúhelníkovitým signálem, sinusovým signálem o různé frekvenci, signálem, který se skládá z více sinusových průběhů, náhodným schodovitým signálem, atd.

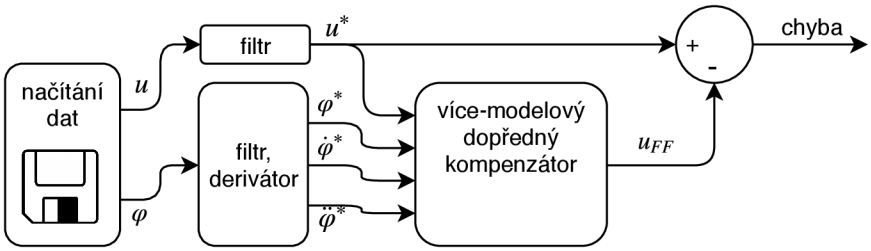
Vzhledem k velmi nelineárnímu chování obou aktuátorů neprobíhalo měření dat v otevřené smyčce, protože se aktuátory díky své charakteristice pohybovaly pouze v malé části svého pracovního prostoru, případně narážely do dorazů. Je snaha mít signál pro testování co „nejbohatší“ a tak jsme měřili data v uzavřené regulační smyčce se „slabým“ PID regulátorem (obrázek 4.12). Náhodný signál tedy sloužil jako vstup do této regulační smyčky a skutečná poloha aktuátoru zhruba sledovala tento signál. Tím se nám podařilo vhodněji získat testovací data.

Vlastní testování probíhalo dle schématu na obrázku 4.13. Naměřená



Obrázek 4.12: Logování testovacích dat s využitím uzavřené regulační smyčky

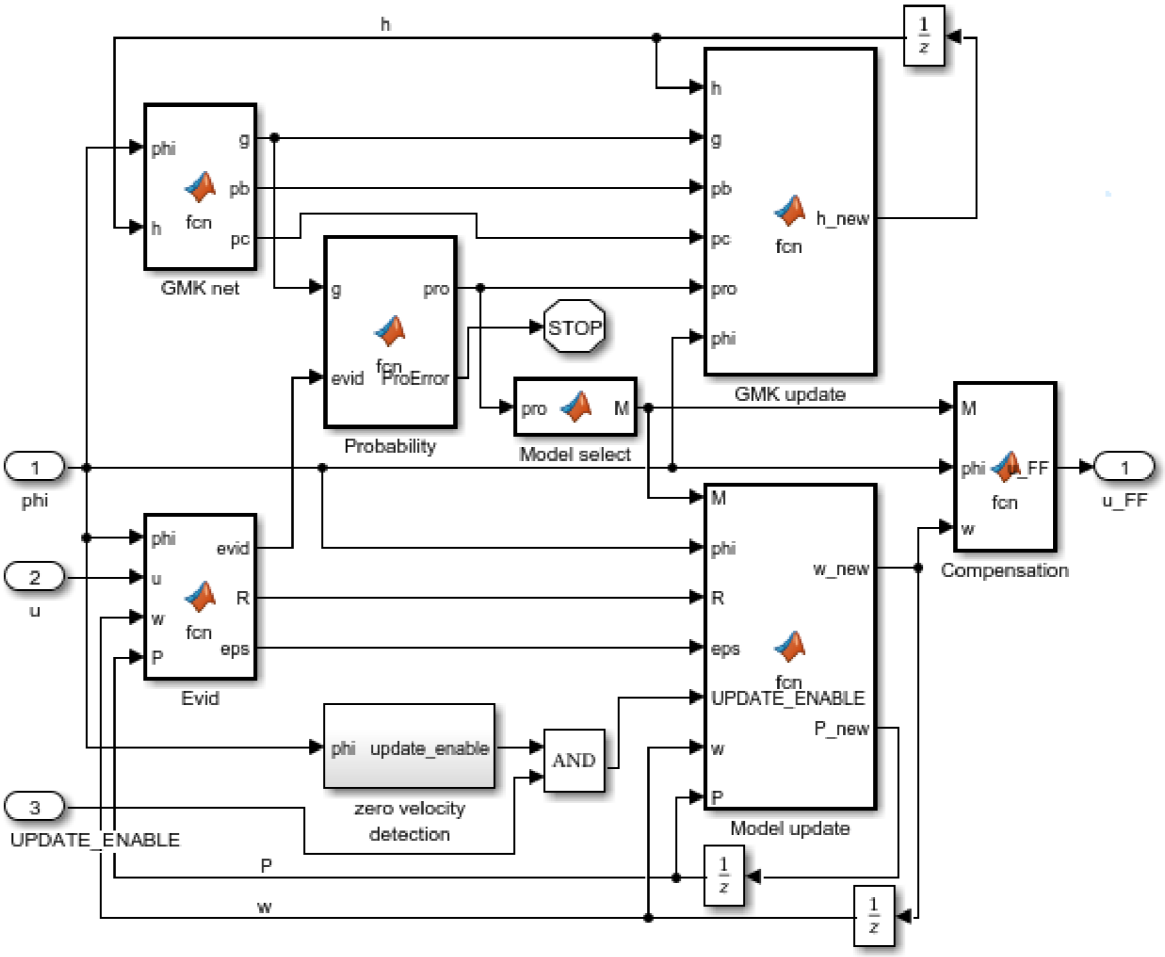
data jsme filtrovali a získávali jsme z nich dále dvě derivace polohového signálu. Tato data poté slouží pro učení adaptivního kompenzátoru a také pro výpočet kompenzačního napětí u_{FF}, které jsme porovnávali s původním napětím po filtraci. V ideálním případě by rozdíl mezi těmito signály měl být nulový, v praxi tomu tak ale být nemůže. Už jen z toho důvodu, že použité lokální modely jsou zjednodušením skutečné soustavy, a i kdyby se korektně naučily zamýšlenou dynamiku, nemůžou plně postihnou chování skutečného aktuátoru.



Obrázek 4.13: Simulační ověřování navrhovaných algoritmů

4.2.1 Implementace algoritmu

Během vývoje byl program implementován a programován ve vývojovém prostředí a skriptovacím programovacím jazyce MATLAB[26]. V tomto prostředí taktéž probíhaly veškeré pomocné práce jako například návrh digitálních filtrů, estimace parametrů skutečné soustavy, tvorba modelů aktuátorů, atd. Velkou výhodou tohoto prostředí je také to, že obsahuje grafické modelovací prostředí Simulink[27] a že spolupracuje s velkým množstvím HW pro RCP a HIL simulace, čehož využijeme v závěru této práce.



Obrázek 4.14: Simulink model více-modelového adaptivního kompenzátoru

Vlastní algoritmus byl implementován v MATLABu a skripty jsou dostupné v přílohách této práce. Funkci algoritmu si ale přiblížíme na modelu v Simulinku (obrázek 4.14), protože dostaneme názornější představu. Algoritmus v MATLABu jsme museli vytvořit pomocí Simulinku kvůli implementaci na RCP a HIL HW a experimentálnímu ověření. Model obsahuje několik oddělených bloků *MATLAB function*, jejichž funkci si popíšeme:

- **GMK net**
  Výpočet hodnoticí funkce pomocí *Gaussian mixture gating network*. Obsahuje rovnice 10.20, 10.21 a 10.22[1]. Vektor hodnoticí funkce je označen jako g. Signály pb a pc obsahují mezivýpočty.

- **GMK update**
  Učení hodnoticí funkce (update vektoru parametrů h). Obsahuje rovnici 10.30[1] a update parametrů je povolen pouze pokud se nacházíme v prostoru mapovací funkce.

- **Probability**
  Výpočet pravděpodobnosti $P(\hat{M}_i(k)|\hat{S}(k-1), \hat{I}^k)$[1] dle rovnice 10.15[1].

- **Evid**
  Výpočet podmíněné pravděpodobnosti $p(y(k)|\hat{M}_i(k), \hat{S}(k-1), \hat{I}^{k-1})$[1]. Obsahuje rovnice 10.12 a 10.13[1].

- **Model select**
  Volba aktuálně platného modelu. Vybere se ten model, který má nejvyšší hodnotu pravděpodobnosti $P(\hat{M}_i(k)|\hat{S}(k-1), \hat{I}^k)$[1].

- **Model update**
  Aktualizace parametrů aktuálně platného modelu pomocí Kálmánova filtru. Obsahuje rovnici 10.19[1]. Učení je povoleno, pouze pokud je na vstupu *UPDATE_ENABLE* logická jedna. Výstupem je aktualizace parametrů modelu a kovarianční matice.

- **Compensation**
  Výpočet kompenzačního napětí na základě aktuálně platného modelu, parametrů modelu a vektoru stavů. V této verzi je implementován výpočet kompenzačního napětí jako výstup nejpravděpodobnějšího modelu.

- **zero velocity detection**
  Tento blok porovnává velikost rychlostí a pokud je rychlost pod nastavenou hranicí, blízká nule, deaktivuje se učení lokálních modelů.

4.2.2 Návrh digitálního filtru, získávání derivací

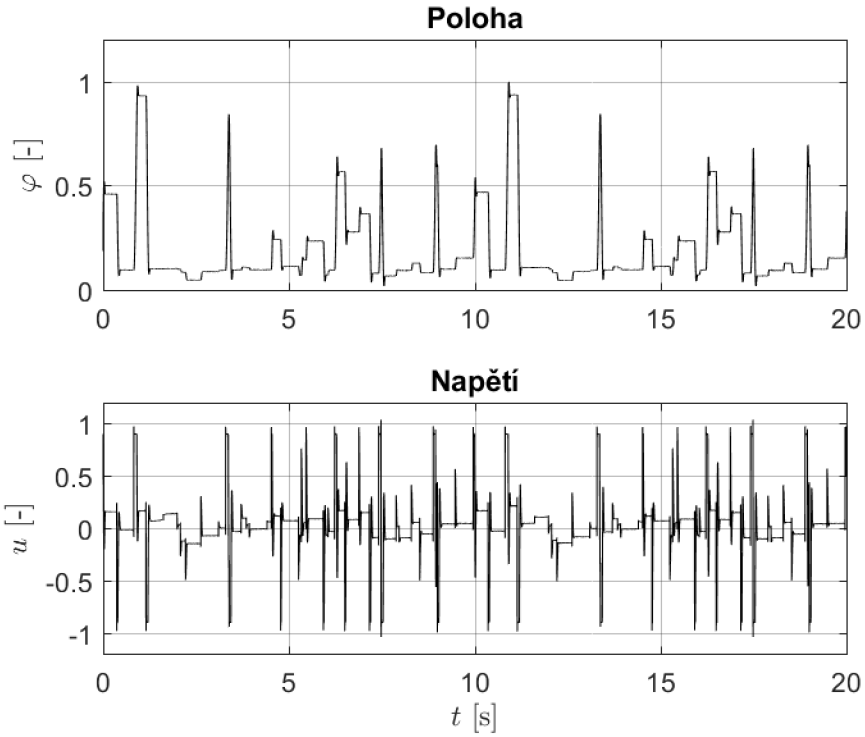
Tato část není přímo součástí adaptivního kompenzátoru, ale výraznou měrou se podílí na kvalitě kompenzace. Pokud je tato část nevhodně navržena, může způsobit nefunkčnost celého algortimu.

Na začátku této kapitoly jsme si řekli, že ze signálu polohy potřebujeme získat první a druhou derivaci a použijeme na to digitální low-pass derivátor. Výchozím bodem pro návrh low-pass filtrů je cut-off frekvence, po kterou by měl být signál teoreticky nedotčen a nad touto frekvencí by měl být původní

[1] Odkazuje na rovnice v původní publikaci [15]

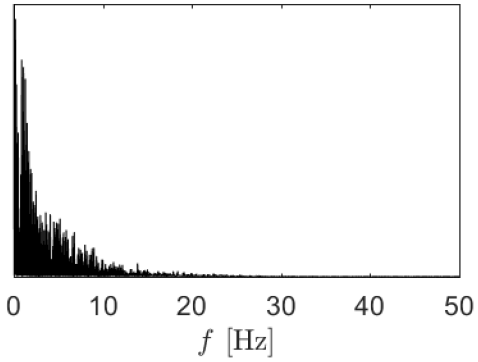
signál utlumen. Klíčová je tedy otázka, jakou hodnotu cut-off frekvence zvolit. Je potřeba najít, jaká nejvyšší užitečná frekvence se může v polohovém signálu objevit.

Teoreticky potřebujeme dosáhnout nejrychlejší možné změny v signálu polohy a tuto frekvenci zachytit. Zvolili jsme přístup, kdy aktuátor budíme náhodným schodovitým signálem. Průběh polohy a napětí je na obrázku 4.15. Tyto signály jsme si zaznamenali a provedli jsme rychlou Fourierovu



Obrázek 4.15: Náhodný schodovitý signál pro odhad maximální rychlosti změny v signálu

transformaci, jejíž výsledek (frekvenční spektrum) je zobrazen na obrázku 4.16. Z frekvenčního spektra můžeme usoudit, že nejvyšší užitečné složky signálu se objevují zhruba do frekvence 25 Hertz. Budeme se tedy snažit navrhnout digitální filtr tak, aby jeho cut-off frekvence byla právě v tomto pásmu.



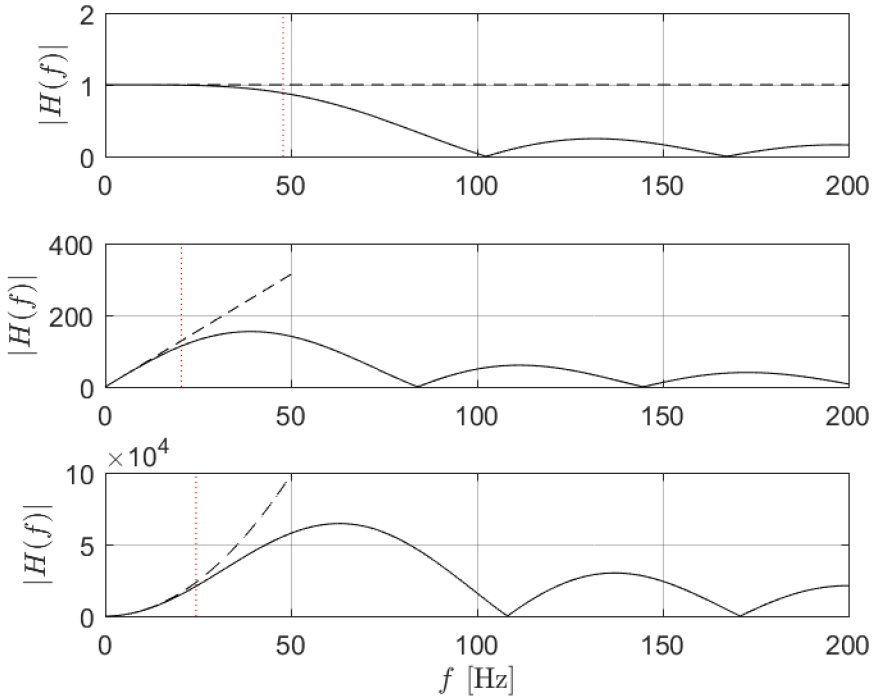
Obrázek 4.16: Frekvenční spektrum signálu polohy

Pro simulační ověření jsme pro jednoduchost volili Savitzky-Golay filtr, protože jedním krokem vrátí koeficienty i pro derivace až do řádu použitého polynomu. Po několika cyklech volby parametrů filtru, zobrazení a vyhodnocení frekvenční odezvy jsem skončili u Savitzky-Golay filtru s těmito parametry:

- délka impulsní odezvy je 17 vzorků
- výpočet derivace se provádí ve středu okna, získáváme symetrickou impulsní odezvu a jedná se o filtr s lineární fází
- data aproximujeme polynomem 2. stupně

Pro tyto parametry jsme získali filtr, jehož frekvenční charakteristiky jsou na obrázku 4.17. Shora dolů se jedná o frekvenční charakteristiku filtru, první a druhé derivace. Černou čárkovanou čarou jsou znázorněny charakteristiky ideálního přenosu pro identitu, první a druhou derivaci. Červenou tečkovanou čarou je znázorněna cut-off frekvence pro jednotlivé filtry s uvažováním poklesu od ideální charakteristiky $1\,dB$.

Vidíme, že pro jednu sadu vstupních parametrů dostaneme rozdílné cut-off frekvence. Teoreticky bychom se mohli snažit dosáhnout stejné cut-off frekvence pro jednotlivé filtry, to by ale vedlo na různě dlouhé impulsní odezvy jednotlivých filtrů a museli bychom se vypořádat s nestejným zpožděním signálu pro polohu a její derivace. To by však nepředstavovalo výrazný problém, pouze bychom zpozdili signál z kratších filtrů tak, abychom dosáhli všude stejného zpoždění. Pro účely simulace, ale i pro účely učení při reálných experimentech není toto zpoždění problémem, protože se nejedná o součást zpětnovazební uzavřené regulační smyčky.



Obrázek 4.17: Amplitudo frekvenční spektrum použitých filtrů, derivátorů (shora dolů jsou to grafy: polohy, rychlost a zrychlení)

4.2.3 Výsledky simulací

Zde uvedeme několik výsledků simulací pro různé varianty nastavení vícemodelového adaptivního kompenzátoru. Jednotlivé varianty se liší především počtem a vyjádřením lokálních modelů. Dále existuje mnoho parametrů (nastavení), které vstupují do výpočtu adaptivního kompenzátoru. Jedná se o tyto parametry:

- adaptivita lokálních modelů:
  - $\mathbf{w}_i^*$ vektor počátečního odhadu parametrů pro každý model
  - σ^2 roztyl měřených hodnot - vyjadřuje míru s jakou nevěříme měření (souvisí i s učením hodnoticí funkce)
  - $\mathbf{Q}$ matice, která se přičítá ke kovarianční matici P během predikce Kálmánova filtru (není součástí původní implementace dle [15])
  - $\mathbf{P}_{init}$ počáteční hodnota kovariační matice Kálmánova filtru, která vyjadřuje míru, jak věříme počátečnímu odhadu parametrů modelu.
  - v_{tr} hranice rychlosti, pod kterou nedochází k učení lokálních modelů (není součástí původní implementace dle [15])
- parametry hodnoticí funkce:
  - pokrytí mapovacího prostoru Gaussovými jádry (počet, střední hodnoty a rozptyl Gaussových jader)
  - η rychlost učení hodnoticí funkce (nemusí být konstantní během regulace)
- volba, zda budeme kompenzační napětí uvažovat jako výsledek nejpravděpodobnějšího modelu nebo budeme počítat kompenzační napětí jako sumu jednotlivých lokálních modelů vážených hodnoticí funkcí
- amplitudo-frekvenční charakteristiky filtrů/derivátorů

Vidíme, že možností, jak ovlivnit algoritmus je poměrně mnoho a některé mají zcela zásadní vliv na kvalitu kompenzace, či přímo způsobí nefunkčnost celého algoritmu. Dá se říci, že tím, že je těchto parametrů takové množství, ztrácí se výhody adaptivního algoritmu. Z toho důvodu se musíme podrobněji seznámit s kompenzovanou soustavou a jí na míru zvolit vhodné parametry kompenzátoru. Zvoleným nastavením způsobíme, že algoritmus bude optimálně fungovat pouze pro omezenou množinu podobných soustav.

Další problém je ten, že ne všechny parametry se dají exaktně určit. Parametry jako počet a vyjádření lokálních modelů, charakteristika filtru, pokrytí mapovacího prostoru Gaussovými jádry a rozptyl měřených hodnot se dají poměrně jasně určit. Ale například parametry jako rychlost učení hodnoticí funkce, matice $\mathbf{Q}$, počáteční odhad kovarianční matice $\mathbf{P}_{init}$ a použití, či velikost hranice rychlosti, pod kterou nedochází k učení se nedají exaktně určit. Toto vede během ladění algoritmů na metodu pokus-omyl s mnoha možnými variantami.

Dalším nedostatkem je skutečnost, že se nám ani po mnoha pokusech s laborováním počátečního nastavení nepodařilo zajistit, aby kompenzátor začal úspěšně fungovat i za situace, kdy počáteční odhad parametrů modelů není vůbec znám a počáteční mapovaní hodnoticí funkce je také neznámé. Tedy za situace, kdy počáteční odhad parametrů libovolně náhodně zvolíme, případně nastavíme všem lokálním modelům stejné parametry, například vektor hodnot jedna.

V původní publikaci [15] je demonstrována funkčnost algoritmu na dvou příkladech, z nichž ten komplikovanější obsahuje tři lokální modely se třemi parametry a segmentace probíhá ve dvou dimenzích. Ještě je třeba zmínit, že každá oblast platnosti lokálního modelu zaujímá poměrně velkou oblast mapovacího prostoru. Neobjevuje se žádná taková malá oblast jako je například *limp-home* poloha u našich aktuátorů. V těchto jednoduchých ukázkách algoritmus poměrně úspěšně funguje a je schopen se naučit parametry lokálních modelů i pokud je počáteční odhad pro všechny lokální modely identický. Taktéž hodnoticí funkce se poměrně úspěšně naučí segmentaci mapovacího prostoru.

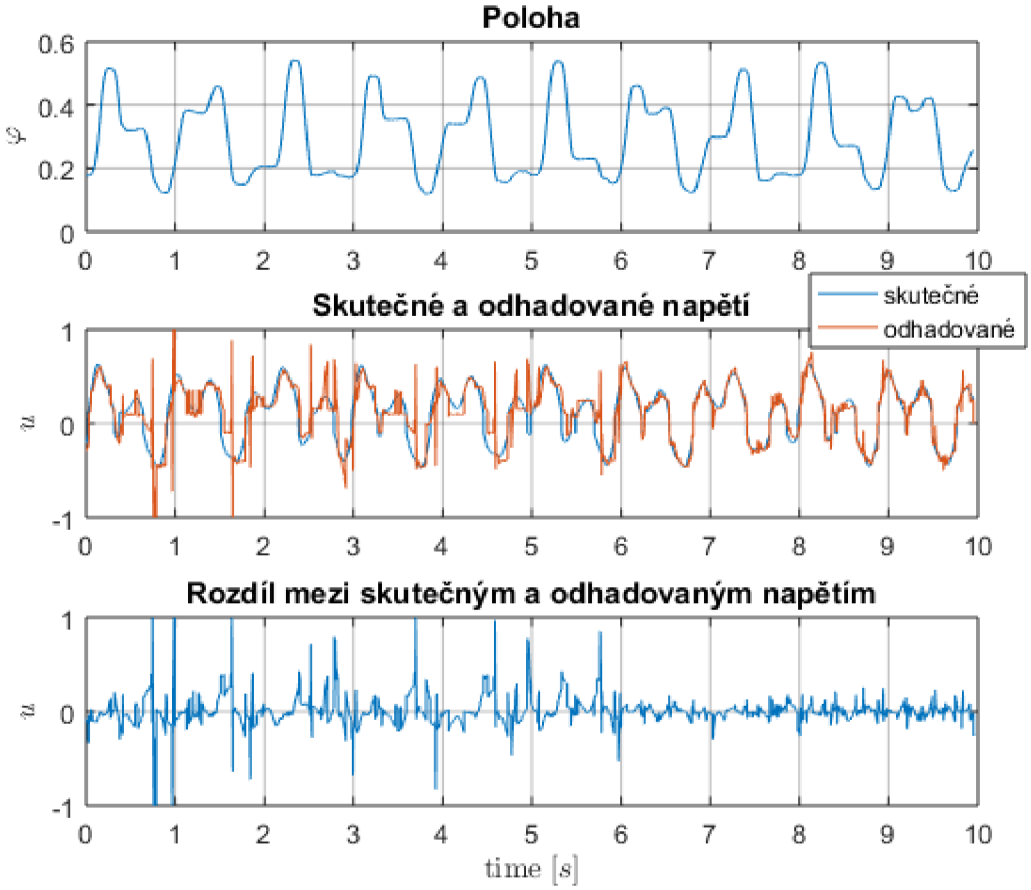
V našem případě jsme tak úspěšní nebyli a počáteční parametry lokálních modelů jsme museli volit poměrně „blízko“ skutečným parametrům modelů. Pojem „blízko“ chápeme tak, že jsme většinu simulací spouštěli s parametry, kdy jsme ke skutečným parametrům modelu přičítali náhodné číslo z normálního rozdělení se střední hodnotou nula a rozptylem zhruba 10% hodnoty parametru. Hodnoticí funkce začínala pracovat bez jakéhokoliv předučení.

EGR ventil - šest lokálních modelů

Jako první uvedeme výsledek simulace pro kompenzaci EGR ventilu, která používá šest lokálních modelů. Některé parametry simulace jsou uvedeny v tabulce 4.1. Kromě těchto parametrů je zvoleno rovnoměrné pokrytí mapovacího prostoru hodnoticí funkce gausovými jádry v intervalu $(0, 1, 0.9)$ pro dimenzi φ a $(-5, 5)$ pro dimenzi $\dot{\varphi}$. Celkový počet Gaussových jader je $200 \times 6 = 1200$. Učení lokálních modelů je povoleno od šesté sekundy simulace. Výpočetní krok simulace je 1 ms.

Tabulka 4.1: Některé parametry simulace pro EGR ventil a šest lokálních modelů

σ^2	0.05
$\mathbf{Q}$	diagonální matice s prvky 1e-6
$\mathbf{P}_{init}$	diagonální matice s prvky 1e-4
η	0.5 v prvních čtyřech sekundách simulace
	0.005 po zbytek simulace
v_{tr}	0



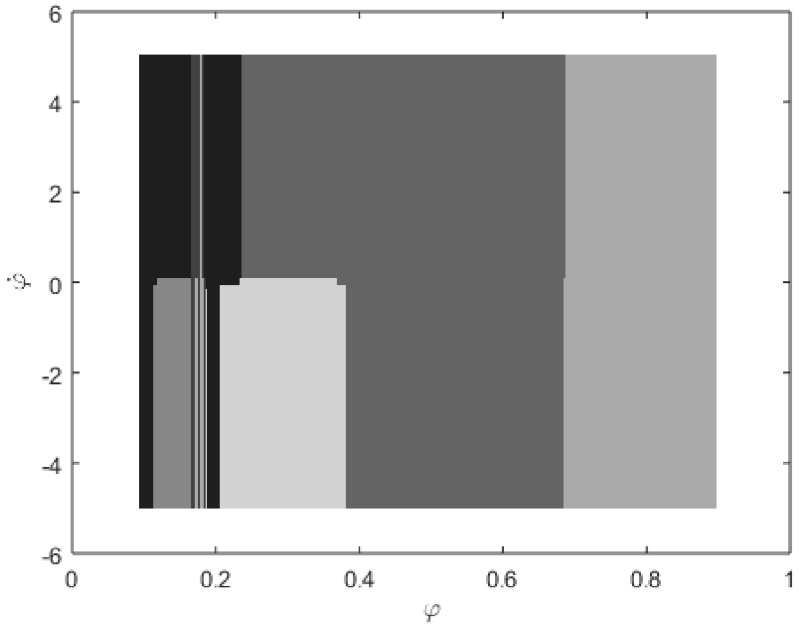
Obrázek 4.18: Průběh polohy, skutečného a simulovaného napětí pro EGR ventil a šest lokální modelů

V grafech na obrázku 4.18 jsou shora dolů zobrazeny grafy průběhu referenčního signálu polohy, průběhů skutečného vstupního napětí EGR ven-

tilu a vypočítaného kompenzačního napětí a graf rozdílu mezi skutečným a kompenzovaným napětím. Jednotky v grafech přímo neodpovídají fyzikálním veličinám, ale jsou normovány do vhodného rozsahu. Například napětí v intervalu $\langle -1, 1 \rangle$ odpovídá dostupnému napájecímu napětí, což bylo v našem případě 12 V. Poloha je normována do rozsahu $\langle 0, 1 \rangle$, atd.

Z průběhů grafů na obrázku 4.18 vidíme, že jsme jako referenční polohu zvolili signál, který se skládá z několika sinusových průběhů. Na počátku simulace není kompenzace příliš kvalitní, což je přirozené, protože hodnoticí funkce se musí naučit segmentaci a parametry lokálních modelů se liší od těch skutečných. Od čtvrté sekundy se zpomaluje rychlost učení hodnoticí funkce a v šesté sekundě se povoluje učení lokálních modelů. Tento okamžik je v datech poměrně zřetelný, v čase 6 s se rapidně sníží chyba kompenzace.

Na obrázku 4.19 vidíme vizualizaci naučeného mapování hodnoticí funkce. Rozdílným odstínem šedi jsou zobrazeny rozsahy platnosti jednotlivých lokálních modelů. Vidíme, že rozdělení dle směru rychlosti neexistuje v celém rozsahu polohy (φ). Krajní polohy rozsahu nejsou rozděleny dle směru rychlosti a je zde platný pouze jeden model pro oba směry rychlosti. To bude způsobeno z velké části tím, že referenční signál polohy se pro tuto simulaci nepohyboval v celém pracovním rozsahu aktuátoru a tím ani nemohl algoritmus získat žádná data pro danou oblast. Také si můžeme všimnout, že v

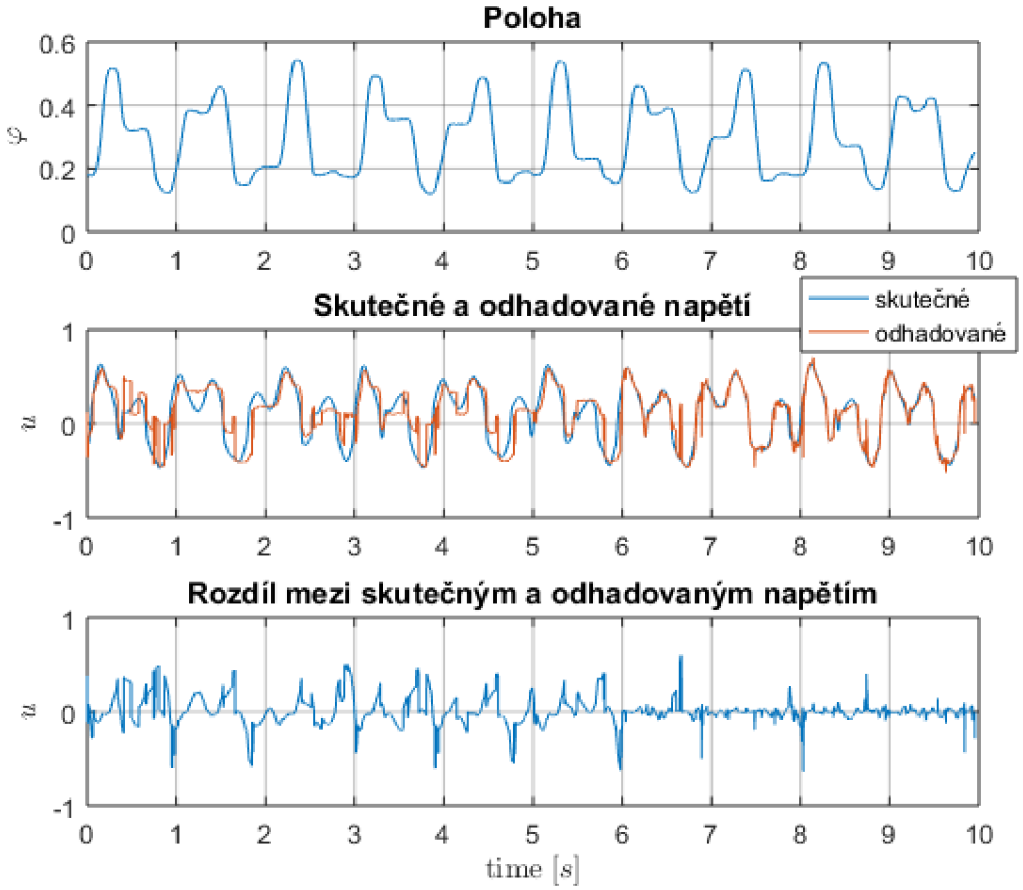


Obrázek 4.19: Segmentace mapovacího prostoru hodnoticí funkce - EGR ventil, šest lokální modelů

okolí *limp-home* pozice ($\varphi \approx 0.19$) je více modelů platných pro celý rozsah rychlosti.

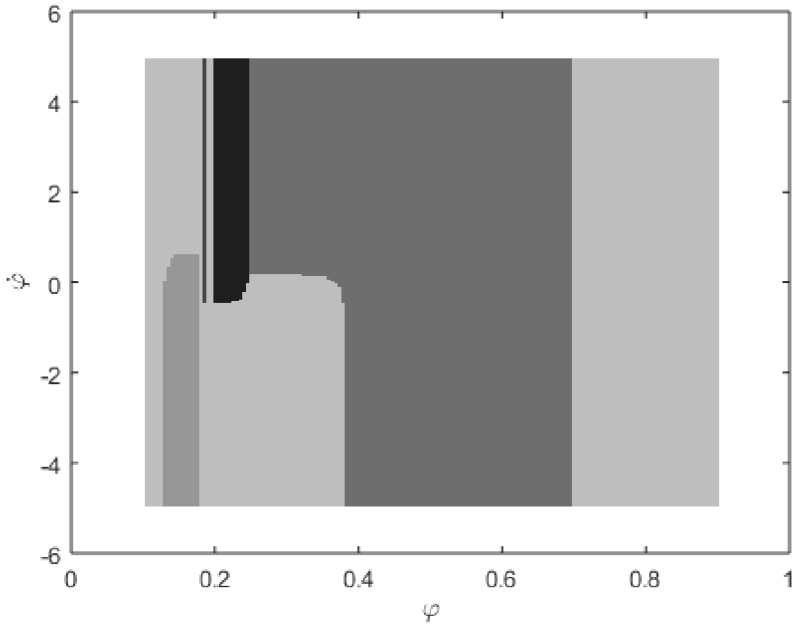
EGR ventil - pět lokálních modelů

Pro tuto simulaci kompenzace EGR ventilu použijme pouze pět lokálních modelů, tak jak bylo představeno v kapitole 4.1.1. Parametry simulace byly voleny stejně jako v předchozí sekci. Výsledné průběhy jsou na obrázku 4.20 a mapování hodnoticí funkce po deseti sekundách simulace je na obrázku 4.21.



Obrázek 4.20: Průběh polohy, skutečného a simulovaného napětí pro EGR ventil a pět lokální modelů

Referenční signál je opět stejný jako v předchozím případě a v čase 6 s vidíme výrazné zlepšení kompenzace. Porovnáme-li kvalitu kompenzace vůči předchozímu případu, zjistíme dokonce nepatrné zlepšení, které nám



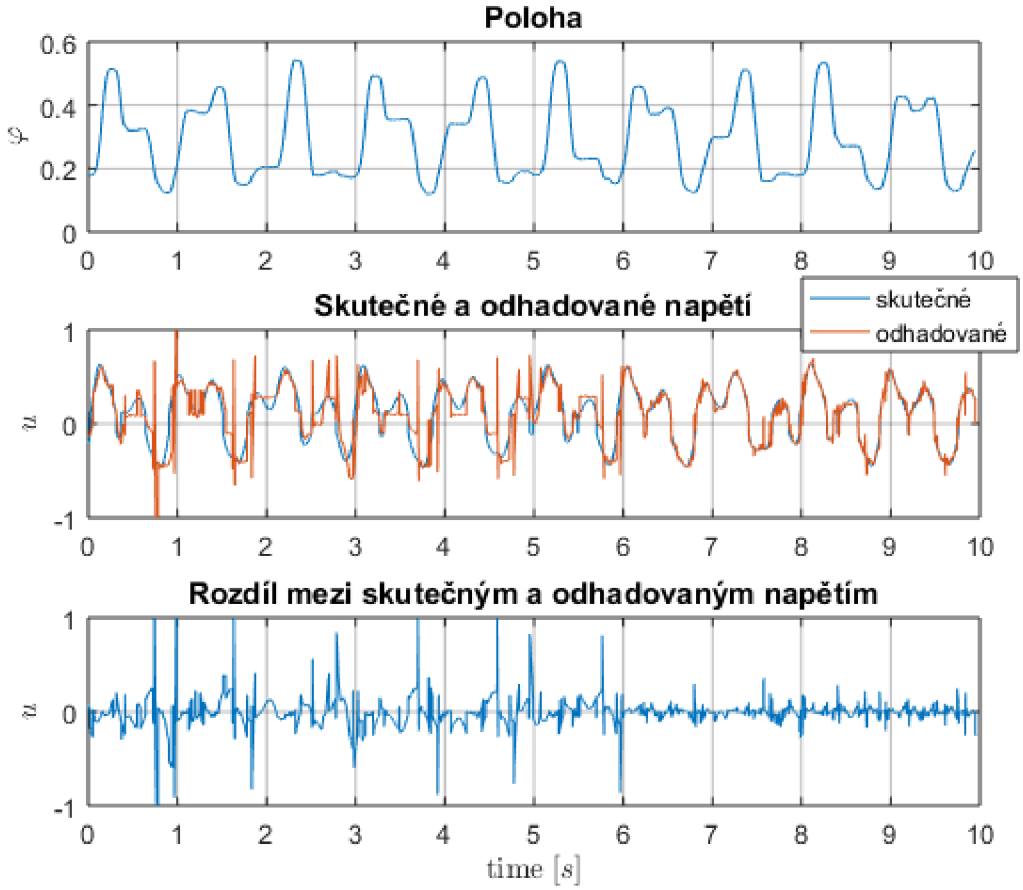
Obrázek 4.21: Segmentace mapovacího prostoru hodnoticí funkce - EGR ventil, pět lokální modelů

potvrdila i nižší hodnota MSE (střední kvadratické odchylky) pro poslední sekundy simulace.

Za pozornost stojí především výsledné mapování hodnoticí funkce 4.21, kde vidíme, že aktivní jsou pouze čtyři modely. V oblasti nad *limp-home* pozicí je prostor rozdělen na modely dle směru rychlostí, ale v okolí limp-home pozice každý model zasahuje částečně i do druhého směru rychlosti. Kraje rozsahů jsou opět pokryty pouze jedním modelem, který je platný pro oba směry rychlosti. To bude opět způsobeno nedostatečným rozkmitem referenčního signálu.

EGR ventil - čtyři lokální modely

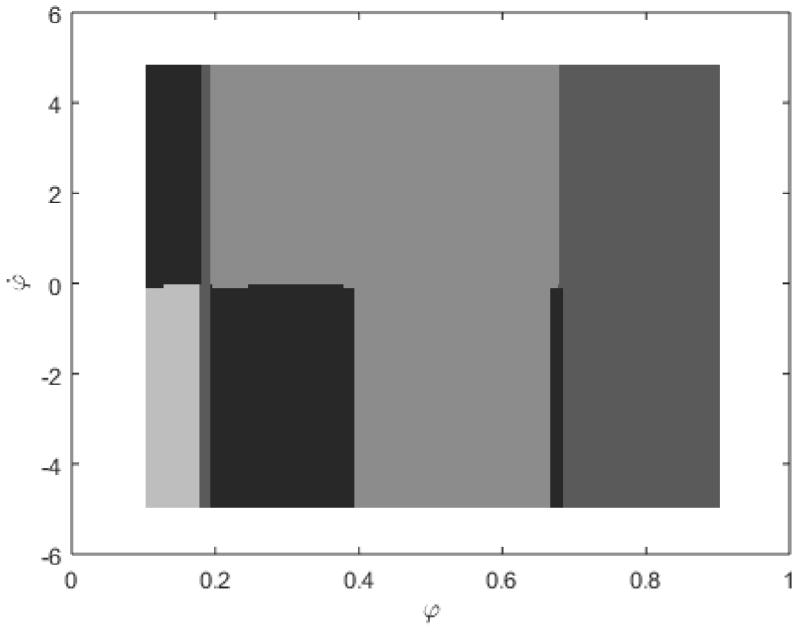
Pro tuto simulaci jsme použili pouze čtyři lokální modely a vycházeli jsme z kapitoly 4.1.1. Parametry simulace a budicí signál jsou stejné jako v předchozích dvou příkladech. Podíváme-li se na průběhy signálů na obrázku 4.22, vidíme, že kvalita kompenzace nedosahuje předchozího případu a hodnota MSE byla podobná jak pro variantu s šesti lokálními modely.



Obrázek 4.22: Průběh polohy, skutečného a simulovaného napětí pro EGR ventil a čtyři lokální modely

Podíváme-li se ale na segmentaci hodnoticí funkce (obrázek 4.23), můžeme konstatovat, že výsledné mapování téměř přesně odpovídá předpokladům v kapitole 4.1.1. Oblast okolo *limp-home* pozice pokrývá jeden lokální model platný pro oba směry rychlostí. Dále existuje jeden lokální model, který pokrývá oblast kladné rychlosti před LH pozicí a záporné rychlosti za LH pozicí. Zbývající dva modely pokrývají oblast záporné rychlosti před LH po-

zicí a kladné rychlosti za LH pozicí. Oblasti přibližně nad polohou 0,4 nebyly dostatečně vybuzeny referenčním signálem a tak jejich mapování nemůže odpovídat předpokladům.



Obrázek 4.23: Segmentace mapovacího prostoru hodnoticí funkce - EGR ventil, čtyři lokální modely

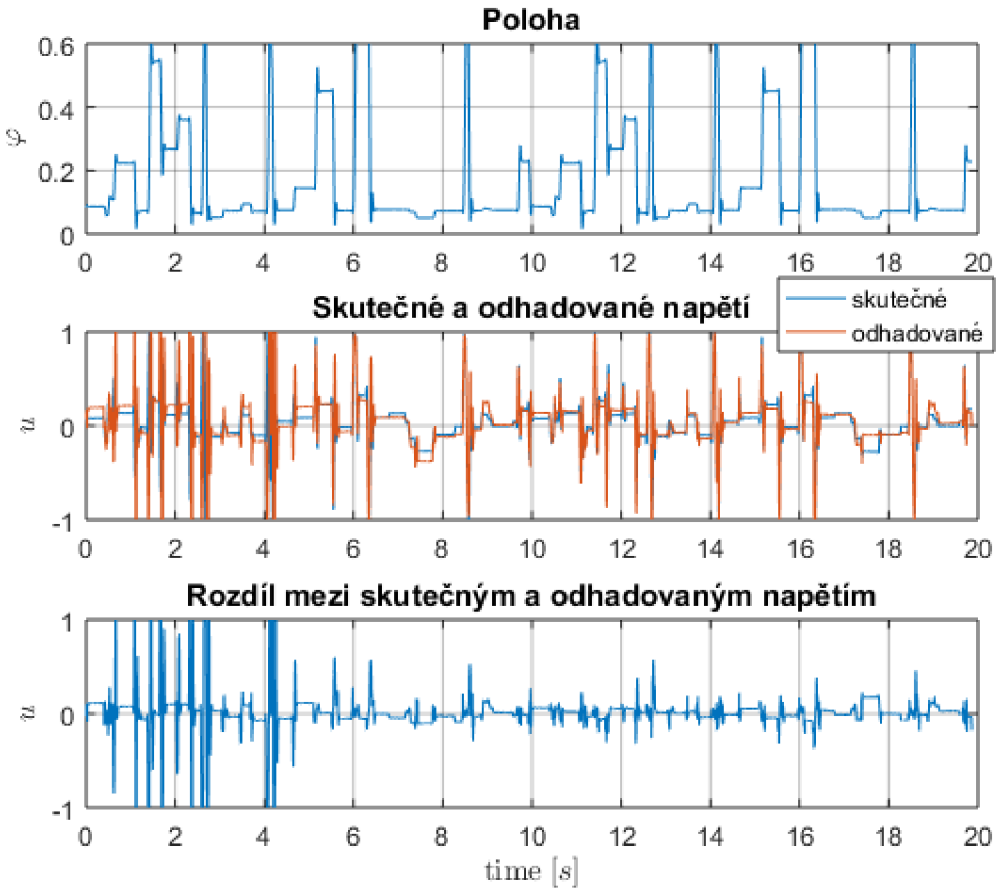
Škrticí klapka - pět lokálních modelů

Nyní se jedná o simulaci kompenzace elektronické škrticí klapky a pro adaptivní kompenzátor použijeme pět lokálních modelů. Kromě typu aktuátoru se tato simulace liší ve výpočetním kroku, který je nyní 5 ms a referenčním signálem. Referenční signál nyní tvoří náhodně schody v signálu, které trvají náhodnou dobu. Referenční signál byl tvořen tak, aby se poloha pohybovala více v okolí LH pozice. Další parametry simulace jsem uvedeny v tabulce 4.2. Mapovací prostor je opět rovnoměrně pokryt Gaussovými jádry, ale jejich počet je vyšší, protože oblast okolo *limp-home* pozice je užší a v dimenzi φ potřebujeme vyšší rozlišení. Celkový počet těchto jader je 2400. Průběhy signálů vidíme na obrázku 4.24. Učení lokálních modelů je povoleno od páté sekundy simulace a od tohoto okamžiku vidíme zlepšení v kompenzovaném signálu. Kvalita kompenzace na první pohled nedosahuje předchozích případů, ale je nutno brát v úvahu, že tentokrát je referenční signál schodo-

Tabulka 4.2: Některé parametry simulace pro elektronickou škrticí klapku a pět lokálních modelů

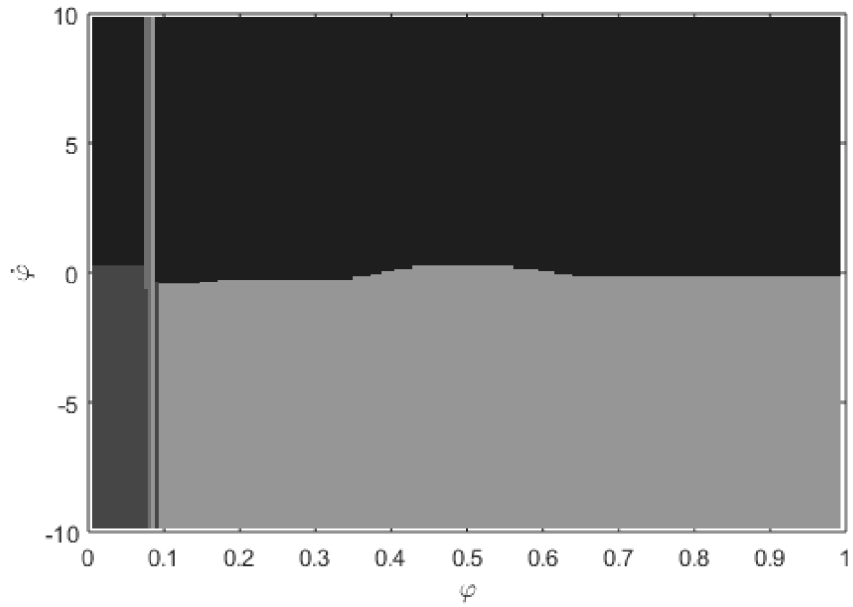
σ^2	0.05
$\mathbf{Q}$	diagonální matice s prvky 1e-7
$\mathbf{P}_{init}$	diagonální matice s prvky 1e-3
η	0.2 po celou dobu simulace
v_{tr}	0.005

vitý a po většinu doby je rychlost aktuátoru nulová a v tuto dobu nemůže probíhat a neprobíhá učení.



Obrázek 4.24: Průběh polohy, skutečného a simulovaného napětí pro elektronickou škrticí klapku a pět lokální modelů

Segmentace mapovacího prostoru hodnoticí funkce je zobrazena na obrázku 4.25. Vidíme, že až na oblast *limp-home* pozice je prostor rozdělen dle směru rychlosti. Ovšem oblast kladné rychlosti před i za LH pozicí pokrývá jeden lokální model, což neodpovídá skutečnosti a pro oba směry rychlosti dochází v oblasti LH pozice k jednomu přechodu modelu navíc.



Obrázek 4.25: Segmentace mapovacího prostoru hodnoticí funkce - škrticí klapka, pět lokální modelů

4.3 Experimentální ověření

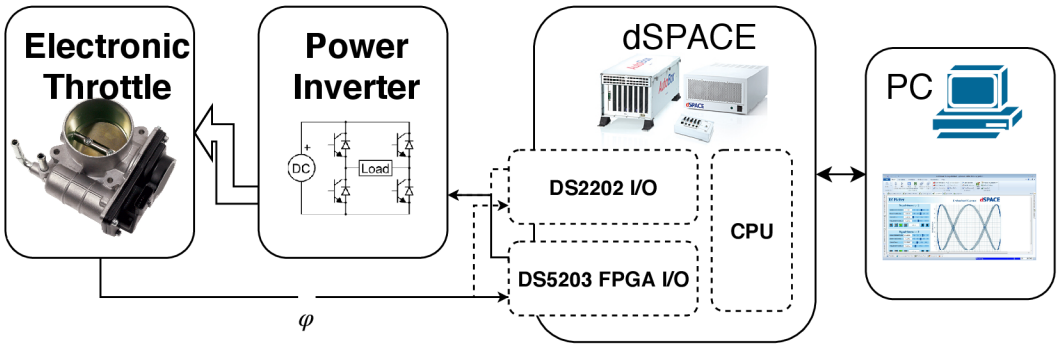
V této kapitole stručně představíme stav pro experimentální ověření navrhovaných algoritmů, popíšeme implementaci algoritmu na real-time HW a experimentální ověření. V závěru této kapitoly uvedeme několik poznámek ze simulačního a experimentálního ověření, jež mohou být inspirací pro případný budoucí vývoj.

4.3.1 Popis testovacího stavu pro vyhodnocení navrhovaných algoritmů

Pro experimentální ověření navrhovaných algoritmů, ale i pro identifikaci parametrů obou elektromechanických aktuátorů vznikl testovací stav, který umožňuje jejich řízení, logování dat a rychlé prototypování navrhovaných algoritmů jak na procesoru, tak i s využitím FPGA. Blokové schéma testovacího stavu je na obrázku 4.26. Jako vstupně/výstupní HW slouží real-time počítač - modulární zařízení od firmy dSPACE dostupné v mechatronické laboratoři [28], které obsahuje tyto komponenty:

- **DS1006** procesorová karta
  - čtyřjádrový procesor AMD Opteron taktovaný na 2,8 GHz
  - 1 GB DDR RAM
- **DS2202** HIL I/O karta
  - 16x diferenciální analogový vstup - rozlišení 14 bit, vstupní rozsah 0..60 V
  - 20x analogový výstup - rozlišení 12 bit, 0..10 V s interní referencí
  - 38x digitální vstup (24x čtení PWM signálu)
  - 16x digitální výstup (9x generování PWM)
- **DS5203** FPGA karta
  - Xilinx Virtex-5 SX95T-2C - 94298 logických bloků, 640 DSP slice, 100 MHz
  - 16x digitální vstup/výstup (3.3 V nebo 5 V logika)
  - 6x analogový vstup - rozlišení 14 bit, vzorkování 10 MHz, vstupní rozsah volitelný ±5 V nebo ±30 V
  - 6x analogový výstup - rozlišení 14 bit, vzorkování 10 MHz, výstupní rozsah ±10 V

- **DS5203M1** rozšiřovací modul pro FPGA kartu
  - zdvojnásobuje počet všech vstupů/výstupů karty DS5203
- **DS814** komunikační karta



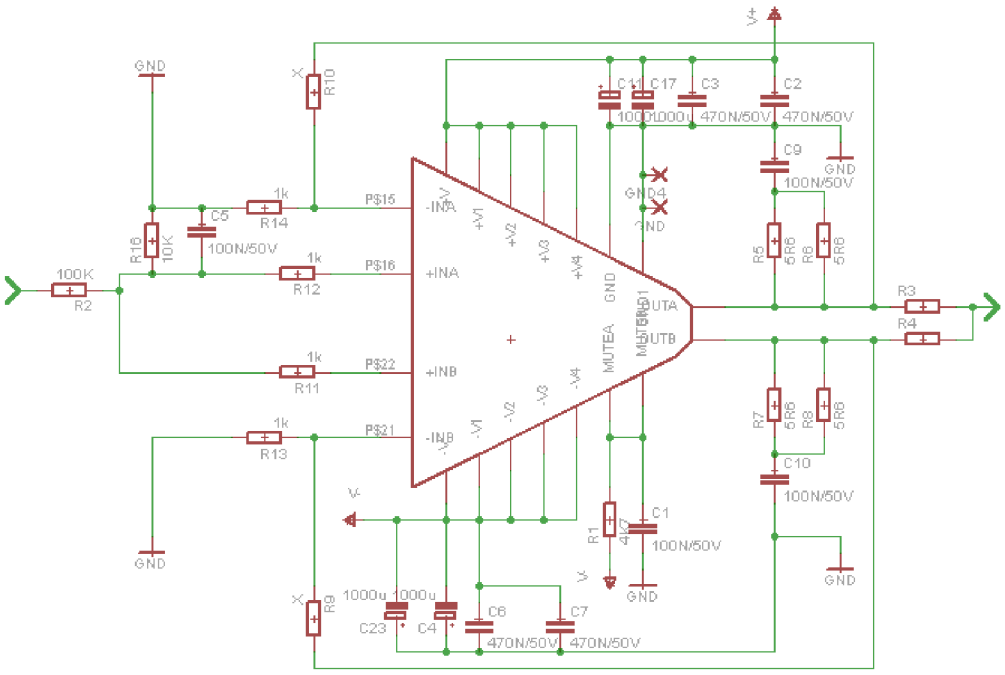
Obrázek 4.26: Blokové schéma experimentálního stavu pro ověřování navržených algoritmů

Pro výkonové posílení akčního signálu máme k dispozici dvě možnosti. Jednou je použití pulsně-šířkové modulace a výkonového H-můstku, který byl dostupný v mechatronické laboratoři. A druhou možností je analogový zesilovač signálu, který jsme navrhli s využitím obvodu LM4780 [29]. LM4780 je dvoukanálový audio zesilovač s nízkým zkreslením, kde každý kanál má výkon 60 W. Využili jsme možnosti spojit tyto kanály do jednoho, a tak máme teoretický výkon 120 W, což pohodlně stačí pro napájení obou aktuátorů. Schéma analogového zesilovače je na obrázku 4.27.

Při implementaci řízení je přirozeně uvažováno s výkonovým H-můstkem, ale při identifikaci parametrů a jemném ladění řízení je vhodné mít možnost plynule řídit vstupní napětí obou aktuátorů bez zvlnění způsobeného pulsně-šířkovou modulací.

Navrhované algoritmy budou implementovány v programu MATLAB/ Simulink s využitím rozšíření dSPACE Real-Time Interface [30]. Výpočetní krok na zařízeních firmy dSPACE můžeme dle složitosti modelu očekávat v řádu 10-100μs. V případě požadavku na vyšší výpočetní krok (např. pro digitální zpracování signálu) můžeme využít FPGA na kartě DS5203, které můžeme programovat s pomocí softwaru Xilinx System Generator for DSP [19] opět v prostředí MATLAB/Simulink. Fotografie testovacího stavu ještě v provedení pro dSPACE AutoBox je na obrázku 4.28.

Jako sensor polohy je u obou aktuátorů použit potenciometr. Pro čtení signálu z potenciometru jsme využili FPGA kartu nejen z důvodu vysoké



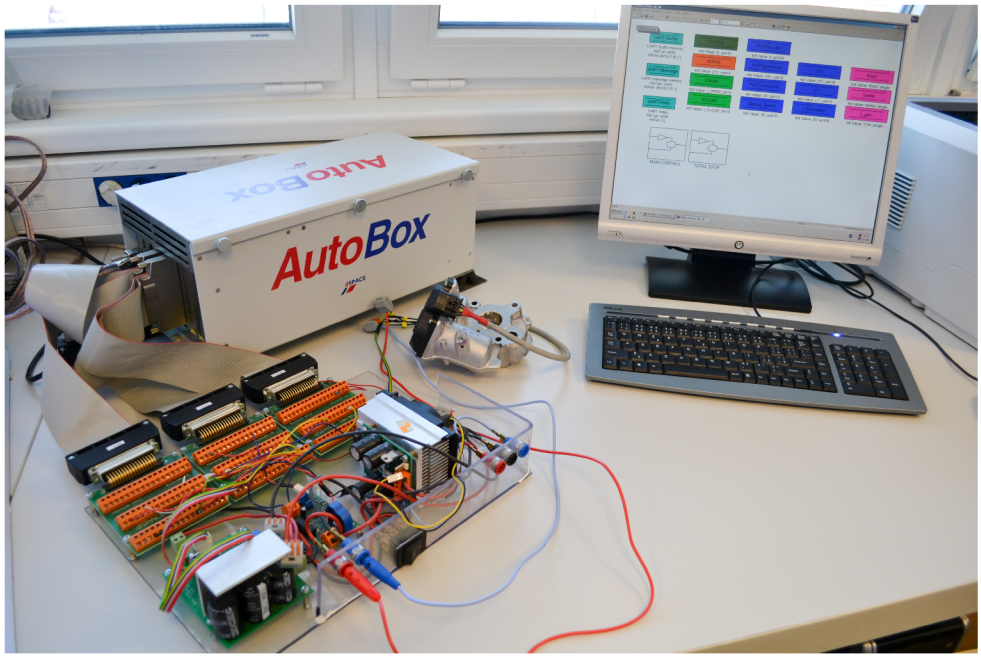
Obrázek 4.27: Schéma analogového zesilovače s využitím integrovaného obvodu LM4780 [29]

vzorkovací frekvence a výpočetního výkonu pro implementaci filtrace, ale také z toho důvodu, že pro nás má vhodnější vstupní rozsah.

Vstupní rozsah analogových vstupů procesorové části dSPACE je 0..60 V, ale potenciometr může být dlouhodobě napájen napětím pouze 5 V. Tím bychom využili pouze malou část vstupního rozsahu A/D převodníku a měli bychom poměrně malé rozlišení signálu ($2^{14}/60 * 5 = 1365,\overline{3}$). Signál z potenciometru bychom mohli zesílit, ale museli bychom vytvořit poměrně komplikovaný analogový zesilovač a ještě bychom do cesty signálu zanášeli další nejistoty. Analogový vstup na FPGA má rozsah ±5 V a ten mnohem lépe využijeme.

4.3.2 Optimalizace algoritmu a příprava pro běh na real-time zařízení

Prvotní příprava algoritmu spočívala v převedení programu napsaného v programovacím jazyce MATLAB do Simulinku. Tento krok byl nutný z toho důvodu, že program pro dSPACE HW je možno generovat s využitím dSPACE Real-Time Interface pouze ze Simulinku. Při převádění programu do Simu-



Obrázek 4.28: Foto experimentálního stavu (jako vstupně/výstupní zařízení slouží dSPACE HW v provedení AutoBox)

linku jsme v maximální možné míře využili bloků *MATLAB function*, jak bylo znázorněno na obrázku 4.14. Aby bylo možno ze Simulink modelu generovat C kód nejen pro dSPACE, ale i obecně, bylo třeba ve všech blocích přesně definovat rozměry jednotlivých signálů. Po tomto kroku je možno ze Simulink modelu generovat C kód.

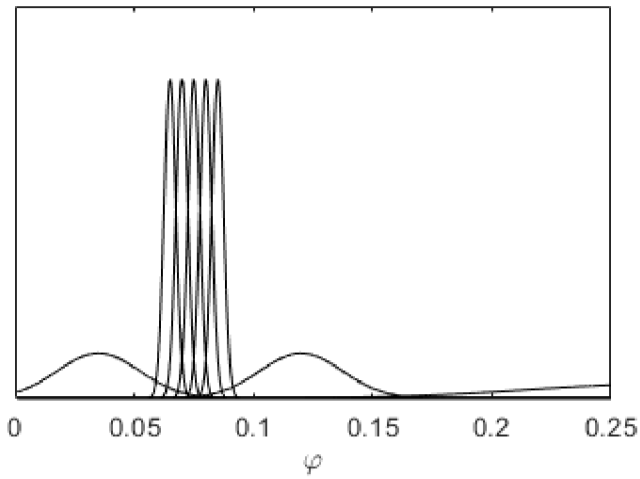
Ještě zbývá přidat bloky představující vstupy a výstupy reálného HW a dát unikátní popisek těm signálům v Simulinku, ke kterým budeme chtít přistupovat během simulace. Poslední krok je vytvoření uživatelského rozhraní v prostředí Control Desk [31].

Při prvním spuštění algoritmu dopředného kompenzátoru na dSPACE HW s výpočetním krokem 1 ms jsme ihned narazili na problém. Výpočetní čas programu trval déle než zvolený krok a program nemohl běžet v *real-time* režimu. Program běžel pouze na jednom jádře ze čtyř dostupných, ale místo paralelizace výpočtů jsme se vydali cestou optimalizace s cílem snížit výpočetní náročnost. Velký výpočetní čas souvisel s výpočtem hodnoticí funkce a jejím učení. Tento výpočetní čas přímo souvisí s počtem Gaussových jader a tak jsme se vydali cestou snížení jejich počtu při zachování funkce algoritmu.

Pokrytí prostoru mapování hodnoticí funkce

Během simulačního ověření byl mapovací prostor hodnoticí funkce EGR ventilu rovnoměrně pokryt 1800 Gaussovými jádry. V dimenzi $\dot{\varphi}$ je 6 jader a v dimenzi φ 300 jader. Vyjdeme-li z předpokladu, že v dimenzi $\dot{\varphi}$ očekáváme právě jednu hranici okolo nulové rychlosti, můžeme počet jader snížit na dvě a tato jádra rovnoměrně umístíme.

V dimenzi φ, rozdělení hranice oblastí dle polohy, je situace složitější. Očekáváme celkem dvě hranice, které jsou ale velmi blízko sebe a u kraje prostoru. Algoritmus hodnoticí funkce tedy uzpůsobíme tomu, aby všechna Gaussova jádra v dané dimenzi nemusela mít stejný rozptyl a oblast okolo *limp-home* pozice pokryjeme hustěji. Na obrázku 4.29 je znázorněno pokrytí dimenze φ v oblasti *limp-home* pozice. Tímto krokem jsem snížili celkový počet Gaussových jader na $10 \times 2 = 20$ a radikálně jsme snížili výpočetní náročnost.



Obrázek 4.29: Pokrytí dimenze φ hodnoticí funkce Gaussovými jádry s proměnným rozptylem

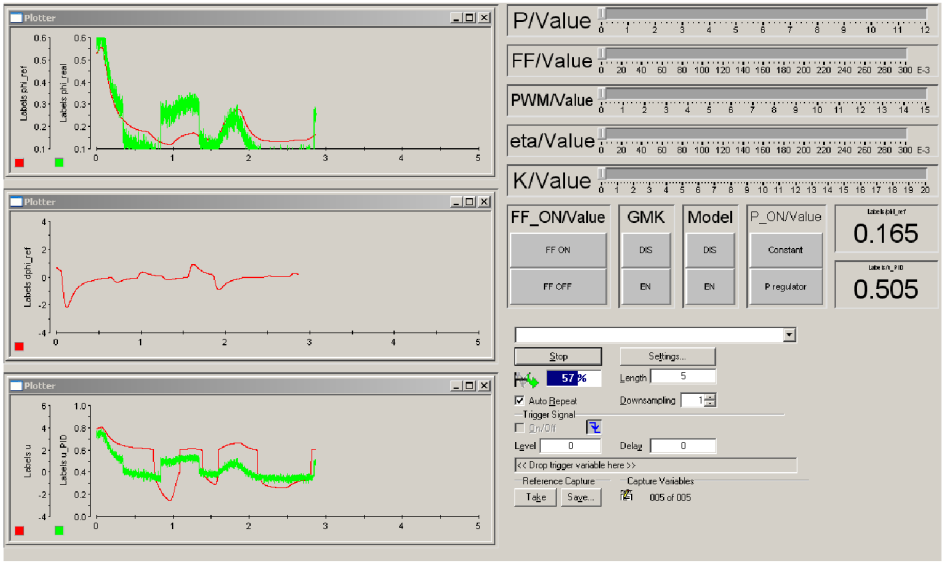
V tomto stavu již bylo možno program bez obtíží spustit na dSPACE HW v *real-time* režimu, ale v optimalizaci jsme se vydali ještě dále. Dopředu jsme spočítali jmenovatele rovnice 2.28 pro každé Gaussovo jádro a tyto informace jsme umístili do paměti. Tyto hodnoty jsou konstantní během simulace, závisí pouze na topologii pokrytí mapovacího prostoru.

Finální pokrytí prostoru mapování hodnoticí funkce Gaussovými jádry není zdaleka ideální z pohledu rovnoměrnosti a přesnosti. Zvláště s ohledem na to, že pokrytí prostoru Gaussovými jádry v našem případě je cíleno

na naměřenou kvazistatickou charakteristiku obou aktuátorů. Pokud by se charakteristika výrazněji změnila, především pokud by se oblast kolem *limp-home* pozice posunula do oblasti, která není hustě pokryta Gaussovými jádry, celý algoritmus by se stal nefunkční.

4.3.3 Dosažené výsledky

Praktické ověření probíhalo přesně dle schématu na obrázku 4.5. V programu ControlDesk jsme si vytvořili jednoduché uživatelské prostředí pro ovládání experimentu a logování dat (obrázek 4.30). Pomocí tohoto prostředí můžeme za běhu povolovat učení lokálních modelů, upravovat rychlost učení hodnoticí funkce, upravovat parametry referenčního signálu, ovládat zesílení P regulátoru a upravovat mnoho jiných parametrů.



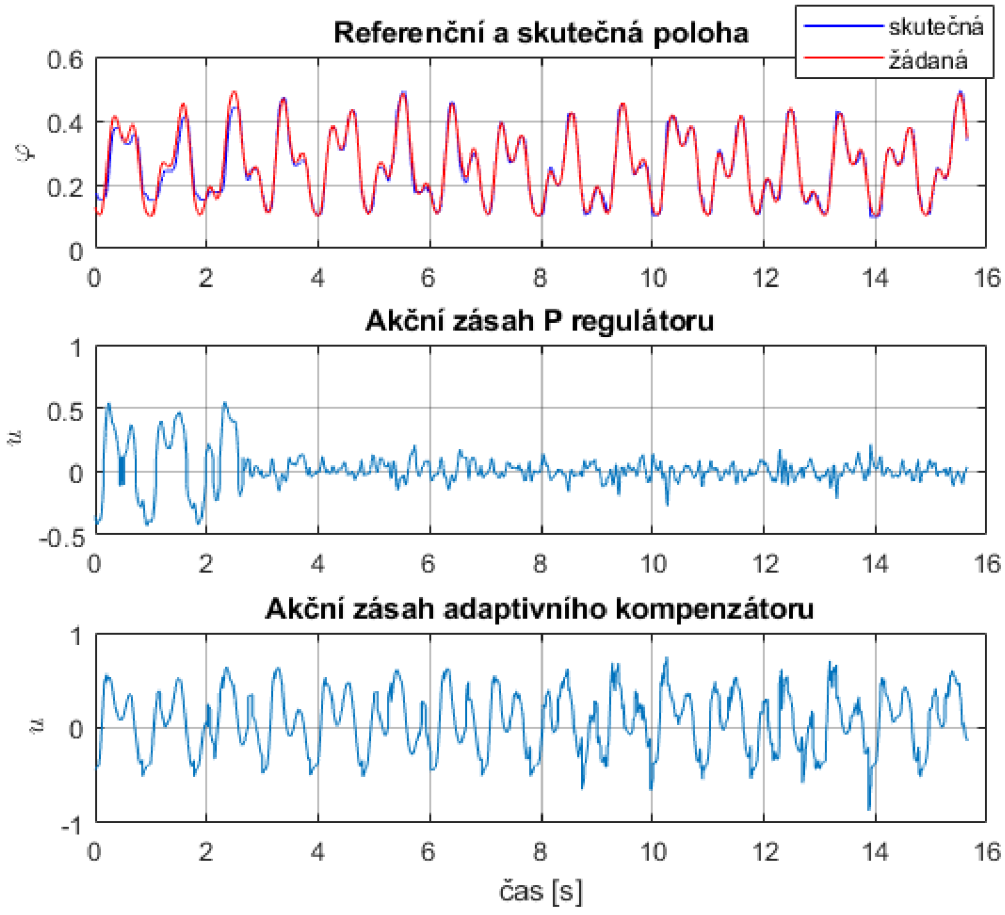
Obrázek 4.30: Rozhraní v Control Desku

Experimentální ověření jsme provedli s EGR ventilem a podstatné parametry adaptivního kompenzátoru jsou uvedeny v tabulce 4.3. Ve zpětnovazební regulační smyčce byl pouze P regulátor, jehož zesílení bylo během experimentu nastaveno na hodnotu 5. Na obrázku 4.31 jsou zobrazeny průběhy (shora dolů) skutečné a žádané polohy, akční zásah P regulátoru a akční zásah adaptivního kompenzátoru.

V prvních sekundách experimentů vidíme, že skutečná poloha EGR ventilu sleduje žádanou polohu velmi zhruba, je mezi nimi poměrně velký rozdíl. Také vidíme, že akční zásah P regulátoru je poměrně velký, rozsahem přibližně rovný akčnímu zásahu adaptivního kompenzátoru, u kterého ale ještě není

Tabulka 4.3: Některé parametry adaptivního kompenzátoru pro experimentální ověření s EGR ventilem

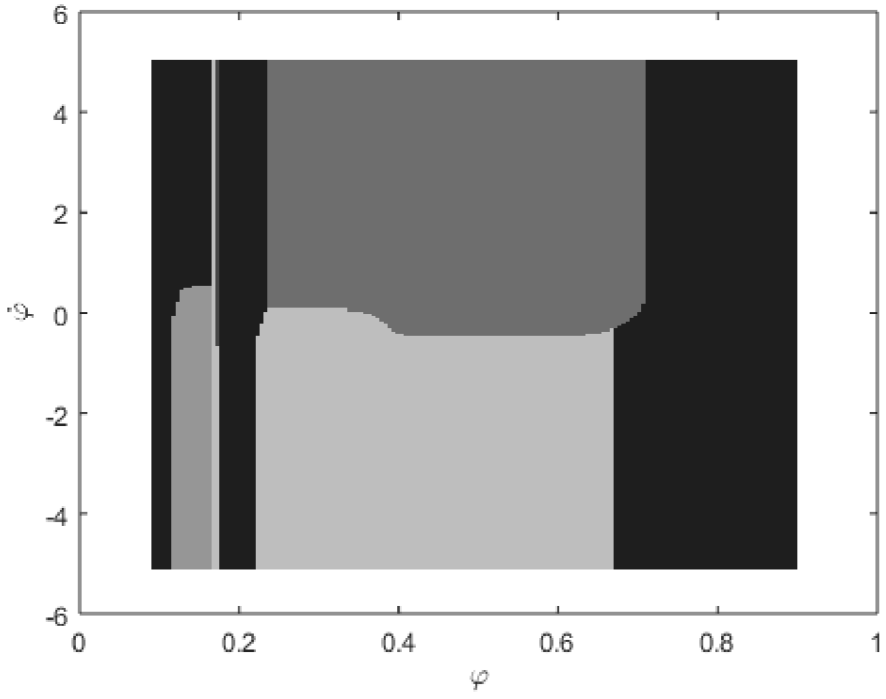
počet lokálních modelů	5
σ^2	0.1
$\mathbf{Q}$	diagonální matice s prvky 1e-4
$\mathbf{P}_{init}$	diagonální matice s prvky 1e-4
η	0.5 v prvních čtyřech sekundách simulace
	0.05 po zbytek simulace
v_{tr}	0



Obrázek 4.31: Průběh skutečné a žádané polohy, akční zásah P regulátoru a akční zásah dopředného kompenzátoru

povolena adaptivita lokálních modelů. V čase experimentu tři sekundy je povololeno učení lokálních modelů a od čtyř sekund se zpomalí učení hodnoticí funkce. Od tohoto okamžiku můžeme pozorovat, že skutečná poloha mnohem lépe sleduje žádanou a akční zásah P regulátoru se radikálně sníží. Většinu akčního zásahu tedy převezme dopředný kompenzátor a můžeme konstatovat, že dopředná kompenzace funguje.

Na obrázku 4.32 vidíme segmentaci stavového prostoru po 15-ti sekundách experimentu. Vidíme, že každý z pěti modelů má určitou oblast platnosti v mapovacím prostoru. Po většinu rozsahu polohy je platné rozdělení dle směru rychlosti a v okolí LH pozice pokrývá lokální model oba směry rychlosti. Předpokladům ale neodpovídá to, že v okolí LH pozice se vyskytují 2-3 lokální modely.



Obrázek 4.32: Segmentace mapovacího prostoru hodnoticí funkce - EGR ventil, reálný experiment

4.4 Poznámky ze simulačního a experimentálního ověření

V této části uvedeme několik poznámek k simulačnímu ověření, jež by mohly být inspirací pro budoucí vývoj:

Vliv výpočetního kroku algoritmu

Během simulací jsme zjistili poměrně zásadní vliv výpočetního kroku na funkci celého algoritmu a reálné experimenty nám to pouze potvrdily. Většinou obecně platí (regulace, estimace parametrů, numerické metody, atd.), že menší výpočetní krok vede k lepším výsledkům a větší výpočetní krok k horším výsledkům.

Pro tuto více-modelovou adaptivní metodu ale platí, že nejlepších výsledků nedosahuje ani při nejkratším, ani při nejdelším výpočetním kroku. Optimum leží někde uprostřed. To, že nejlepších výsledků nedosáhneme při dlouhém výpočetním kroku je přirozené, ale to, že od určitého hranice se s kratším výpočetním krokem funkce algoritmu zhoršuje, neodpovídá očekávání. Při velmi krátkém výpočetním kroku dochází k tomu, že veškerou dynamiku se snaží zachytit pouze jeden lokální model.

Vysvětlujeme si to tím, že pokud je výpočetní krok velmi malý, je rozdíl mezi jednotlivými lokálními modely v každém výpočetním kroku poměrně nízký a vzhledem k šumu v měřených signálech není snadné rozlišit jednotlivé lokální modely. Dle našeho zkoumání leží optimální výpočetní krok pro naše elektromechanické aktuátory v pásmu 1..5 ms.

Znovuaktivování lokálního modelu

Během experimentálního ověření, ale i během simulací se stávalo, že odhad parametrů některého lokálního modelu nekonverguje ke skutečným parametrům. Tato situace je obzvláště nepříjemná, pokud ostatní modely ještě poměrně intenzivně podléhají učení a mohou se snažit pojmout i dynamiku modulu, jehož parametry divergují. Hodnoticí funkce se poté chybně naučí segmentaci a divergujicí model je navždy vyloučen z procesu učení.

V původní publikaci je popsáno možné řešení, jak rozpoznat tuto situaci, jak detekovat divergující model, jak tento model znovu „resetovat“ a zpřístupnit pro nový odhad parametrů.

Adaptivní pokrytí mapovacího prostoru Gaussovými jádry

K dalšímu studiu se nabízí řešení otázky, zda nelze prostor mapování pokrýt Gaussovými jádry automaticky procesem adaptace tak, aby byla Gaussovy jádra pouze v místech, kde se mohou nacházet stavy soustavy a s odpovídajícím rozptylem. Při bližším pohledu se většinou ukáže, že stavový prostor dané soustavy pokrývá pouze malou část vymezeného pravidelného mapovacího prostoru. Podobné algoritmy již existují pro neuronové sítě s radiální bázovou funkcí [32][33][34], které mají mnoho společného s použitou hodnoticí funkcí.

Kapitola 5

Filtrace a derivování digitálního signálu na FPGA

Výpočetní krok vlastního regulačního algoritmu se může pohybovat v řádu desítek hertz až jednotek kilohertz. V případě, že bychom vzorkovali a derivovali analogový signál na stejné frekvenci, mohli bychom implementovat digitální zpracování signálu na procesor vedle regulačního algoritmu. V kapitole 2.2 jsme si řekli, že získávání derivací digitálního signálu je klíčově pro správnou funkci algoritmu a derivace jsou silně ovlivněny přítomným šumem.

Vzorkujeme-li analogový signál frekvencí vyšší než odpovídá Nyquistově frekvenci [1], mluvíme o tzv. *oversamplingu*, ze kterého vyplývají tyto výhody [35][36]:

- snižují se požadavky na anti-aliasingový filtr
- dodatečným postprocessingem můžeme získat vyšší rozlišení signálu
- zvyšujeme odstup signálu od šumu (signal-to-noise ratio)

Implementujeme-li digitální zpracování signálu na FPGA, můžeme vzorkovat a zpracovávat signál až v řádu desítek až stovek megahertz a tím velmi těžit z výhod *oversamplingu*.

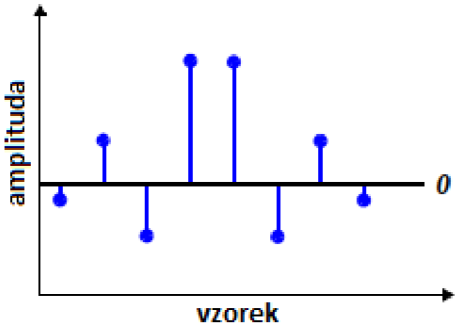
5.1 Implementace FIR filtrů

FIR filtr můžeme vyjádřit pomocí následující rovnice:

$$y(n) = \sum_{k=0}^{N-1} w(k)x(n-k) \tag{5.1}$$

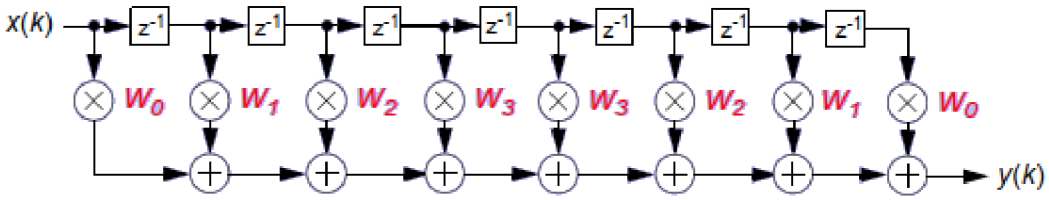
[1] dvojnásobek nejvyšší frekvence užitečného signálu

kde $x(n)$ představuje vstupní signál, $y(n)$ výstupní signál, n číslo vzorku a $w(k)$ koeficienty filtru délky N. Rovnice 5.1 vyjadřuje rovněž konvoluci vstupního signálu s prvky filtru. Nemáme-li speciální požadavky na filtr, vede většina návrhových metod na koeficienty, jež jsou symetrické podle středu. Například na obrázku 5.1 jsou znázorněny koeficienty FIR filtru, impulsní odezva, která má osm symetrických prvků a tím se jedná o filtr sedmého řádu.



Obrázek 5.1: Symetrická impulsní odezva filtru s lineární fází (převzato z [18])

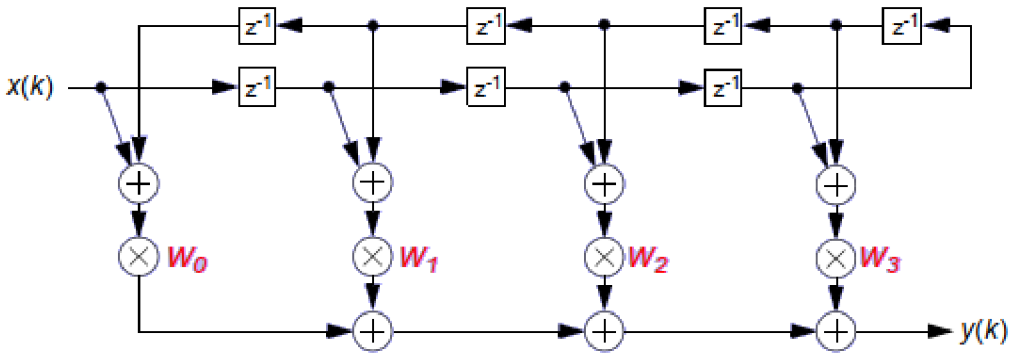
Z této symetrie plyne jedna významná žádaná vlastnost filtru - lineární fáze (změna fáze s frekvencí je lineární). Rozvineme-li rovnici 5.1 pro FIR filtr sedmého řádu na základní aritmetické operace, bude vypadat implementace podle obrázku 5.2. Této struktuře se říká *přímá*, anglicky *direct form*.



Obrázek 5.2: Přímá struktura FIR filtru (převzato z [18])

Koeficienty w_0 - w_3 představují symetrické koeficienty filtru. Ze symetrie plyne další výhoda, můžeme filtr implementovat v takzvané struktuře pro lineární fázi, anglicky *linear-phase form* (obrázek 5.3). Díky tomu jsme snížili na polovinu počet operací násobení[2].

[2]Důvod, proč se obecně snážíme vyhnout operaci násobení je ten, že skládáme-li ze základních hradel (či na FPGA) násobičku, násobení dvou n-bitových čísel obecně vyžaduje n^2 základních bloků. V případě sčítání dvou n-bitových čísel je situace mnohem příznivější, vyžaduje pouze $n + n$ základních stavebních bloků



Obrázek 5.3: Struktura pro FIR filtr s lineární fází (převzato z [18])

Dnešní mikroprocesory a mikrokontroléry často obsahují tzv. *floating-point unit* - dedikovanou část pro operace s čísly s plovoucí desetinnou čárkou, a tak není příliš velký rozdíl, zda operujeme s celočíselnými operandy nebo operandy s pevnou či plovoucí desetinnout čárkou. Při implementaci na FPGA je ale mezi těmito operacemi velký rozdíl a operace s čísly s plovoucí desetinnou čárkou je velmi náročná na zdroje FPGA. Proto je snaha se operacím s plovoucí desetinnou čárkou vyhnout a nahradit je operacemi s pevnou desetinnou čárkou nebo ideálně celočíselnými operacemi.

Extrémní situací je, když se nám podaří nahradit násobení konstantou bitovým posunem. Potřebujeme-li násobit nebo dělit číslo mocninou dvou, je tato situace očividná. Např. $4x = x << 2$, $0.125x = x >> 3$[3]. Tato operace je na FPGA téměř „zadarmo“. Méně očividná situace ale nastává při násobení konstantou, která není mocninou dvou. Přesto se ale dá operace rozložit a využít bitový posun (příklad rovnice 5.2, 5.3). Tento rozklad může být při implementaci na FPGA mnohem výhodnější než běžné násobení i kdybychom operaci rozložili na více částí než dvě.

$$5x = 2^2x + 2^0x = (x << 2) + x \tag{5.2}$$

$$14x = 2^4x = 2^1x = (x << 4) - (x << 2) \tag{5.3}$$

[3]Znaky << a >> označují bitový posun doleva, respektive doprava o daný počet míst.

5.2 Představení jednotlivých variant

Při implementaci různých variant filtrů na FPGA budeme sledovat využití zdrojů FPGA (počet registrů, LUT, případně vestavěných násobiček, které jsou potřeba pro implementaci dané varianty).

Základní rozdělení bude podle typu filtru zvoleného pro návrh koeficientů derivátoru:

- Maximally flat filtr (podrobnosti v kapitole 2.2.4) - očekáváme celočíselné koeficienty
- Libovolná jiná metoda (použijeme equiripple filtr 2.2.3) - očekáváme neceločíselné koeficienty

Další rozdělení bude podle struktury filtru:

- přímá, *direct form*
- struktura pro filtr s lineární fází, *linear-phase form*

Přirozeně očekáváme menší využití FPGA pro *linear-phase form*. Poslední rozdělení bude podle realizace násobení:

- celočíselné násobení - Pro násobení využijeme bloků *Multiply*, které jsou dostupné v LabVIEW FPGA módu.
- celočíselné násobení pomocí bitového posunu - Veškeré násobení budeme realizovat rozkladem na součet násobků mocnin dvou. Je nutné zvláště ošetřit bitový posun pro záporná čísla. Podrobnosti budou uvedeny v následující sekci.
- celočíselné násobení pomocí bitového posunu - Stejné jako předchozím případě, ale s tím rozdílem, že pro bitový posun využijeme bloku *Scale By Power of Two*, který nabízí přímo knihovna LabVIEW. LabVIEW se postará samo o ošetření bitového posunu pro záporná čísla.
- fixed point - Násobení s využitím bloku *Multiply* v LabVIEW, ale všechny signály i koeficienty filtru budou datového typu *fixed-point*. Budeme se snažit zachovat co nejmenší bitovou šíři všech signálů při zachování rozumné přesnosti filtru.

Z předchozích možností jsme vytvořili šest variant, které jsou uvedeny v tabulce 5.1. Tyto varianty implementujeme na FPGA a porovnáme využití jeho zdrojů.

Tabulka 5.1: Varianty implementovaných FIR filtrů

označení	typ filtru[4]	délka filtru	K	L	struktura filtru[5]	další
MA1	MA	5	3	0	D	násobení bitovým posunem
MA2	MA	9	7	0	D	násobení bitovým posunem
MA3	MA	9	7	0	L	násobení bitovým posunem
MA4	MA	9	7	0	L	celočíselné násobení
MA5	MA	9	7	0	L	násobení LabVIEW bitovým posunem
EQ1	EQ	9	–	–	L	fixed-point násobení

Tabulka 5.2: Parametry FPGA na zařízení myRIO-1900[37]

počet LUT	17600
počet registrů	35200
počet DSP slice	80
vestavěná pamět (Mb)	2,1

5.3 Experimentální ověření

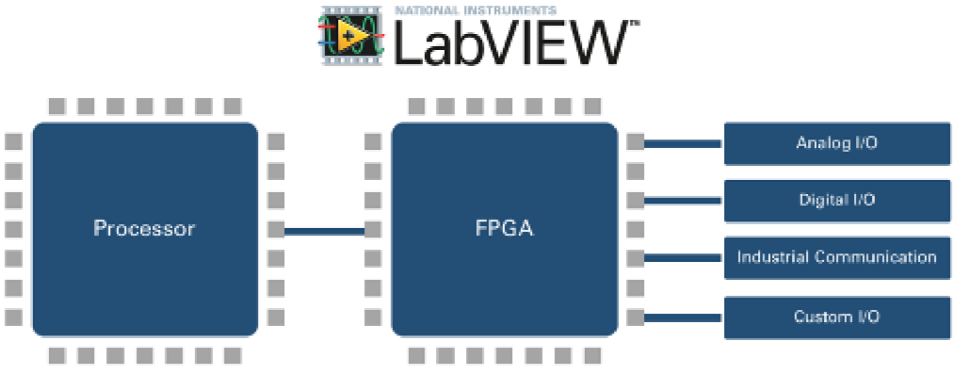
Test jednotlivých variant proběhl na zařízení myRIO-1900[37] od firmy National Instruments. Jedná se o samostantný měřicí a řídicí HW, který obsahuje analogové a digitální vstupy a výstupy, dvoujádrový ARM procesor *Cortex A-9* a FPGA od firmy Xilinx. Procesor i FPGA jsou na jednom integrovaném obvodu, tzv. *System on Chip - SoC* z rodiny *Zynq*[38] od firmy Xilinx.

Konkrétní parametry FPGA na myRIO-1900 jsou uvedeny v tabulce 5.2. Zařízení myRIO-1900 je založeno na *RIO*[21] architektuře, která kombinuje *real-time* procesor, programovatelné FPGA a vstupně/výstupní moduly. Klíčové je propojení procesoru a vstupně/výstupních modulů skrz programovatelné FPGA (obrázek 5.4), na kterém se dají vytvořit velmi rychlé algoritmy pro zpracování digitálního signálu.

Abychom potvrdili funkčnost jednotlivých variant filtrů, budeme na *real-time* části generovat sinusový signál, tento signál budeme pomocí fronty přenášet do FPGA, kde bude signál filtrován/derivován s výpočetní frekvencí $1,25MHz$. Po filtraci signál přeneseme pomocí druhé fronty zpět ro RT, kde si signál zobrazíme a uložíme pro pozdější zpracování. Tím potvrdíme funkčnost filtru/derivátoru.

[4] MA - maximally flat, EQ - equiripple

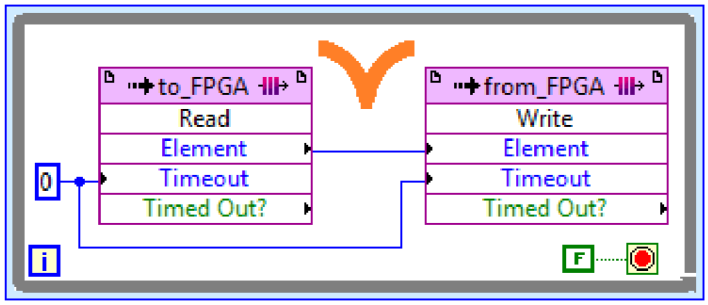
[5] D - přímá struktura, L - struktura pro filtr s lineární fází



Obrázek 5.4: NI RIO architektura (převzato z [21])

Na FPGA budou kromě našeho algoritmu také části, které se věnují obsluze front do/z RT části. Abychom odlišili zdroje FPGA, které jsou využity pro implementaci filtrů od částí, které jsou využity pro obsluhu front, vytvořili jsme si prázdný program na FPGA, který čte data ze vstupní fronty a ihned je posílá do výstupní fronty. Implementace tohoto programu ve vývojovém prostředí *LabVIEW* je na obrázku 5.5. Oranžovou šipkou je znázorněna část, do které budeme vkládat filtry. Tento „prázdný“ program využívá z FPGA:

- 10336 registrů
- 11057 LUT
- 0 DSP slice



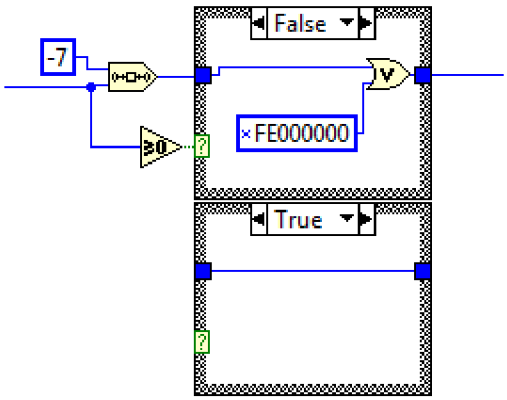
Obrázek 5.5: Ukázka programu na FPGA, který přijímá data pomocí fronty z RT části a ihned je odesílá přes druhou frontu zpět do FPGA

Tabulka 5.3: Implementantace násobení mocninou dvou pro 32-bitový celočíselný datový typ

	$n > 0$	$n < 0$
$x \geq 0$	$x << n$	$x >> n$
$x < 0$	$x << n$	$(x >> n) \mid (0xFFFFFFFF << (32 - n))$

5.3.1 Bitový posun pro čísla vyjádřená pomocí dvojkového doplňku

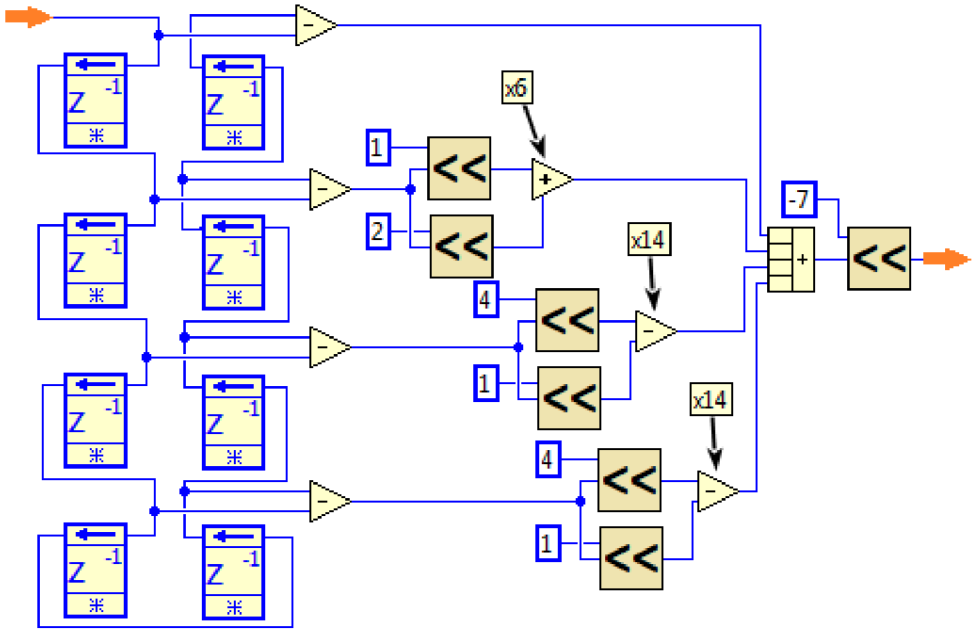
Již jsme si řekli, že násobení mocninou dvou můžeme nahradit bitovým posunem. Tohle ale doslova platí pro nezáporná čísla (*unsigned integer*). Jsou-li záporná čísla vyjádřena pomocí dvojkového doplňku *two's complement*, musíme zvlášť přistoupit k situaci, pokud žádáme bitový posun doleva. Všechny čtyři varianty pro kladná a záporná čísla a násobení mocninou dvou větší nebo menší než nula jsou znázorněny v tabulce 5.3. Implementaci pomocí LabVIEW vidíme na obrázku 5.6



Obrázek 5.6: Celočíselné dělení konstantou 128 pomocí datového posunu pro celočíselný znaménkový typ

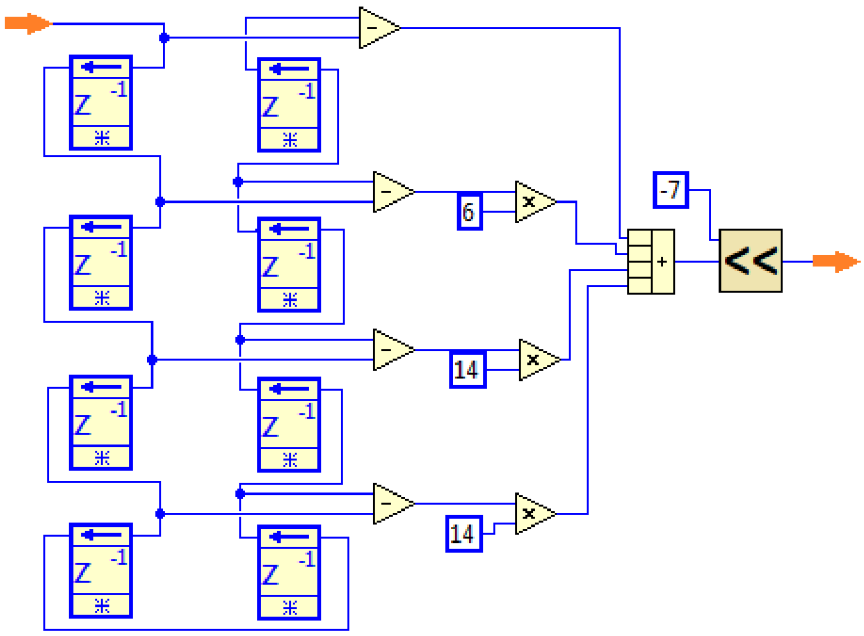
5.3.2 Implementace jednotlivých variant

Na následujících obrázcích si ukážeme implementaci jednotlivých variant filtrů pomocí LabVIEW. Budou zobrazeny pouze filtry se strukturou *linear-phase form*. Na obrázku 5.7 je zobrazena implementace varianty *MA3* a veškeré násobení je realizováno pomocí vlastního bitového posunu. Bloky bitového posunu obsahují program zobrazený na obrázku 5.6. Díky tomu, že veškeré koeficienty filtru mají společného dělitele, jež je mocninou dvou (podrobnosti v kapitole 2.2.4), můžeme každou větev filtru násobit celým čísel a celý součet potom dělit společným dělitelem. Tím můžeme vytvořit velmi efektivní implementaci filtru, která zabírá velmi malou část FPGA.



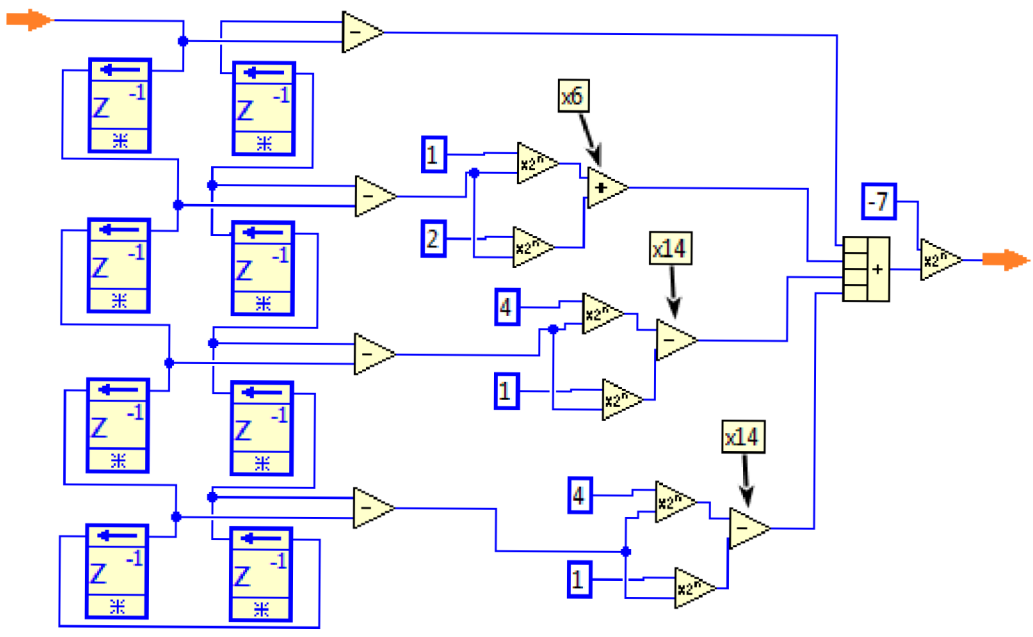
Obrázek 5.7: LabVIEW implementace *MA3* - bitový posun

Další varianta *MA4* je zobrazena na obrázku 5.8. Od předchozí implementace se liší tím, že pro násobení je použito bloků *Multiply*. Optimalizaci násobení má na starost LabVIEW a můžeme očekávat větší využití zdrojů FPGA. Pro dělení jsme použili bitový posun z toho důvodu, protože implementace této funkce na FPGA pomocí LabVIEW má příliš dlouhou cestu signálu a tato funkce se nestihne vykonat během jednoho hodinového taktu FPGA.



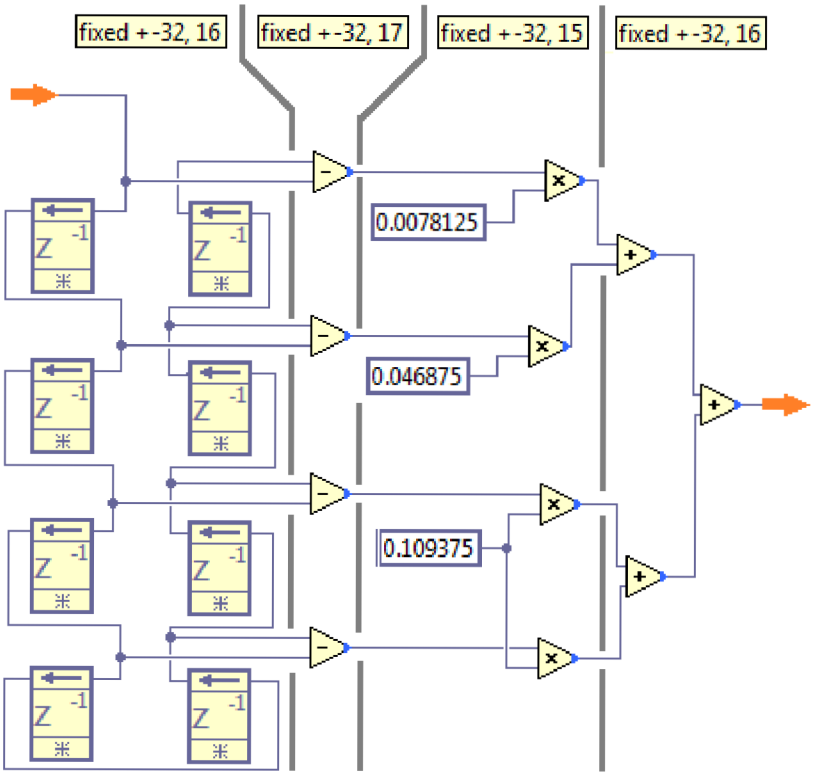
Obrázek 5.8: LabVIEW implementace *MA4* - celočíselné násobení

Následující varianta (obrázek 5.9) využívá bitového posunu implementovaného pomocí bloku v LabVIEW. Tato varianta je zde proto, abychom porovnali, zda je efektivnější naše varianta bitového posunu nebo LabVIEW.



Obrázek 5.9: LabVIEW implementace *MA5* - LabVIEW bitový posun

Poslední a teoreticky nejvíce náročná varianta na zdroje FPGA je na obrázku 5.10. Vycházíme z předpokladu, že koeficienty filtru jsou obecně reálná nesoudělná čísla a celý filtr musíme implementovat pomocí operací s pevnou desetinnou čárkou.



Obrázek 5.10: LabVIEW implementace *EQ1* - LabVIEW fixed-point násobení

5.3.3 Výsledky porovnání jednotlivých variant

Pro každou variantu jsme měli nastavené stejné optimalizace kompilace programu. Byla zvolena optimalizace, která se snaží provést kompilaci a mapovaní za nejkratší čas - *minimum compilation time*. V případě potřeby bychom mohli zvolit jinou variantu (například optimalizace s cílem nižšího využití zdrojů FPGA, snížení spotřeby čipu nebo pro nejkratší cestu signálu). Výsledky využití FPGA jsou shrnuty v tabulce 5.4. Můžeme konstatovat:

- Nejmenší využití zdrojů FPGA má varianta *MA1*. To je ale očekáváme i přes přímou strukturu filtru, protože má tato varianta nejméně koeficientů.
- Z filtrů délky 9 je nejméně náročná na zdroje FPGA varianta *MA5*, těsně následovaná variantou *MA3* (varianta *MA4* má sice nejmenší využití LUT, ale potřebuje 9 DSP slice).
- Implementace LabVIEW bitového posunu pro celá čísla je nepatrně efektivnější než námi vytvořená varianta.
- Zdaleka nejvíce zdrojů využívá varianta *EQ1* - implementace s pomocí operací s pevnou desetinnou čárkou.

Tabulka 5.4: Porovnání jednotlivých variant implementovaných FIR filtrů

		bez filtru	MA1	MA2	MA3	MA4	MA5	EQ1
počet elementů	registry	10336	10402	10466	10466	10466	10466	10465
	LUT	11057	11105	11284	11252	112201	11246	11626
	DSP48s	0	0	0	0	9	0	16
rozdíl vůči *bez filtru*	registry	–	66	130	130	130	130	129
	LUT	–	48	227	195	144	189	569
	DSP48s	–	0	0	0	9	0	16

Kapitola 6

Závěr

Pro tuto práci byly stanoveny tři hlavní cíle a to:

1. Přeformulování metody více-modelového řízení uvedené v publikaci [15] pro použití jako dopředný kompenzátor

2. Úprava a optimalizace algoritmu této metody pro běh na *real-time* HW a experimentální ověření

3. Implementace filtrace a předzpracování signálu senzoru na FPGA

První dva body spolu úzce souvisí a vlastní vykonaná práce je popsána v kapitole 4. Jako úvod do dané problematiky slouží kapitoly 2.1, 2.3 a 2.4 v rešeršní části této práce. Motivací bylo použít zajímavou metodu zpětnovazební více-modelové regulace v původní podobě dle publikace [15] pro řízení elektromechanických aktuátorů, což se ukázalo jako neúspěšné. Došlo tedy k přeformulování této metody pro použití jako dopředný kompenzátor a simulačnímu a experimentálnímu ověření. První dva cíle se podařilo naplnit, ovšem ukázalo se, že navržená metoda kompenzace obsahuje velké množství parametrů, z nichž ne všechny lze exaktně určit a proto je praktické využití této metody poměrně omezené.

Třetí cíl (*Implementace filtrace a předzpracování signálu senzoru na FPGA*) se ukázal jako zajímavější a s širším praktickým využitím. Kvůli adaptivnímu dopřednému kompenzátoru, jež vyžaduje pro svou funkci také derivace vstupního signálu jsme se zaměřili především na filtry, jejichž výstupem je numerická derivace vstupního signálu. Věříme, že se nám podařilo zmapovat danou oblast vědění a vybrat nejdůležitější metody získávání derivace digitálního signálu (kapitola 2.2 v rešeršní části). Z těchto metod jsme mohli vybrat jednu, jež vede při určitých návrhových parametrech na implementaci, která

se dá vytvořit pouze pomocí celočíselného násobení, sčítání a bitového posunu, což je velmi výhodné při implementaci na FPGA, jak je uvedeno v kapitole 5.

6.1 Přínosy disertační práce

- Přeformulování více-modelového řízení dle publikace [15] pro použití jako dopředný kompenzátor včetně simulačního ověření. Z experimentů během práce na této části vznikla publikace [25], která se věnuje jednoduššímu adaptivnímu dopřednému kompenzátoru a porovnává různá vyjádření matematického modelu v kompenzátoru.

- Došlo k úpravě předchozí metody tak, aby mohla být implementována pro běh na real-time HW. Jako real-time zařízení pro řízení aktuátoru sloužil modulární HW od firmy dSPACE pro RCP a HIL simulace. Součástí úprav byla také optimalizace s cílem snížit výpočetní a paměťovou náročnost algoritmu, protože původní verze pro simulační ověření byla příliš výpočetně náročná a nebyla by na námi vybraném HW schopna běžet s daným výpočetním krokem.

- Pro experimentální ověřování navržených algoritmů a pro měření parametrů elektromechanických aktuátorů vznikl jednoduchý stav, který se skládá z výkonového posílení signálů z real-time HW a přizpůsobení výstupního signálu z polohového sensoru elektromechanických aktuátorů pro vstup do měřicího HW. Především jsme navrhli a vytvořili analogový zesilovač s nízkým zkreslením, který je schopen dodat dostatečný výkon pro řízení elektromechanických aktuátorů. Tento analogový zesilovač používáme pouze při měření dat, jinak je aktuátor řízení výkonovým h-můstkem.

- Zmapování oblasti získávání derivací digitálního signálu pomocí digitálních filtrů s konečnou impulsní odezvou. Podstatné metody jsou uvedeny v kapitole 2.2 v rešeršní části této práce. Z informací získaných během studia této oblasti vzniká článek[3], jež bychom rádi publikovali ve vědeckém časopise.

- Využili jsme zjištění, že určité návrhové parametry *maximally flat* derivátoru vedou na celočíselné koeficienty impulsní odezvy filtru. Tím je možno tyto filtry velice jednoduše implementovat na FPGA s minimem využitých zdrojů. Vzniká příspěvek [39] na vědeckou konferenci, kde bychom chtěli prezentovat toto zjištění a ukázat, jakých úspor využití

zdrojů FPGA je možno s touto metodou dosáhnout oproti běžným filtrům s neceločíselnými koeficienty.

6.2 Možnosti dalšího výzkumu v této oblasti

Zde prezentovaná metoda adaptivního dopředného kompenzátoru byla ověřena při řízení elektromechanických aktuátorů. Dosažená kvalita regulace určitě není dokonalá a je prostor pro další zlepšení. Adaptivní dopředný kompenzátor má mnoho parametrů, z nichž ne všechny lze exaktně určit. Za další studium by stálo podrobněji prozkoumat jednotlivé parametry, jejich vliv na kvalitu regulace a pokusit se stanovit postup, jak jednotlivé parametry nebo aspoň většinu z nich určit.

Představený adaptivní kompenzátor jsme ověřovali při řízení elektromechanických aktuátorů, ale za úvahu by stálo, zda nelze najít další oblasti využití pro tento algoritmus. Především pro soustavy, kde se nevyskytuje problém podobný suchému tření, jež výrazně komplikuje funkci celého algoritmu.

Některé další oblasti ke studiu a rozvinutí jsou shrnuty v kapitole 4.4. Pro další studium, inspiraci a případně srovnání navrhujeme metodu zvanou *Receptive Field Weighted Regression - RFWR* [40][41], která má se zde prezentovaným algoritmem jisté podobnosti.

Literatura

[1] B. Grepl, R.and Lee. Modeling, parameter estimation and nonlinear control of automotive electronic throttle using a rapid-control prototyping technique. *International Journal of Automotive Technology*, 11(4):601–610, Aug 2010.

[2] Abraham. Savitzky and M. J. E. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, 36(8):1627–1639, 1964.

[3] M. Brablc and V. Sova. Summary of differentiation filtering algorithms for parameter estimation purposes. *bude publikováno.*

[4] T. Parks and J. McClellan. Chebyshev approximation for nonrecursive digital filters with linear phase. *IEEE Transactions on Circuit Theory*, 19(2):189–194, March 1972.

[5] L.R. Rabiner and B. Gold. *Theory and application of digital signal processing.* Prentice-Hall signal processing series. Prentice-Hall, 1975.

[6] O. Herrmann. On the approximation problem in nonrecursive digital filter design. *IEEE Transactions on Circuit Theory*, 18(3):411–413, May 1971.

[7] I. W. Selesnick. Maximally flat low-pass digital differentiator. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 49(3):219–223, March 2002.

[8] I. W. Selesnick. Maximally flat low-pass digital differentiators. http://eeweb.poly.edu/iselesni/lowdiff/lowdiff.m. Přístup: 14.08.2018.

[9] M. Athans, K. . Dunn, C. S. Greene, W. H. Lee, N. R. Sandell, I. Segall, and A. S. Willsky. The stochastic control of the f-8c aircraft using the multiple model adaptive control (mmac) method. In *1975 IEEE Conference on Decision and Control including the 14th Symposium on Adaptive Processes*, pages 217–228, Dec 1975.

[10] M. Athans, D. Castanon, K. Dunn, C. Greene, Wing Lee, N. Sandell, and A. Willsky. The stochastic control of the f-8c aircraft using a multiple model adaptive control (mmac) method–part i: Equilibrium flight. *IEEE Transactions on Automatic Control*, 22(5):768–780, October 1977.

[11] S. M. Haris and W. S. Aboud. Weighted mixing multiple model adaptive control in a mechatronic active suspension system application. In *The SICE Annual Conference 2013*, pages 1485–1490, Sept 2013.

[12] L. Seban, N. Sahoo, and B. K. Roy. Multiple model based predictive control of magnetic levitation system. In *2014 Annual IEEE India Conference (INDICON)*, pages 1–5, Dec 2014.

[13] Ren-Guang Wang, Zhao-Du Liu, Zhi-Quan Qi, Yue-Feng Ma, and Hai-Feng Cui. Multiple model adaptive control of antilock brake system via backstepping approach. In *2005 International Conference on Machine Learning and Cybernetics*, volume 1, pages 591–595, Aug 2005.

[14] K. S. Narendra and J. Balakrishnan. Adaptive control using multiple models. *IEEE Transactions on Automatic Control*, 42(2):171–187, Feb 1997.

[15] S. Fabri and V. Kadirkamanathan. *Functional Adaptive Control: An Intelligent Systems Approach*. Communications and Control Engineering. Springer London, 2001.

[16] Monson H. Hayes. *Statistical Digital Signal Processing and Modeling*. John Wiley & Sons, Inc., 1996.

[17] M. I. Jordan and R. A. Jacobs. Hierarchical mixtures of experts and the em algorithm. In *Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan)*, volume 2, pages 1339–1344 vol.2, Oct 1993.

[18] University of Strathclyde and Steepest Ascent. *DSPforFPGA Primer teaching materials*, Q4 2011.

[19] Xilinx. System Generator for DSP. https://www.xilinx.com/products/design-tools/vivado/integration/sysgen.html. Přístup: 12.08.2018.

[20] National Instruments. LabVIEW FPGA Module. http://sine.ni.com/nips/cds/view/p/lang/cs/nid/11834. Přístup: 12.08.2018.

[21] National Instruments. LabVIEW RIO architektura. http://www.ni.com/white-paper/10894/en/. Přístup: 08.08.2018.

[22] MathWorks. MATLAB. https://www.mathworks.com/products/matlab.html. Přístup: 26.08.2018.

[23] Pavel Holoborodko. Smooth noise robust differentiators. http://www.holoborodko.com/pavel/numerical-methods/numerical-derivative/smooth-low-noise-differentiators, 2008.

[24] L. Ljung. *System Identification: Theory for the User*. Pearson Education, 1998.

[25] M. Brablc, V. Sova, and R. Grepl. Adaptive feedforward controller for a dc motor drive based on inverse dynamic model with recursive least squares parameter estimation. In *2016 17th International Conference on Mechatronics - Mechatronika (ME)*, pages 1–5, Dec 2016.

[26] MathWorks. HDL Coder. https://www.mathworks.com/products/hdl-coder.html. Přístup: 12.08.2018.

[27] MathWorks. Simulink. https://www.mathworks.com/products/simulink.html. Přístup: 26.08.2018.

[28] mechlab - Mechatronics Laboratory. http://mechlab.fme.vutbr.cz. Přístup: 18.08.2018.

[29] alldatasheet.com. LM4780. http://www.alldatasheet.com/view.jsp?Searchword=Lm4780. Přístup: 26.08.2018.

[30] dSPACE Inc. Real-Time Interface (RTI). https://www.dspace.com/en/inc/home/products/sw/impsw/realtimeinterf.cfm. Přístup: 18.08.2018.

[31] dSPACE Inc. ControlDesk. https://www.dspace.com/en/inc/home/products/sw/experimentandvisualization/controldesk.cfm. Přístup: 28.08.2018.

[32] Douglas L. Reilly, Leon N. Cooper, and Charles Elbaum. A neural model for category learning. *Biological Cybernetics*, 45(1):35–41, Aug 1982.

[33] Qiao J. Han H, Chen Q. Research on an online self-organizing radial basis function neural network. *Neural Computing & Applications*, Jul 2010.

[34] Michael R. Berthold, Forschungszentrum Informatik, and Jay Diamond. Boosting the performance of rbf networks with dynamic decay adjustment. In *Advances in Neural Information Processing Systems*, pages 521–528. MIT Press, 1995.

[35] E.C. Ifeachor and B.W. Jervis. *Digital Signal Processing: A Practical Approach*. Electronic systems engineering series. Prentice Hall, 2002.

[36] T. Claasen, W. Mecklenbrauker, J. Peek, and N. van Hurck. Signal processing method for improving the dynamic range of a/d and d/a converters. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(5):529–538, October 1980.

[37] National Instruments. myRIO. http://www.ni.com/cs-cz/support/model.myrio-1900.html. Přístup: 08.08.2018.

[38] Xilinx. Zynq-7000 SoC. https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html. Přístup: 08.08.2018.

[39] M.Brablc V. Sova and R.Grepl. Fpga implementation of multiplierless low-pass fir differentiator. *bude publikováno*.

[40] Stefan Schaal and Christopher G. Atkeson. Constructive incremental learning from only local information. *Neural Computation*, 10:2047–2084, 1998.

[41] Robert Grepl, Vaclav Sova, and Jan Chalupa. Adaptive control of electro-mechanical actuator using receptive field weighted regression. In Ryszard Jabłoński and Tomas Brezina, editors, *Advanced Mechatronics Solutions*, pages 621–626, Cham, 2016. Springer International Publishing.

Příloha A

Seznam publikací autora

[A1] BRABLC M., SOVA V., GREPL R.: *Adaptive Feedforward Controller for a DC Motor Drive based on Inverse Dynamic Model with Recursive Least Squares Parameter Estimation*, In Mechatronika (ME), 2016 17th International Conference on Mechatronics., 2016

[A2] HRBÁČ, Z.; SOVA, V.; GREPL, R.: *Sensorless Speed Control of BLDC Motor using EKF with Computed Inputs and Disturbance.*, In Advanced Mechatronics Solutions. Advances in Intelligent Systems and Computing., 2015

[A3] GREPL, R.; SOVA, V.; CHALUPA, J. : *Adaptive Control of Electro-Mechanical Actuator using Receptive Field Weighted Regression.*, In Advanced Mechatronics Solutions. Advances in Intelligent Systems and Computing., 2015

[A4] SOVA, V.; CHALUPA, J.; GREPL, R. : *Fault Tolerant BLDC Motor Control for Hall Sensors Failure.*, In Automation, Computing and Manufacturing for New Economic Growth., 2015

[A5] CHALUPA, J.; GREPL, R.; SOVA, V. : *Design of Configurable DC Motor Power-Hardware-In-the-Loop Emulator for Electronic-Control-Unit Testing.*, In Automation, Computing and Manufacturing for New Economic Growth., 2015

[A6] LAMBERSKÝ, V.; Vejlupek. J.; SOVA, V.; GREPL, R.: *Creating support for fully automatic code generation for Cerebot MX7cK hardware from Simulink environment*, International Journal of Circuits, Systems and Signal Processing, 2014

[A7] SOVA, V.; GREPL, R. : *Hardware in the Loop Simulation Model of BLDC Motor Taking Advantage of FPGA and CPU Simultaneous Implementation.*, In Mechatronics 2013: Recent Technological and Scientific Advances. Mechatronics. Cham, Heidelberg, New York, Dordrecht, London: Springer International Publishing,, 2013

Příloha B

Seznam příloh

- adresář ***throttle_model*** - obsahuje simulační model škrticí klapky včetně estimovaných parametrů
- adresář ***Throttle_MMACasFF*** - obsahuje skripty do prostředí MATLAB pro simulační ověření dopředného kompenzátoru pro elektronickou škrticí klapku
- adresář ***EGR_MMACasFF\Matlab\01_MMAC_5M*** - obsahuje skripty do prostředí MATLAB pro simulační ověření dopředného kompenzátoru pro EGR ventil
- adresář ***EGR_MMACasFF\Simulink\01_MMAC_5M*** - obsahuje Simulink model pro simulační ověření dopředného kompenzátoru pro EGR ventil
- adresář ***EGR_MMACasFF\Simulink\xx_data*** - obsahuje naměřené záznamy skutečných průběhů signálů EGR ventilu pro testování kompenzace
- adresář ***EGR_dSPACE*** - obsahuje Simulink model pro dSPACE, skript parametrů a rozhraní v ControlDesk
- adresář ***FPGA_LabVIEW*** - LabVIEW projekt pro myRIO - implementace a porovnání filtrace