UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Jure Menart

Frekvenčna regulacija radiofrekvenčne votline

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Dušan Kodek

Ljubljana, 2014

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Pospeševalniki delcev uporabljajo za pospeševanje radiofrekvenčne votline. Pri delovanju je zelo pomembno, da zunanji dejavniki, kot so temperatura in mehanski pritiski, kar najmanj vplivajo na resonančno frekvenco votline. To se doseže s sistemom za frekvenčno regulacijo, ki s pomočjo koračnih motorjev krmili regulacijski vijak votline. Razvijte in izdelajte sistem za frekvenčno regulacijo z uporabo že obstoječe merilne opreme podjetja Instrumentation Technologies. Delovanje sistema preizkusite in ovrednotite njegovo uspešnost.

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani/-a Jure Menart,

z vpisno številko 63000217,

sem avtor/-ica diplomskega dela z naslovom:

Frekvenčna regulacija radiofrekvenčne votline.

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom prof. dr. Dušan Kodeka,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 1. 6. 2014 Podpis avtorja/-ice:

Zahvala

Zahvaljujem se mentorju prof. dr. Dušanu Kodeku za nasvete in vodenje pri izdelavi tega diplomskega dela. Zahvaljujem se tudi sodelavcem pri podjetju Instrumentation Technologies d.d. za podporo in nasvete. Ne nazadnje se zahvaljujem še družini, še posebno ženi Kristini, za podporo v času študija.

Kazalo

Seznam uporabljenih kratic in simbolov

ADC	*(angl.)*	Analog to Digital Converter; analogno-digitalni pretvornik.
COM	*(angl.)*	Computer On Module; računalnik na modulu.
CORBA	*(angl.)*	Common Object Request Broker Architecture; arhitektura posrednikov skupnih objektov.
EMMA	*(angl.)*	Electron Machine for Many Applications; ime pospeševalnika v Angliji.
FFAG	*(angl.)*	Fixed Field Alternating Gradient; nespremenljivo polje izmenični gradient (tip pospeševalnika, magnetno polje).
FPGA	*(angl.)*	Field Programmable Gate Array; programirljivo polje vrat.
ICB	*(angl.)*	Inter-Connection Board; osrednja povezovalna plošča.
IF	*(angl.)*	Intermediate Frequency; vmesna frekvenca.
IP	*(angl.)*	Internet Protocol; internetni protokol.
LLRF	*(angl.)*	Low Level Radio Frequency; nizkonivojska radijska frekvenca.
LO	*(angl.)*	Local Oscillator; Lokalni oscilator.
LVDS	*(angl.)*	Low-Voltage Differential Signal; nizkonapetostni diferencialni signali.
MO	*(angl.)*	Master Oscillator; glavni oscilator.
PCI	*(angl.)*	Peripheral Component Interconnect; periferna povezava med komponentami.
PID	*(angl.)*	Proportional-Integral-Derivative; proporcionalno-integralno-diferencialni.
RF	*(angl.)*	Radio Frequency; radijska frekvenca.
SCL	*(angl.)*	Serial Command Language; serijski ukazni jezik.
TCP	*(angl.)*	Transmission Control Protocol; transportni kontrolni protokol.
TM	*(angl.)*	Timing Module; časovni modul.
VM	*(angl.)*	Vector Module; vektorski modul.

Povzetek

Pospeševalniki delcev se uporabljajo na veliko različnih področjih, od raziskav o atomski fiziki in raziskav materialov do uporabe v medicinske namene. Kljub temu da obstaja veliko različnih tipov pospeševalnikov delcev, se še vedno odkrivajo tudi novi tipi. Eden takih je neskalirni FFAG (angl. non-scaling Fixed Field Alternating Gradient), ki bi zaradi relativne majhnosti in dobrega izkoristka energije lahko razširil uporabo zdravljenja raka z obsevanjem s protoni in ioni ogljika. V angleškem Daresburyju je bil zgrajen prvi prototip, poimenovan EMMA (angl. Electron Machine for Many Applications). EMMA je le prvi korak pri potrditvi neskalirnega tipa FFAG, saj je načrtovan za pospešitev elektronov in ne težjih delcev (protonov, ionov ogljika).

To diplomsko delo opisuje načrtovanje, razvoj in uporabo sistema za frekvenčno regulacijo EMMA pospeševalniške radiofrekvenčne votline. Problem pri radiofrekvenčnih votlinah je, da se zaradi različnih zunanjih dejavnikov (sprememba temperature okolice, mehanski stres, ...) neprestano frekvenčno premikajo (angl. detuning). Posledica frekvenčnega premika votline je, da elektromagnetno polje v votlini ni optimalno za pospešitev delcev. Zaradi tega morajo biti radiofrekvenčne votline krmiljene tako, da imajo resonančno frekvenco čim bližje predpisani. Ker so naštete spremembe relativno počasne, se lahko problem reši s počasno krmilno zanko.

Za zajem radiofrekvenčnih signalov, njihovo analizo, samo krmilno zanko in pa izhod na koračne motorje sem uporabil napravo Libera LLRF podjetja Instrumentation Technologies d.d. Libera LLRF pretvori visokofrekvenčne analogne signale v digitalne in ima možnost hitrega dostopa iz vgrajenega računalnika do zajetih podatkov preko vmesnika PCI Express.

Ključne besede:

pospeševalniki delcev, krmilnik PID, Instrumentation Technologies d.d.

Abstract

Particle accelerators are used in many different fields. From research in atomic physics, material research to the applications in medical treatments. Even if there are already many different types of particle accelerators new types are still being discovered. One of the new types is a non-scaling FFAG (Fixed Field Alternating Gradient). Because it is relatively small and has good energy efficiency it could increase the usage of particle accelerators in cancer treatment with protons and carbon ions. First prototype named EMMA (Electron Machine for Many Applications) was built in English Daresbury. EMMA is only a first step to confirm the non-scaling FFAG type because it was designed to accelerate electrons and not heavier particles (protons, carbon ions).

This work describes the design, development and application of the system for frequency control of EMMA accelerator radiofrequency cavity. The problem with the radiofrequency cavities is, that because of various factors (change in the environmental temperature, mechanical stress, ...), they are continuously frequency detuning. The consequence of the frequency detuning is that electromagnetic field in the cavity is not optimal for the acceleration of the particles. For this reason the resonant frequency of the cavities must be controlled so that it remains stable and as close to the nominal as possible. Because the above-mentioned factors are slow the problem can be solved with a slow control loop.

For the acquisition of the radiofrequency signals, their analysis, control loop and the controlling of the stepper motors I have used instrument Libera LLRF from the company Instrumentation Technologies d.d.. Libera LLRF converts high frequency analog signals to digital ones and offers a possibility to access them from embedded computer over the PCI Express interface.

Key words:

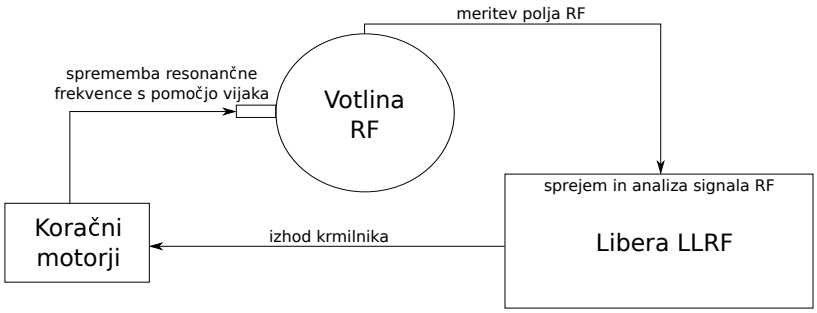
particle accelerators, PID controller, Instrumentation Technologies d.d.

Poglavje 1

Uvod

To diplomsko delo opisuje načrtovanje, razvoj in uporabo sistema za frekvenčno regulacijo radiofrekvenčne votline. Te votline so zelo nestabilne, saj se že z relativno majhnimi spremembami temperature njihova resonančna frekvenca spremeni za več 10 kilohercev. Za optimalno delovanje pospeševalnikov delcev, ki za pospeševanje delcev uporabljajo prav radiofrekvenčne votline, je zato treba regulirati njihovo resonančno frekvenco.

Slika 1.1 prikazuje poenostavljen prikaz sistema za regulacijo skupaj z radiofrekvenčno votlino. Celoten sistem je sestavljen iz treh delov:



Slika 1.1: Poenostavljen prikaz celotnega sistema

- **Libera LLRF**: Naprava podjetja Instrumentation Technologies d.d., ki se uporablja za sprejem radiofrekvenčnega signala, implementacijo krmilnika in krmiljenje koračnih motorjev.
- **Radiofrekvenčna votlina**: Radiofrekvenčna votlina, uporabljena v prototipnem pospeševalniku delcev EMMA. Votlina vsebuje anteno za

merjenje radiofrekvenčnega polja znotraj votline in vijak, s pomočjo katerega uravnavamo resonančno frekvenco.

- **Koračni motorji**: Koračni motorji, uporabljeni za premikanje vijaka, pritrjenega na votlino.

Vidimo lahko, da so za uspešno regulacijo tega sistema potrebni meritev radiofrekvenčnega polja v votlini, sprejem in analiza signala na merilni napravi, krmilni algoritem za nastavljanje primerne resonančne frekvence v votlini in krmiljenje koračnih motorjev. Na grobo se tako programska oprema deli na tri logične dele, ki sovpadajo s strojno opremo:

- **zajem podatkov in analiza**: vsebuje zajem podatkov RF in njihovo analizo za pridobitev resonančne frekvence;
- **krmilnik zanke**: vsebuje opis krmilnika PID za regulacijo votline;
- **izhodni del**: opisuje krmiljenje koračnih motorjev.

Posamezni deli strojne opreme so opisani v drugem poglavju, kjer je prikazan tudi celoten razvojni sistem. Programska oprema je opisana v tretjem poglavju, v četrtem poglavju pa so prikazane meritve in rezultati.

Poglavje 2

Strojna oprema

2.1 Libera LLRF

Slika 2.1: Naprava Libera LLRF

Naprava Libera LLRF podjetja Instrumentation technologies je namenjena amplitudnemu, faznemu in frekvenčnemu krmiljenju polja v radiofrekvenčnih votlinah [1]. Krmiljenje poteka preko nizkonivojskih radiofrekvenčnih signalov. Izhodni signal gre nato preko predojačevalnika in močnostnega ojačevalnika (klystron ali IOT) v votlino, kjer ustvari polje, namenjeno pospeševanju delcev. Zelo pomembno je, da se signal v Liberi LLRF hitro obdela in tako uspešno krmili zanko. Čas signala od vhoda do izhoda je približno 270 ns.

Vsi moduli v napravi so povezani preko zadnje povezovalne plošče (angl. backplane). Libera LLRF je sestavljena iz naslednjih modulov:

- **Osrednja povezovalna plošča** (Inter-Connection Board, v nadaljevanju ICB): Poleg povezovalnih linij vsebuje še naslednje pomembnejše sklope:
  - Modul COMExpress: Na tem modulu so Intelov procesor Core2Duo, pomnilnik in vhodno-izhodne povezave na ICB (PCI Express, USB, I^2C, ...). Na tem modulu teče operacijski sistem GNU/Linux Ubuntu 8.04, na katerem se izvaja večina časovno nekritičnih analiz in visokonivojsko krmiljenje delovanja aplikacije LLRF. COMExpress ima dostop tudi do dveh mrežnih vmesnikov [2].
  - Procesor ARM9: Procesor, ki skrbi za nižjenivojski nadzor naprave Libera LLRF. Med drugim to zajema vklapljanje napajanja drugih modulov, preverjanje temperature, napetosti in toka na vseh modulih in hitrost vrtenja ventilatorjev. Glavna naloga tega procesorja je skrb za zdravje naprave.
  - Stikalo PCI Express: Omogoča dostop modulu COMExpress do drugih delov naprave preko vodila PCI Express. COMExpress je povezan s stikalom PCI Express z vodilom PCI Express s 16 stezami (16 lanes), z drugimi moduli pa je stikalo povezano z 8- (v primeru modula ADC in VM) ali 1- (v primeru modula ICB in TM) steznim vodilom PCI Express.
  - Lattice FPGA: FPGA, ki skrbi za vhodne in izhodne povezave plošče ICB (zunanji priključek USB, vodila UART, LXI, VGA, I^2C in SPI). ARM preko FPGA-ja krmili napajalnike za vse dele naprave.

  Na splošno je torej ICB namenjen krmiljenju in nadzoru zdravja naprave, digitalnim povezavam, prenašanju zajetih surovih in decimiranih podatkov iz drugih modulov in računanju zahtevnih, časovno nekritičnih operacij.

- **Časovni modul** (Timing Modul, v nadaljevanju TM): Skrbi za pravilno ustvarjanje in distribucijo lokalnega oscilatorja (naprej tudi LO), digitalne ure in takt delovanja ali sprožilec (angl. trigger). Kot vhod prejme radiofrekvenčni referenčni signal (naprej tudi MO), s pomočjo katerega potem ustvari LO in digitalno uro. LO je speljan na izhodni konektor in se distribuira zunaj naprave na vhode LO drugih modulov. Digitalna ura

in sprožilni signal se distribuirata preko zadnjega povezovalnega modula. Frekvence posameznih signalov so:

- vhodni signali: Radiofrekvenčni vhodi s spremenljivo frekvenco, ki se giblje v razponu 5,5 MHz okoli centralne frekvence 1300 MHz

$$f_{RF_{min}} = 1296{,}0 \text{ MHz},$$

$$f_{RF_{max}} = 1301{,}5 \text{ MHz};$$

- referenčni signal: Glavni referenčni signal s frekvenco

$$f_{MO} = 1300 \text{ MHz};$$

- lokalni oscilator: Lokalni oscilator poskrbi za pretvorbo vhodnega radiofrekvenčnega signala na vmesno frekvenco (naprej tudi IF)

$$f_{LO} = 1329{,}5454 \text{ MHz};$$

- vmesna frekvenca: Vmesna frekvenca, ki je definirana z enačbo

$$f_{IF} = f_{LO} - f_{RF}; \tag{2.1}$$

  Če je frekvenca vhodnega signala 1300 MHz, je vmesna frekvenca

$$f_{IF} = 1329{,}5454 \text{ MHz} - 1300 \text{ MHz} = 29{,}5454 \text{ MHz}.$$

- digitalna ura: Glavna ura za delovanje analogno-digitalnih in digitalno-analognih pretvornikov in logike v FPGA-jih. Definirana je

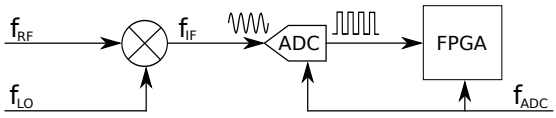
$$f_{ADC} = f_{MO}/n \tag{2.2}$$

$$f_{ADC} = 1300 \text{ MHz} /12 = 108{,}33 \text{ MHz}.$$

- **Sprejemni modul** (v nadaljevanju modul ADC ali ADC9): Sprejemni modul ima 11 radiofrekvenčnih vhodov, razdeljeni so:
  - 8 sprejemnih vhodov RF: Namenjeni so za sprejem radiofrekvenčnega signala od 1296 MHz do 1301,5 MHz
  - 1 referenčni vhod: Sprejem referenčnega signala s frekvenco 1300 MHz. Ko so signali pretvorjeni v digitalne znotraj FPGA-ja, vseh 8 vhodov izenači fazo s tem referenčnim vhodom;

– 1 vhod lokalnega oscilatorja: Sprejem LO, ki je namenjen pretvorbi drugih vhodov na vmesno frekvenco, na kateri potem poteka pretvorba v digitalni signal;
– 1 kalibracijski vhod: Vhod, namenjen kalibraciji radiofrekvenčnih verig znotraj naprave Libera LLRF.

Vseh 8 vhodov za hitro zanko in referenčni vhod se meša (angl. mixing) z vhodom LO na nižjo frekvenco IF. Ti signali se nato pretvorijo iz analognih v digitalne s pomočjo 16-bitnih pretvornikov. Pretvorniki vzorčijo signal z glavno digitalno uro (poimenovano tudi f_{ADC}), ki jo dobijo preko zadnjega povezovalnega modula iz časovnega modula. Slika 2.2 prikazuje poenostavljen sprejem radiofrekvenčnega signala v FPGA. Signali se nato



Slika 2.2: Poenostavljen prikaz sprejema signala

v Xilinx FPGA-ju fazno uskladijo s fazo referenčnega signala in seštejejo v signalni vektor. Signalni vektorji se nato preko linij LVDS prenesejo na vektorski modul. V napravi Libera LLRF so lahko hkrati 4 moduli ADC. Signalni vektorji iz vseh priključenih sprejemnih modulov se nato na vektorskem modulu seštejejo v skupni, globalni vektor (tudi GVS). FPGA na sprejemnem modulu omogoča zajem 8 vhodnih signalov v dveh oblikah:

– surovi podatki: Neobdelani podatki dolžine 262144 vzorcev. Časovno to ustreza

$$t_{raw} = n_{raw}/f_{ADC} \tag{2.3}$$

$$t_{raw} = 262144/108{,}33 \text{ MHz} = 2{,}4 \text{ ms};$$

– decimirani podatki: Podatki, ki so korelirani z referenčnim signalom in decimirani s faktorjem 208. Dolžina enega zajema je dolga 1023 vzorcev. Signal je predstavljen s kompleksno dvojico I in Q ($Z = I + j \cdot Q$), s pomočjo katere s preprosto pretvorbo dobimo amplitudo in fazo

$$amp = \sqrt{I^2 + Q^2}, \tag{2.4}$$

$$ph = \arctan\left(\frac{Q}{I}\right). \tag{2.5}$$

Decimirani podatki zajemajo časovni odsek dolžine

$$t_{dec} = n_{dec}/f_{ADC} \cdot dec \tag{2.6}$$

$$t_{dec} = 1023/108{,}33 \text{ MHz} \cdot 208 = 2 \text{ ms}.$$

Obe vrsti podatkov sta shranjeni na sprejemnem modulu v pomnilniku DDR2 in nato s pomočjo krmilnika DMA preneseni na modul COMExpress.

- **Vektorski modul** (v nadaljevanju VM): Xilinx FPGA na vektorskem modulu sprejme signalne vektorje vseh prisotnih sprejemnih modulov in jih sešteje v globalni signalni vektor. Ta signalni vektor se nato primerja z nastavljeno želeno vrednostjo (angl. set point), razlika ali napaka tega seštevka pa gre v hitri krmilnik PID, realiziran v FPGA-ju. Izhod krmilnika PID gre nato iz FPGA-ja v digitalno-analogni pretvornik in naprej v pretvorbo iz vmesne frekvence na višjo radijsko frekvenco. Digitalna ura in LO, uporabljena pri tem, sta enaka kot na sprejemnih modulih. Tudi vektorski modul ima možnost zajema 8 signalov naenkrat na oba že prej opisana načina.

Pomembno je ločiti med t. i. hitro zanko, v kateri se podatki obdelujejo izključno na nivoju FPGA-ja, in počasno frekvenčno regulacijo votline, katere realizacija je cilj tega diplomskega dela. Pri realizaciji počasne zanke lahko neposredno upoštevamo le glavni povezovalni modul ICB in sprejemni modul ADC, posredno pa seveda tudi časovni modul.

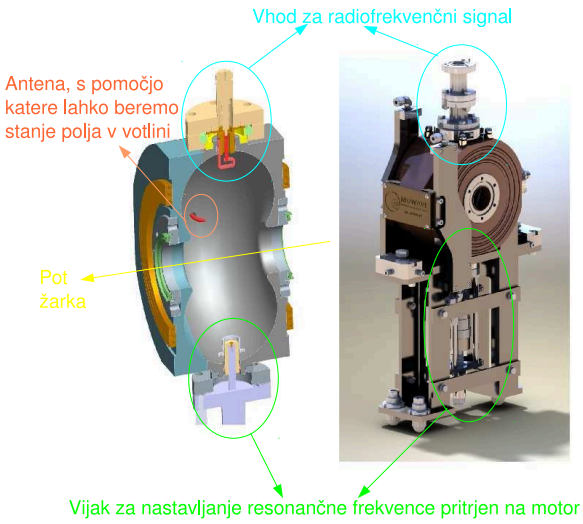
2.2 Radiofrekvenčna votlina

Za izvedbo frekvenčne regulacijske zanke je bila uporabljena normalno prevodna pospeševalniška votlina EMMA. Zasnovana in izdelana je bila za prototipni pospeševalnik tipa neskalirni FFAG, ki ga gradi angleški center za pospeševalniško znanost in tehnologije v Daresburyju [3, 4]. V končanem pospeševalniku bo 19 takih votlin. Slika 2.3 prikazuje prerez in končno inštalacijo votline.

Votlina ima frekvenco 1300 MHz z možnostjo odmika od -4 MHz do +1,5 MHz; izmerjeni faktor kvalitete (faktor Q) ob delovnih pogojih (angl. loaded Q, tudi Q_l) je 10250, kar nam da pasovno širino (s 3-decibelskim pragom) votline

$$BW = f_{RF}/Q_l, \tag{2.7}$$

$$BW = 1300 \text{ MHz } /10250 = 127 \text{ kHz}.$$



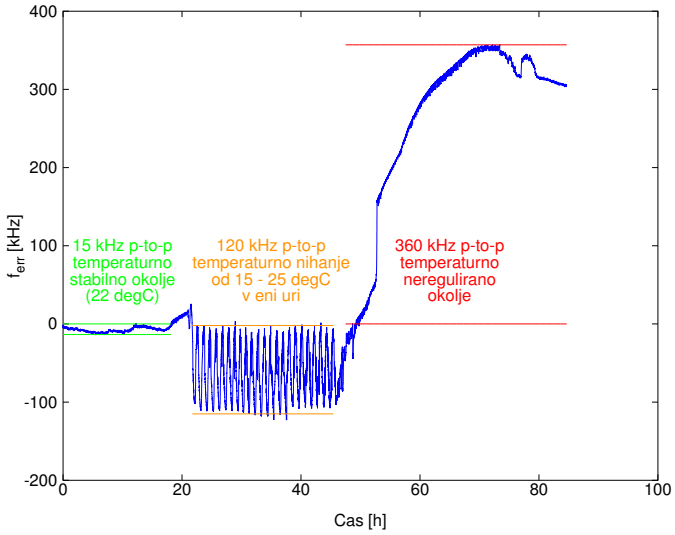
Slika 2.3: Prerez in inštalacija EMMA RF votline.

Maksimalna napetost v votlini je 180 kV.

Kot vidimo iz slike 2.3, ima votlina vhod za radiofrekvenčni signal, ki ga generira Libera LLRF in je nato ojačen. Z anteno znotraj votline lahko izvemo stanje polja v votlini in na podlagi teh podatkov tudi izračunamo trenutno resonančno frekvenco votline. Pri analizi podatkov nam je v veliko pomoč dejstvo, da je radiofrekvenčna votlina harmonični oscilator drugega reda. S pomočjo vijaka (angl. plunger) lahko na podlagi dobljenih podatkov nato spreminjamo geometrijo votline. S tem vplivamo na njeno kapacitivnost in resonančno frekvenco. Zaradi različnih zunanjih dejavnikov, kot so mehanski pritiski in sprememba temperature okolice, votlina neželeno spreminja svojo resonančno frekvenco. Mehanske pritiske na votlino se najlažje blaži z manj stresno okolico (odmik izvorov, kot so generatorji, črpalke in podobno), vzmetenjem, peno in podobnimi postopki. Težje pa je stabilizirati temperaturo okolice, saj sta votlina in distribucija radiofrekvenčnih signalov prevelika za temperaturno stabilne komore. Ker pa je temperaturno nihanje počasno je rešitev v frekvenčni regulaciji votline s pomočjo vijaka.

Slika 2.4 prikazuje meritev resonančne frekvence votline pri meritvi, dolgi 84 ur. Meritev je bila opravljena v temperaturni komori v treh režimih. Prvih približno 20 ur je bila temperatura stabilizirana, nastavljena na 22 stopinj Celzija. Naslednjih 24 ur je komora vsako uro spremenila temperaturo med 15 in 25 stopinjami Celzija. V tretji fazi pa je komora izklopila temperaturno regulacijo. Vidimo lahko, kako hitro prihaja do odstopanja od resonančne

frekvence pri spreminjanju okoliške temperature.



Slika 2.4: 84-urna meritev napake resonančne frekvence votline EMMA.

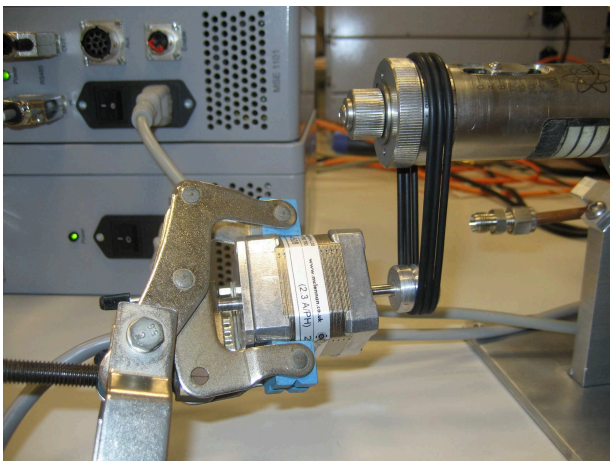
2.3 Koračni motorji

Za krmiljenje resonančne frekvence votline sem uporabil koračne motorje podjetja McLennan. Snovalci pospeševalnika EMMA so se odločili za McLennan, ker so njihovi motorji zelo zanesljivi in, ker podjetje daje dobro podporo za svoje izdelke. Ti motorji imajo RS485-vmesnik [6, 7]. Povezava med Libero LLRF in motorji zato poteka preko pretvornika iz mrežnega v RS485-protokol in gre skozi dodatni razdelilnik, ki en RS485-vhod razdeli v 4 RS485-izhode (slika 2.5). Razdelilnik se uporablja tudi kot krmilnik koračnih motorjev. Dodaten razlog za uporabo razdelilnika pa je tudi dejstvo, da se poleg koračnega motorja za uravnavanje frekvence uporablja še dodaten koračni motor, ki je povezan na fazni sukalnik, vgrajen v distribucijo radiofrekvenčnega signala pred votlino. Tudi pretvornik in razdelilnik sta produkta podjetja McLennan. Uporaba takšne rešitve zelo poenostavi krmiljenje motorjev, saj McLennanovi izdelki uporabljajo t. i. serijski ukazni jezik (SCL), razvit v podjetju Applied Motion [5].

V končni rešitvi bo motor pritrjen na vijak v posebni, za to izdelani inštalaciji (slika 2.3). Za namen razvoja sem motorje pritrdil na vijak votline s

Slika 2.5: Pretvornik in razdelilnik McLennan

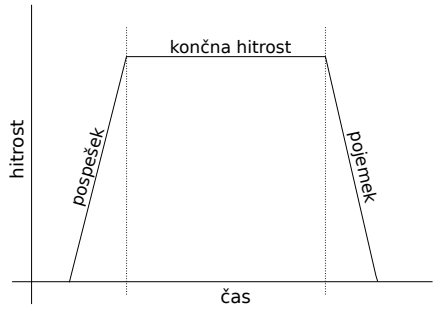
pomočjo treh gumijastih jermenov, sam motor pa je pritrjen ob mizo s splošno namenskim pritrdilnikom. Slika 2.6 prikazuje pritrditev motorja na vijak votline.



Slika 2.6: Pritrditev koračnega motorja na vijak votline

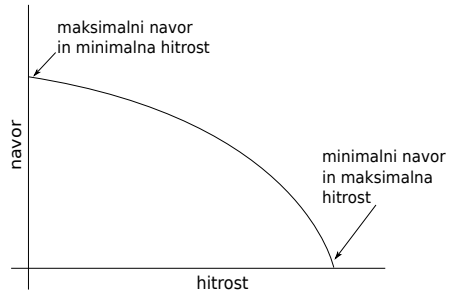
Motorji imajo nastavljiv hitrostni profil, ki je sestavljen iz treh delov (slika 2.7):

- **pospešek**: faktor, s katerim se hitrost veča do končne hitrosti;
- **končna hitrost**: hitrost premikanja motorja, ko se pospeševanje ustavi;
- **pojemek**: faktor, s katerim se hitrost manjša od končne hitrosti do ustavljenega stanja.



Slika 2.7: Hitrostni profil koračnih motorjev

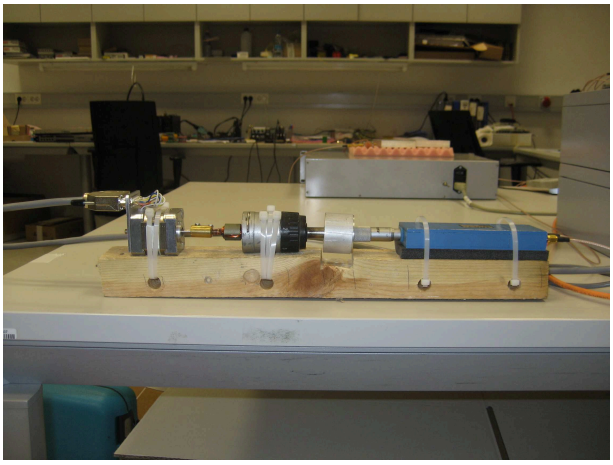
Poleg pravilno izbrane hitrosti je treba paziti še na navor samih motorjev, saj morajo biti ti sposobni obračati breme. Slika 2.8 prikazuje razmerje med navorom in hitrostjo. Želimo doseči čim višjo hitrost, a pri tem imeti navor, ki še vedno lahko obrača breme.



Slika 2.8: Razmerje med navorom in hitrostjo

Pri povezavi motorjev in votline z uporabo neposrednega prenosa (trije jermeni) so imeli motorji dovolj navora, da so lahko premikali vijak. Drugače pa je bilo pri premikanju faznega sukalnika. Motorji niti pri največjem navoru niso mogli premikati faznega sukalnika. V prenos je bil zato vnesen reduktor,

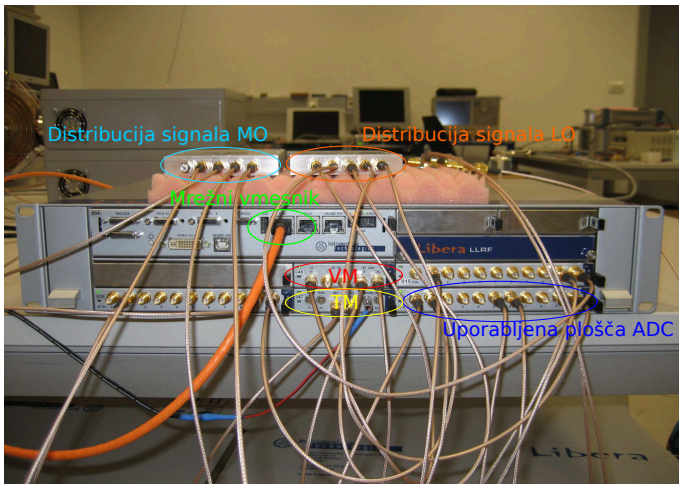
vzet kar iz ročnega električnega vrtalnika (slika 2.9). Ta sprememba prenosa motorjev na fazni sukalnik je omogočila nastavitev najvišje hitrosti in najmanjšega navora. Realizacija premikanja faznega sukalnika ni cilj tega diplomskega dela, podaja pa nam pomembno podrobnost, in sicer, da mora biti rešitev izdelana čim bolj prožno, saj nam obnašanje končnega prenosa moči iz motorjev ni znano. V drugem razdelku poročila bomo videli, da je bila zanka za regulacijo zato izdelana v obliki t. i. krmilnika PID. Ta nam omogoča optimalno delovanje s pravilno nastavitvijo samih parametrov krmiljenja. Programabilno mora biti tudi nastavljanje hitrostnega profila vsakega posameznega motorja.



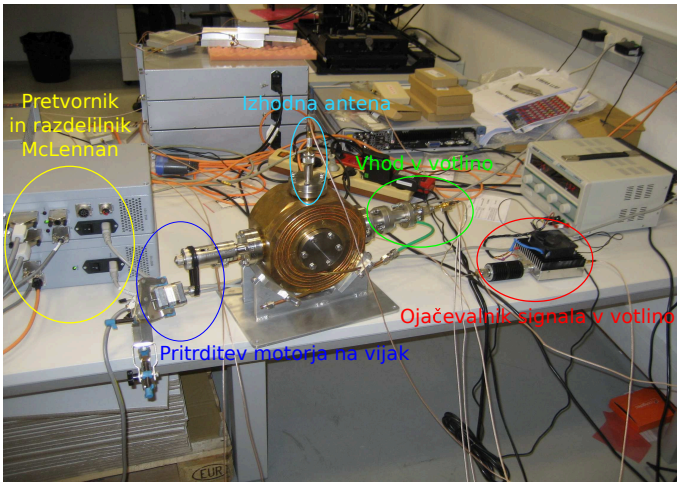
Slika 2.9: Pritrditev koračnega motorja na fazni sukalnik

2.4 Celoten razvojni sistem

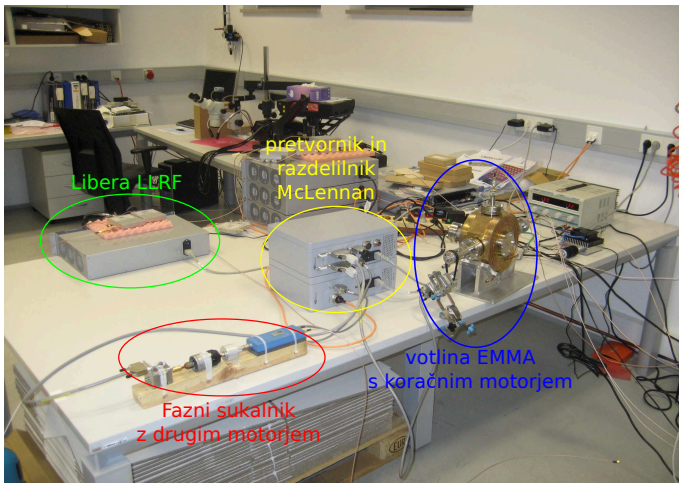
Slike od 2.10 do 2.13 prikazujejo končno priključitev celotnega sistema.



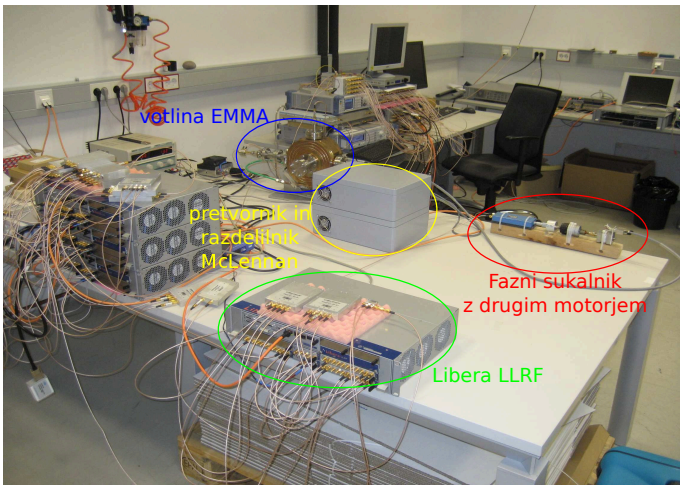
Slika 2.10: Priključitev naprave Libera LLRF



Slika 2.11: Priključitev votline EMMA



Slika 2.12: Celoten razvojni sistem 1

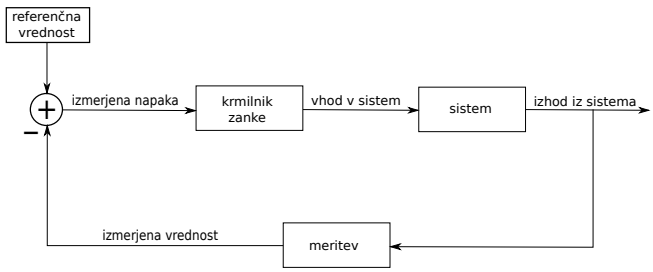


Slika 2.13: Celoten razvojni sistem 2

Poglavje 3

Programska oprema

Krmilna zanka je implementirana v programskem jeziku C++, ki teče na operacijskem sistemu GNU/Linux Ubuntu 8.04 na modulu COM Express. Slika 3.1 prikazuje tipično krmilno zanko, ki vključuje zajem podatkov (ali meritev), in jo primerja z nastavljeno referenčno vrednostjo. Razliko (tudi napako) teh dveh vrednosti obdela krmilnik in jo poizkusi krmiliti s pomočjo izhoda v sistem.



Slika 3.1: Tipična krmilna zanka

S pomočjo slike 3.1 lahko razdelimo implementacijo krmilne zanke v naslednje sklope:

- **Zajem podatkov in analiza**: Del, v katerem se zajamejo surovi podatki iz sprejemne plošče in se opravi analiza signala, da se pridobi resonančna frekvenca votline. Ta korak po potrebi skrbi tudi za povprečenje več zaporednih meritev.

- **Primerjava meritve z referenčno vrednostjo**: Izračuna razliko med želeno vrednostjo in dejansko vrednostjo resonančne frekvence v votlini.

- **Krmilnik zanke**: Krmilnik zanke je implementiran kot krmilnik PID. Skrbi za pravilno krmiljenje resonančne frekvence v votlini in hkrati tudi beleži zgodovino prejšnjih stanj, ki so uporabljena za izračun dolgoročnejše napake.

- **Izhod zanke**: Krmilnik koračnih motorjev preko mrežnega vmesnika.

Sistem na sliki 3.1 poleg same votline vključuje tudi odziv koračnih motorjev, prenos iz motorja na vijak in distribucijo celotnega signala, skupaj z obdelavo signala.

Kljub temu da je cilj diplomskega dela izdelava frekvenčnega krmiljenja ene votline, je sama rešitev splošna. Zato imamo še nekaj dodatnih pogojev, ki jih moramo upoštevati pri načrtovanju rešitve. Rešitev mora podpirati:

- **več vzporednih enakih zank**; podpirati mora na primer frekvenčno krmiljenje več votlin naenkrat;

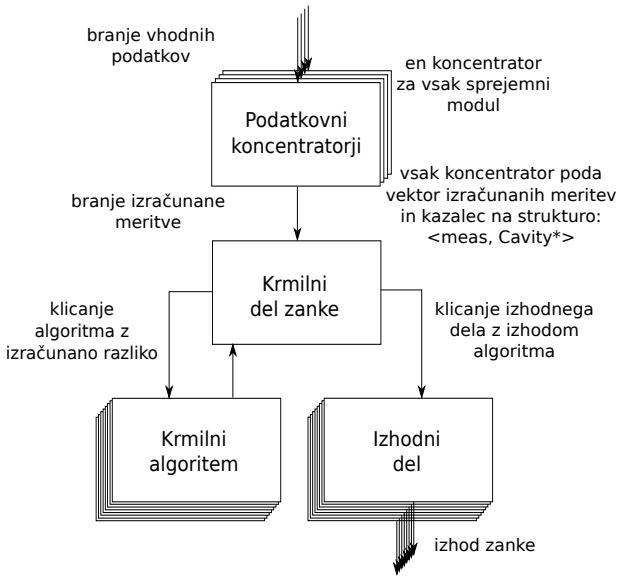
- **druge tipe zank**; podpora za druge tipe zank, pri čemer je mogoče zamenjati vsakega izmed delov krmilne zanke (vhod, izračun napake, krmilni del, izhod). Pri tem lahko privzamemo, da ima vsak tip zanke enake pogoje za delovanje (sistem, izhod, ...);

- **dostop do podatkov več signalov naenkrat**; pri tem gre predvsem za podporo t. i. referenčne zanke. Na podlagi le-te druge vzporedne zanke izračunavajo svojo napako. Meritev torej ni absolutna glede na sistem, ampak relativna glede na druge zanke;

- **nastavitve vsake posamezne zanke**; s tem lahko vsaki zanki nastavimo drugačno referenčno vrednost ali jo vklopimo/izklopimo.

Na podlagi zgoraj določenih sklopov in dodatnih pogojev izberemo primerno arhitekturo rešitve. Diagram na sliki 3.2 prikazuje predlagano rešitev.

Podatkovni koncentrator je razred, ki skrbi za zajem podatka, primernega za izračun meritve in povprečenje ali decimacijo, če je ta potrebna. Koncentrator iz nastavitev v strukturi *Cavity* izve, kateri vhodi so vklopljeni, in naredi izračun le zanje. V sistemu je za vsak sprejemni modul instanciran en podatkovni koncentrator. Vsi instancirani koncentratorji nato posredujejo izračune krmilnemu delu. Branje podatkov iz koncentratorja je blokirano, dokler podatki niso na voljo. Krmilni del od vsakega koncentratorja sprejme vektor meritev in kazalcev na strukturo *Cavity*. Če je ta neveljavna (ima rezervirano vrednost *NULL*), zanj meritev ni veljavna in zanke za ta signal ne izvaja. Sicer pa iz pridobljenih podatkov izračuna napako za vsak posamezen signal (izvede



Slika 3.2: Implementacija krmilne zanke

lahko absolutni ali relativni izračun, saj ima podatke za vse signale) in pokliče krmilni algoritem. Izhod algoritma pošlje na izhodni del. Izhodni del ni nujno samo izhod na koračne motorje, lahko je na primer RS232-izhod, vhod v naslednji algoritem ali pa vhod v naslednjo krmilno zanko (pri ugnezdenih krmilnih zankah). V naslednjih poglavjih so podrobneje razloženi posamezni bloki.

3.1 Vhodni del in analiza signala

To poglavje zajema podroben opis podatkovnega koncentratorja, implementiranega za izračun resonančne frekvence. Njegova naloga je, da sprejme signal, zajet v FPGA-ju, iz njega izračuna resonančno frekvenco, rezultat povpreči in skrbi za posredovanje vektorja rezultatov krmilnemu delu.

Dostop do podatkov

V operacijskem sistemu znotraj naprave Libera LLRF so naloženi sistemski gonilniki, ki omogočajo dostop preko vodila PCI Express do FPGA-jev. Preko teh gonilnikov je programska oprema obveščena o novih dogodkih (angl. events), ima dostop do registrov FPGA-jev in lahko bere podatke iz posame-

znih sprejemnih modulov. Na napravi Libera LLRF teče strežnik, ki poskrbi za branje signalov in omogoča priklop zunanjih signalnih sprejemnikov. Strežnik je implementiran s pomočjo tehnologije CORBA, kar nam omogoča lažji razvoj odjemalskega signalnega sprejemnika.

Razred za podatkovni koncentrator za izračun resonančne frekvence je poimenovan *ResonanceConcentrator*:

```
class ResonanceConcentrator :
                        public POA_i_MCI::SignalReceiver {
 public:
  // Constructors, destructors
  ResonanceConcentrator(BoardCavities a_cavities);
  virtual ~ResonanceConcentrator();

  // ResonanceConcentrator interface methods
  void Open();
  void Reset();
  void Close();
  void Update();
  const ResonanceVect Read();
  const ResonanceSet  ReadRef();
  // End of ResonanceConcentrator interface methods

  // SignalReceiver interface methods
  ::CORBA::Boolean IsReady() { return m_isReady; };

  virtual void SendRaw(
               const i_MCI::SignalAttributes& a_attr,
               const i_MCI::RawSignal& a_signal);
  // End of SignalReceiver interface methods

 private:
  std::list<ResonanceVect>  m_resonanceFifo;
  ResonanceVect             m_currentResonance;

  ssize_t m_loopCnt;
};
```

Kot vidimo, je koncentrator dedič razreda *POA_i_MCI::SignalReceiver*, ki nam omogoča preprost priklop na signalni strežnik. Delovanje je zelo preprosto, saj ob registriranju našega signalnega sprejemnika določimo, kakšen tip signala hočemo sprejemati. V tem primeru je to tip surovih podatkov.

Vsakič, ko je signal na voljo, se pokliče metoda *SendRaw()*, ki ima kot vhodne argumente vzorce signala in atribute signala. S članom *m_ isReady* določamo, ali naj nam strežnik pošlje nov signal ali ne.

Metode *Open(), Reset(), Close(), Update(), Read()* in *ReadRef()* so vmesniki pred krmilnim delom, s pomočjo katerih krmili delovanje koncentratorja in bere izračunane rezultate.

Član *m_ resonanceFifo* je namenjen zbiranju izračunanih meritev skupaj s pripadajočimi kazalci na strukturo *Cavity*. Ob klicu *Read()* se bo v krmilni del vedno prenesla vrednost, ki je najdlje na seznamu. *m_ currentResonance* in *m_ loopCnt* sta namenjena internemu povprečenju vsakega posameznega rezultata. Meritev surovega signala se opravi s taktom 10 hercev, tako da koncentrator vsakih 100 milisekund dobi nov signal za izračun resonančne frekvence. Za povprečenje se uporablja 30 zaporednih meritev. To pomeni, da se algoritem za krmiljenje zanke pokliče enkrat na vsake 3 sekunde.

Izračun resonančne frekvence

Radiofrekvenčna votlina deluje kot zunanje vzbujani harmonični oscilator (angl. externally driven harmonic oscillator). To pomeni, da pri neki naravni ali t. i. resonančni frekvenci oscilira. Takšne oscilatorje lahko na splošno opišemo z nehomogenimi linearnimi diferencialnimi enačbami drugega reda [8]

$$\frac{d^2x}{dt^2} + 2\zeta\omega_0\frac{dx}{dt} + \omega_0^2 x = A_o \sin\omega t, \tag{3.1}$$

kjer je ζ količnik dušenja, ko oscilatorja ne vzbujamo več z zunanjim signalom, ω_0 pa nedušena frekvenca. Desna stran nam opiše signal, s katerim vzbujamo oscilator - A_o je amplituda signala, ω pa frekvenca signala.

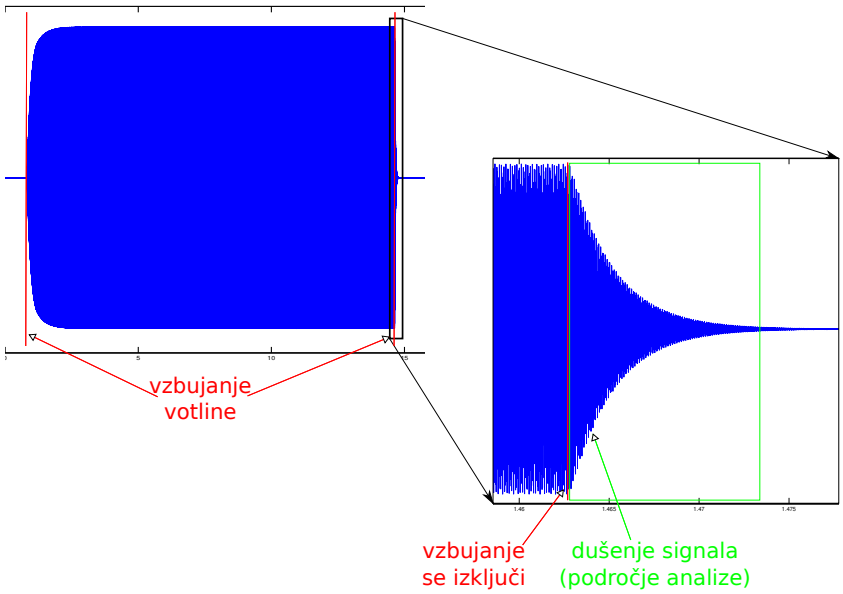
Zanimata nas predvsem resonančna frekvenca votline in Q-faktor, ki je povezan s količnikom dušenja preko enačbe [9]

$$\zeta = \frac{1}{2 \cdot Q} = \frac{\alpha}{\omega_0}, \tag{3.2}$$

kjer α predstavlja dušenje (angl. attenuation) signala. Frekvenco dobimo s pomočjo enačbe

$$f_{res} = Q \cdot \frac{\alpha}{\pi}. \tag{3.3}$$

Kot vidimo, moramo iz enačbe 3.1 pridobiti α in ω_0, zato ζ nadomestimo z zadnjim delom enačbe 3.2. Analiza se bo računala nad vzorci signala, kjer



Slika 3.3: Področje analize signala

votline ne vzbujamo več z radiofrekvenčnim signalom (slika 3.3), zato se lahko znebimo še desnega dela enačbe 3.1; tako dobimo

$$\frac{d^2x}{dt^2} + 2\alpha\frac{dx}{dt} + \omega_0^2 x = 0. \tag{3.4}$$

Časovno dušenje signala opišemo z enačbo

$$y(t) = e^{-\alpha t} \cdot \cos(\omega_0 t), \tag{3.5}$$

ker analiziramo vzorčeni signal, enačbo napišemo v diskretni obliki

$$y[n] = e^{-\alpha n} \cdot \cos(\Omega_o n). \tag{3.6}$$

Diferencialno enačbo 3.4 rešimo s pomočjo matrik. Definiramo vektor y in matriko A

$$y[n] = \begin{bmatrix} x[n] \\ x[n+1] \end{bmatrix}, \tag{3.7}$$

$$A[n] = \begin{bmatrix} x[n-1] & x[n-2] \\ x[n] & x[n-1] \end{bmatrix}, \tag{3.8}$$

kjer je x vhodni vzorčeni signal. Inverz matrike A je

$$A^{-1}[n] = 1/det[n] \cdot \begin{bmatrix} x[n-1] & -x[n-2] \\ -x[n] & x[n-1] \end{bmatrix}, \tag{3.9}$$

$$det[n] = x[n-1] \cdot x[n-1] - x[n-2] \cdot x[n]. \tag{3.10}$$

Izračunamo vektor k, ki je rešitev diferencialne enačbe

$$k[n] = A^{-1}[n] \times y[n], \tag{3.11}$$

in izračunamo α in Ω_0 po formulah

$$\alpha[n] = -\frac{\log(-k[n][2])}{2} \cdot f_{ADC} \tag{3.12}$$

$$\Omega_0[n] = \left| \arccos \left[\frac{k[n][1]}{-2 \cdot e^{\frac{-\alpha[n]}{f_{ADC}}}} \right] \cdot f_{ADC} \right|. \tag{3.13}$$

Izračunali smo vektorja α in Ω_o, ki pa se skozi časovno področje analize ne spreminjata veliko. Pri idealnem signalu po enačbi 3.6 se ne spreminjata, v realnosti pa pride do majhnih odstopanj zaradi šuma v signalu. Dokončno α in Ω dobimo tako, da povprečimo vse pridobljene rezultate. S tem smo povečali natančnost analize, saj smo se znebili odstopanj, do katerih pride zaradi naključnega šuma. S pomočjo enačb 3.2 in 3.3 tako pridemo do Q-faktorja in resonančne frekvence. Zavedati se moramo, da sta izračunani vrednosti dobljeni na signalu, ki je bil vzorčen na vmesni frekvenci. Treba je torej preračunati vrednost z vmesne frekvence na radiofrekvenčno frekvenco

$$f_{res_{RF}} = f_{LO} - f_{res_{IF}}. \tag{3.14}$$

Metoda za izračun frekvence je definirana kot:

```
int ComputeDecay(i_MCI::DecayData &a_dData,
                 const i_MCI::RawSignal &a_data,
                 const DecayParams &a_params);
```

i_MCI::DecayData a_dData je izhodna struktura, v kateri sta izračunana Q-faktor in resonančna frekvenca. *i_MCI::RawSignal a_data* je vzorčeni signal, nad katerim se izvede analiza, *DecayParams a_params* pa je struktura, ki vsebuje parametre, ki so potrebni pri izračunu resonančne frekvence. Definirana je:

```
typedef struct {
    unsigned int offset;
    unsigned int size;
    unsigned int index;
    unsigned int rows;
    double       adcFreq;
    double       loFreq;
} DecayParams;
```

Posamezni parametri so:

- ***offset***: odmik od prvega vzorca v vhodnem signalu. Pri tem odmiku se bo začela analiza;
- ***size***: število vzorcev, ki jih bo upoštevala analiza;
- ***index***: številka kanala, nad katerim želimo izvesti analizo;
- ***rows***: število vseh kanalov v vhodnem signalu;
- ***adcFreq***: frekvenca vzorčenja, f_{ADC};
- ***loFreq***: frekvenca lokalnega oscilatorja f_{LO}, ki je potrebna za preračun rezultata na pravo frekvenco z vmesne frekvence.

3.2 Krmilni del in krmilni algoritem

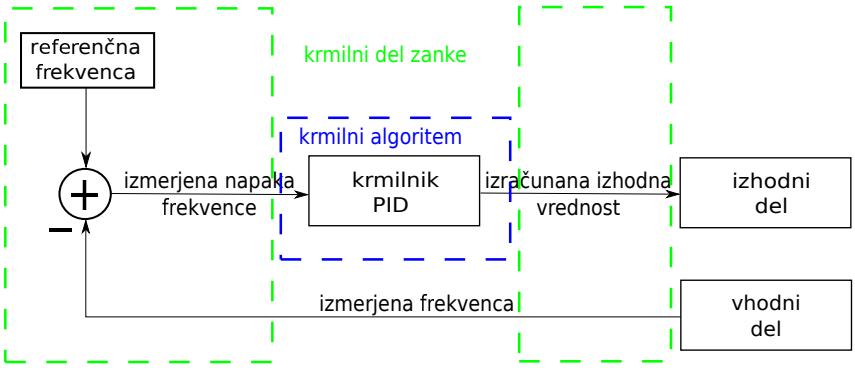
Kot vidimo na sliki 3.2, je krmilni del sestavljen iz dveh delov:

- **krmilni del zanke** skrbi za sprejemanje podatkov in izračun napake med izmerjeno in nastavljeno vrednostjo;
- **krmilni algoritem** skrbi za računanje izhoda na podlagi izračunane napake.

Slika 3.4 prikazuje realizacijo zanke za krmiljenje resonančne frekvence votline, skupaj z izračunom napake in krmilnikom PID.

Krmilni del zanke

Kot vidimo na sliki 3.4, je krmilni del zanke povezovalec vhodnega dela, krmilnega algoritma in izhodnega dela. Za vsak tip krmilne zanke vedno obstaja le ena instanca krmilnega dela, ki na operacijskem sistemu teče kot samostojna nit. Razred *CavityTuning* je zgrajen:



Slika 3.4: Regulacija PID

```
/**
* Cavity resonant frequency control loop.
*/
class CavityTuning : public omni_thread
{
  public:
    CavityTuning(ResonanceConcentratorVect a_rConcentrators);
    ~CavityTuning();

    void Handle();
    virtual bool Pause();
    virtual bool Resume();
    virtual void Stop();
    /* a_time in nanoseconds */
    virtual void Sleep(long a_time);

  private:
    virtual void* run_undetached(void* a_arg);

    ResonanceConcentratorVect m_rConcentrators;

    bool m_pause;
    bool m_stop;

    omni_mutex     m_slMutex;
    omni_condition m_slCond;
};
```

CavityTuning je dedič razreda *omni_thread*, ki je del sistema CORBA in skrbi za izvrševanje razreda v svoji niti. Metode *Pause()*, *Resume()*, *Stop()* in *Sleep()* skrbijo za samo izvajanje niti. Prav tako sta člana *m_pause* in *m_stop* namenjena za krmiljenje izvajanja. Člana *m_slMutex* in *m_slCond* skrbita, da ob prehodih stanj v izvajanju ne prihaja do smrtnega objema (angl. dead lock) in sta prav tako kot *omni_thread* del sistema CORBA. Član *m_rConcentrators* vsebuje vse prisotne podatkovne koncentratorje in se napolni ob konstruiranju razreda.

Med delovanjem se krmilna zanka ves čas vrti le v metodi *Handle()*, kjer opravlja naslednje naloge:

- **sprejem podatkov**: zahteva podatke od vseh koncentratorjev. Če podatki niso na voljo, koncentrator blokira branje podatkov, dokler le-ti niso na voljo;

- **izračun napake**: po sprejemu izmerjenih frekvenc preveri, če je sprejeti podatek veljaven. Če je veljaven, iz strukture *Cavity* prebere nastavljeno frekvenco in izračuna napako:

```
// Get reference value from configuration
double refValue = data.cavity
   ->GetTuningLoopConf(i_MCI::eFrequencyTuning)
   ->refValue;

double freqErr = refValue - data.resFreq;
```

- **klic krmilnega algoritma**: pokliče korak v krmilnem algoritmu. Kot vhod je podana napaka meritve, izhod pa je nova vrednost premika motorja:

```
// Call control algorithm
double freqOut =
       data.cavity
         ->GetTuningLoop(i_MCI::eFrequencyTuning)
         ->Step(freqErr)
```

- **oddaja izhodnih podatkov**: na koncu pokliče še izhodne motorje in jim kot vhod poda izračunano vrednost premika motorja:

```
// Move the output motor
data.cavity
  ->MoveTuningLoopMotor(i_MCI::eFrequencyTuning,
                        freqOut);
```

V zgoraj naštetih točkah lahko vidimo, da krmilni del vse podatke pridobiva iz strukture *Cavity*, ki mu jo posreduje podatkovni koncentrator skupaj z izmerjeno frekvenco. Ta struktura vsebuje podatke o nastavitvah posameznih zank, posameznih vhodnih signalih in kazalce na krmilne algoritme ter motorje. Krmilni del tako le povezuje že zgoraj naštete dele. Razred *Cavity* vsebuje naslednje elemente:

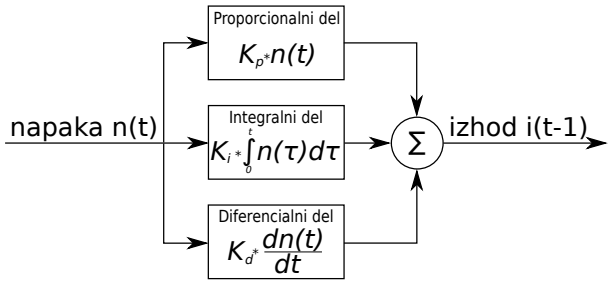
```
class Cavity {
  public:
    TuningLoopParam
            *GetTuningLoopConf(i_MCI::TuningType a_type);
    PIDController
            *GetTuningLoop(i_MCI::TuningType a_type);
    void MoveTuningLoopMotor(const i_MCI::TuningType a_type,
                             const ssize_t a_distance);

  private:
    // slow loop PIDs
    PIDController *m_freqTuning;
    PIDController *m_phaseTuning;

    // tuning motors - there are two motors,
    // one for Freq loop and the one for Phase loop
    McLennanMotor *m_freqOutMotor;
    McLennanMotor *m_phaseOutMotor;
};
```

Krmilni algoritem

Kot je bilo že omenjeno, je bil za krmilni algoritem izbran krmilnik PID. Glavni razlogi za tako izbiro so:

- **nastavljivost samega krmilnika**: zaradi neznanega prenosa moči z motorjev na sam vijak. S tem ohranimo prožnost implementacije, saj lahko zelo preprosto prilagodimo krmilnik pri končni inštalaciji;
- **splošna razširjenost**: krmilniki PID se uporabljajo na zelo različnih področjih (od počasnih regulacij, na primer temperaturna regulacija prostorov, do hitrih regulacij, kot je protizdrsna zaščita v avtomobilih). Zaradi široke razširjenosti je na to temo napisanih veliko knjig in člankov.

Proporcionalno-integralno-diferencialni (PID) krmilnik je, kot prikazuje slika 3.5, sestavljen iz treh glavnih gradnikov [10]:



Slika 3.5: Notranja zgradba krmilnika PID

- **Proporcionalni del**: vpliva na izhod tako, da množi nastavljeni proporcionalni člen z vhodno vrednostjo napake; opišemo ga z enačbo

$$i(t) = K_p \cdot n(t). \tag{3.15}$$

Čim višje je nastavljen proporcionalni faktor, hitreje zanka vpliva na spremembo na izračunani napaki. Pri previsoko nastavljenem členu pa zanka lahko postane tudi nestabilna. To pomeni, da zanka nikoli ne skonvergira k želeni vrednosti, ampak niha okoli nje ali pa celo divergira. Če je v krmilnem algoritmu uporabljen izključno proporcionalni del, pride do t. i. statične napake, kar pomeni, da zanka nikoli ne doseže želene vrednosti. Levi graf na sliki 3.6 prikazuje vpliv višanja proporcionalnega člena na odziv sistema. Vidimo lahko, da zanka z višjim proporcionalnim členom veliko hitreje skonvergira. Če bi ga še višali, bi sistem začel oscilirati in nato divergirati od nastavljene vrednosti.

- **Integralni del**: vpliva na izhod, tako da množi seštevek (integral) napake z integralnim faktorjem. Na izhod integralnega dela tako ne vpliva le trenutna vrednost napake, ampak tudi napake iz preteklosti. Opišemo ga z enačbo

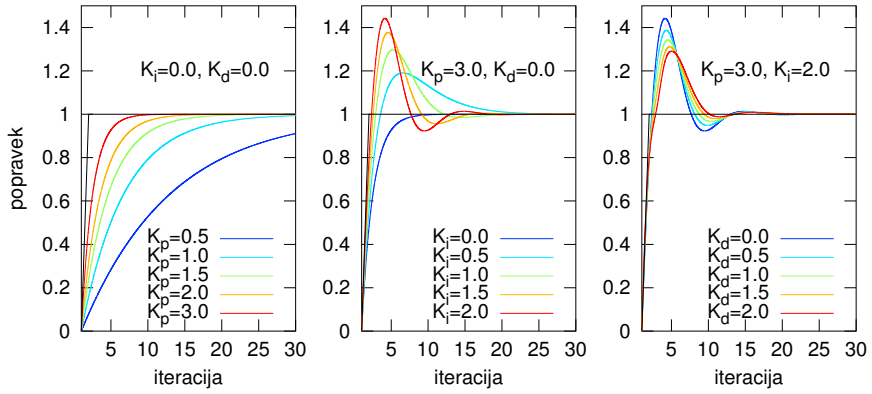
$$i(t) = K_i \cdot \int_0^t n(\tau)d\tau. \tag{3.16}$$

Integralni del popravi statično napako proporcionalnega člena in pri hitrih, nepredvidenih spremembah v sistemu naredi sistem stabilnejši. Pri večjih vrednostih integralnega člena lahko pride do večjih prekoračitev (angl. overshoot) nastavljenih vrednosti, saj na izhod vpliva tudi seštevek napak iz preteklosti. Sredinski graf na sliki 3.6 prikazuje odziv sistema z različnimi vrednostmi integralnega parametra. Vidimo lahko, da z višjimi vrednostmi zanka zelo hitro skonvergira k nastavljeni vrednosti, a pri tem prihaja do vedno višjih prekoračitev nastavljene vrednosti.

- **Diferencialni del**: vpliva na izhod, tako da gleda spremembo v napaki skozi čas (odvod napake po času) in to zmnoži z diferencialnim faktorjem

$$i(t) = K_d \cdot \frac{dn(t)}{dt}. \tag{3.17}$$

Z omejitvijo spremembe napake skozi čas lahko omejimo prevelike prekoračitve nastavljenih vrednosti in dosežemo večjo stabilnost krmiljenja sistema. Desni graf na sliki 3.6 prikazuje odziv sistema na različne nastavitve diferencialnega dela. Vidimo lahko, da se prekoračitev zaradi integralnega dela zmanjšuje z višanjem diferencialnega dela, a tudi sistem počasneje doseže nastavljeno vrednost.



Slika 3.6: Različne nastavitve za K_p, K_i in K_d

Slika 3.6 prikazuje vpliv posameznih parametrov na krmilnik PID na sistemu z idealno prenosno funkcijo. Kot bomo videli v nadaljevanju, so v resničnem sistemu vrednosti parametrov drugačne zaradi vpliva sistema na zanko.

Krmilnik PID na sliki 3.5 je paralelnega tipa. Iz enačb 3.15, 3.16 in 3.17 sestavimo enačbo in jo zapišemo v diskretni obliki

$$i[t_1] = K_p \cdot n[t_1] + K_i \cdot \sum_{k=0}^{t_1} n[k] + K_d \cdot (n[t_1] - n[t_1 - 1]). \tag{3.18}$$

Za digitalno implementacijo nas zanima Z-transformacija enačbe 3.18

$$I[z] = \left[K_p + \frac{K_i}{1 - z^{-1}} + K_d \cdot (1 - z^{-1})\right] \cdot N[z], \tag{3.19}$$

če jo preuredimo, dobimo

$$\frac{I[z]}{N[z]} = \left[\frac{(K_p + K_i + K_d) + (-K_p - 2 \cdot K_d) \cdot z^{-1} + K_d \cdot z^{-2}}{1 - z^{-1}}\right]. \tag{3.20}$$

Definiramo nove spremenljivke

$$K_1 = K_p + K_i + K_d, \tag{3.21}$$

$$K_2 = -K_p - 2 \cdot K_d, \tag{3.22}$$

$$K_3 = K_d. \tag{3.23}$$

Vstavimo 3.21, 3.22 in 3.23 v enačbo 3.20 in jo preuredimo v

$$I[z] = z^{-1} \cdot I[z] + K_1 \cdot N[z] + K_2 \cdot z^{-1} \cdot N[z] + K_3 \cdot z^{-2} \cdot N[z]. \tag{3.24}$$

Če jo transformiramo v časovni prostor, dobimo končno enačbo, ki jo uporabimo za implementacijo

$$i[t_1] = i[t_1 - 1] + K_1 \cdot n[t_1] + K_2 \cdot n[t_1 - 1] + K_3 \cdot n[t_1 - 2]. \tag{3.25}$$

Z nastavljanjem zgoraj naštetih parametrov K_p, K_i in K_d določimo delovanje zanke in lahko dosežemo optimalno krmiljenje. Optimalno delovanje zanke pomeni tako izbiro parametrov, da je zanka stabilna (da vedno konvergira proti nastavljeni vrednosti), da čim manj prekorači nastavljeno vrednost in da se čim hitreje odziva na spremembe v sistemu. Nastavljanje zanke se v praksi vedno izkaže za zelo težavno nalogo. Za določanje optimalnih parametrov zanke je možnih več pristopov:

- **Ročno določanje parametrov**: je najočitnejši način za določitev parametrov PID. S postopnim spreminjanjem parametrov lahko opazujemo sklenjeno delujočo zanko in dosežemo želeno stabilnost. Največja pomanjkljivost tega pristopa je, da je časovno potraten. Pri neizkušenem operaterju obstaja tudi nevarnost, da zanka začne divergirati.

- **Ziegler-Nicholsova metoda**: je ena bolj razširjenih hevrističnih metod za določitev parametrov. Sprva postavimo K_i in K_d na nič. K_p povečujemo, dokler ne zaznamo oscilacij v zanki (tej točki se reče končno ojačenje, angl. ultimate gain - K_u). Ko je sistem v stanju osciliranja, izmerimo končno periodo oscilacije, P_u. Parametrom se nato nastavijo vrednosti

$$K_p = 0{,}6 \cdot K_u,$$
$$K_i = \frac{2 \cdot K_p}{P_u},$$
$$K_d = \frac{K_p \cdot P_u}{8}.$$

Največja slabost tega načina določanja parametrov je, da se sklenjeno zanko nastavi na mejno delovanje (oscilacije) v delujočem sistemu. Ker ima pospeševalniška votlina med delovanjem zelo visoko polje (do 180 kV), tak pristop ni priporočljiv.

- **Tyreus-Luybenova metoda**: je zelo podobna Ziegler-Nicholsovi metodi. Postopek je popolnoma enak, enačbe za določitev parametrov pa so

$$K_p = \frac{K_u}{2{,}2},$$
$$K_i = 2{,}2 \cdot P_u,$$
$$K_d = \frac{P_u}{6{,}3}.$$

Tudi slabosti so enake kot pri Ziegler-Nicholsovi metodi.

- **Cohen-Coonova metoda**: je matematična metoda, ki temelji na modelu prvega reda. Parametre se določi na sistemu v odprti zanki. Postopek je naslednji:
  - izvedemo test z enotino stopnico, da pridobimo parametre za model:
    - počakamo, da sistem preide v stanje mirovanja (je stabilno - to stanje imenujemo A);
    - spodbudimo sistem z enotino stopnico;
    - počakamo, da se sistem stabilizira v novem stanju (imenovanem B) in izmerimo t_0 (začetni čas, ko smo sistem spodbudili z enotino stopnico), t_2 (čas, ko izmerimo polovični nivo končnega stanja - $0,5 \cdot B$) in t_3 (čas, ko izmerimo 0,632-kratnik končnega signala - $0,632 \cdot B$);
  - izračunamo procesne parametre

$$t_1 = \frac{t_2 - (ln(2)) \cdot t_3}{1 - ln(2)},$$
$$\tau = t_3 - t_1,$$
$$\tau_{del} = t_1 - t_0,$$
$$K = \frac{B}{A},$$
$$r = \frac{\tau_{del}}{\tau};$$

– glede na izračunane procesne parametre določimo K_p, K_i in K_p z enačbami

$$K_p = \frac{1}{r \cdot K} \cdot (\frac{4}{3} + \frac{r}{4}),$$
$$K_i = \tau_{del} \cdot \frac{32+6 \cdot r}{13+8 \cdot r},$$
$$K_d = \tau_{del} \cdot \frac{4}{11+2 \cdot r}.$$

Kot vidimo, je ta metoda relativno kompleksna.

Kot že omenjeno, se prenosna karakteristika sistema (komunikacija s koračnimi motorji, izhod iz koračnih motorjev na vijak votline in odziv same votline) skozi čas ne spreminja. Parametre je zato treba določiti le ob postavitvi sistema.

To je razlog, da sem se odločil za najpreprostejšo rešitev, to je ročno določanje najoptimalnejših parametrov. Da to ne bi bilo preveč časovno potratno, sem za izhodiščne vrednosti parametrov uporabil krajšo in poenostavljeno simulacijo sistema, ki na grobo poišče primerne vrednosti. Natančnejša določitev optimalnih vrednosti se nato na sistemu izvede ročno. Delovanje in določanje parametrov je podrobneje opisano v poglavju 4.2.

PIDController je razred, kjer je implementiran krmilni algoritem:

```
class PIDController
{
public:
    PIDController();
    virtual ~PIDController();

    void SetGains(double a_Kp, double a_Ki, double a_Kd);
    void GetGains(double& a_Kp, double& a_Ki,
                  double& a_Kd) const;
    void SetClip(double a_low, double a_high);
    void GetClip(double& a_low, double& a_high) const;
    void SetHysteresis(double a_hyst);
    double GetHysteresis() const;
    void SetDeadZone(double a_dead);
    double GetDeadZone() const;

    double Step(double a_u);

private:
    double m_Kp;
    double m_Ki;
```

```
    double m_Kd;

    limit_t m_clip;
    double m_hyst;
    double m_dead;
    double m_last;

    omni_mutex m_params_x;
};
```

Metodi *SetGains()* in *GetGains()* skrbita za nastavljanje oz. branje vrednosti parametrov K_p, K_i in K_d. Ob naslednjem klicu metode *Step()* se bodo novi parametri uporabili pri izračunu nove izhodne vrednosti. Poleg implementacije same enačbe 3.25 implementacija metode *Step()* vsebuje še nekaj dodatnih parametrov za stabilnejše delovanje zanke:

- ***izrez*** (angl. clip): z metodo *SetClip()* se nastavita najnižja in najvišja izhodna vrednost krmilnika. Če bi krmilnik PID na izhodu zahteval nižjo ali višjo vrednost od parametrov *a_low* in *a_high*, bo izhod omejen na to vrednost. S tem krmilnik ne poizkuša delati predolgih sprememb naenkrat;

- ***histereza*** (angl. hysteresis): z metodo *SetHysteresis()* se nastavi vrednost histereze. Če bo izhod krmilnika PID trenutne iteracije v območju nastavljene histereze, se bo na izhod poslala vrednost, enaka prejšnjemu izhodu:

  ```
  if ( ( pid_out > (m_last_pid_out + hysteresis) ) ||
       ( pid_out < (m_last_pid_out - hysteresis) ) ) {
      m_last = out;
  } else {
      out = m_last;
  }
  ```

- ***mrtva cona*** (angl. dead zone): z metodo *SetDeadZone()* se nastavi t. i. mrtva cona. Mrtva cona je, kadar je izhod krmilnika PID prenizek in koračni motorji ne zmorejo premakniti vijaka za tako majhno vrednost. Če je izhodna vrednost nižja od nastavljenega parametra *a_dead*, se izhod krmilnika nadomesti z 0.

3.3 Izhodni del

Kot je bilo omenjeno v prejšnjih poglavjih, se za izhod uporabljajo koračni motorji McLennan z RS485-vmesnikom. Snovalci pospeševalnika EMMA so se odločili za uporabo teh motorjev zaradi njihove zanesljivosti in dobre podpore podjetja. Z motorji se povežemo preko McLennanovega mrežnega pretvornika in razdelilnika. Izhodni del zanke je tako sestavljen iz dveh delov:

- **modula za mrežno komunikacijo s pretvornikom**,
- **modula za krmiljenje motorjev s pomočjo serijskega ukaznega jezika**.

Modul za komunikacijo s pretvornikom McLennan

Modul za komunikacijo s pretvornikom McLennan je vmesnik TCP/IP med krmilnim algoritmom in modulom za krmiljenje motorjev. Modulu za krmiljenje motorjev omogoči povezavo do pretvornika, sam pa direktno ne krmili motorjev. Razred *McLennanConnection* je definiran kot:

```
class McLennanConnection
{
public:
    McLennanConnection(const std::string &a_hostname,
                       unsigned short a_port);
    virtual ~McLennanConnection();

    bool ExecuteCmd(const char* a_cmd);

private:
    void Connect();
    void Disconnect();

    std::string       m_hostname;
    unsigned short    m_port;

    int               m_socket;
    sockaddr_in       m_addr;

    omni_mutex m_socket_x;
};
```

Konstruktor *McLennanConnection()* kot argument potrebuje mrežni ali IP-naslov pretvornika *a_hostname* in mrežna vrata *a_port*. Ob konstruiranju si objekt zapomni argumenta, a ne sproži nobene akcije. Povezava se odpre šele, ko sistem prvič poizkuša poslati ukaz s pomočjo metode *ExecuteCmd()*. Metoda vrne logično vrednost 'res' (angl. true), če se je povezava uspešno vzpostavila. To pomeni, da se je znakovni niz *a_cmd* uspešno prenesel in je pretvornik odgovoril s potrdilnim nizom. Potrdilni niz je vedno sestavljen iz 2 bajtov. Prvi bajt je ponovljeni bajt znakovnega niza *a_cmd*, ki mu sledi znak '%'. Če se mrežna komunikacija ne more uspešno vzpostaviti ali pretvornik ne odgovori s potrdilnim znakovnim nizom, metoda *ExecuteCmd()* vrne 'narobe' (angl. false). Po prvem klicu te metode se mrežna povezava ne prekine, temveč se uporablja ves čas delovanja sistema. Povezava se prekine šele v destruktorju razreda. Za mrežno komunikacijo se uporabljata sistemska klica *write()* in *read()* operacijskega sistema GNU/Linux. Privatni metodi *Connect()* in *Disconnect()* poskrbita za odpiranje in zapiranje mrežne vtičnice (angl. socket) UNIX preko sistemskih klicev *socket()* (odprtje vtičnice), *connect()* (odprtje povezave) in *close()* (zaprtje povezave in vtičnice).
Privatni člani razreda *McLennanConnection* so:

- ***m_hostname***: kopija mrežnega ali IP-naslova, ki je podan kot argument konstruktorju razreda;

- ***m_port***: kopija mrežnih vrat, ki so podana kot argument konstruktorju razreda;

- ***m_socket***: deskriptor vtičnice (angl. socket descriptor), ki jo dodeli sistem ob njenem odprtju. Pred odprtjem povezave ima rezervirano vrednost -1;

- ***m_addr***: struktura, ki vsebuje opis vtičnice. Posreduje se jo kot argument sistemskemu klicu *connect()* in vsebuje podatke o naslovu računalnika, mrežnih vratih in tipu povezave;

- ***m_socket_x***: član, ki skrbi za vzajemno zaklepanje (angl. mutual exclusion - mutex) dostopanja do člana *m_socket*. Kot je bilo že omenjeno, je vsak pretvornik McLennan hkrati tudi razdelilnik, ki omogoča priklop štirih RS485-naprav. Vsak izmed teh štirih koračnih motorjev je naslovljen kot del ukaznega niza in razred *McLennanConnection* ne razlikuje med njimi. Tako si lahko štiri zanke delijo isto mrežno povezavo in hkrati zahtevajo izvajanje ukaza. Zato moramo poskrbeti, da

se naenkrat izvaja le ena zahteva, druge pa čakajo na prost vir znotraj metode *ExecuteCmd()*.

Modul za krmiljenje motorja McLennan

Razred za krmiljenje koračnega motorja McLennan *McLennanMotor* je sestavljen:

```
class McLennanMotor
{
public:
    McLennanMotor(char a_axis, McLennanConnection* a_conn);
    ~McLennanMotor() { };

    bool Move(ssize_t a_distance);
    bool SetParams(const std::string& a_params);

private:
    bool ExecuteCmd(const char* a_cmd, int a_param);

    bool SetAcc(int a_acc);
    bool SetDec(int a_dec);
    bool SetVel(int a_vel);

    McLennanConnection* m_conn;
    char          m_axisch;
};
```

Vsebuje naslednje metode:

- ***McLennanMotor()***: konstruktor kot argument vzame identifikacijo RS485-naprave (poimenovane tudi os) *a_axis* in kazalec na objekt *McLennanConnection a_conn* za mrežno povezavo. S pomočjo teh dveh argumentov lahko razred dostopa do posameznega koračnega motorja. Možni identifikatorji naprav so znaki '0' (za prvo napravo), '1', '2' in '3' (za četrto, zadnjo napravo).
- ***Move()***: sproži premik koračnega motorja za *a_distance* korakov.
- ***SetParams()***: nastavitve hitrostnega profila koračnega motorja. Znakovni niz *a_params* je sestavljen:

[ACC]:[DEC]:[VEL],

kjer so *[ACC]*, *[DEC]* in *[VEL]* vrednosti za pospešek, pojemek in končno hitrost. Ob klicu metode *SetParams()* se za vsako od teh vrednosti kliče primerna privatna metoda (*SetAcc()*, *SetDec()* ali *SetVel()*).

- ***ExecuteCmd()***: metoda za formatiranje ukaza in klic metode *ExecuteCmd()* iz razreda *McLennanConnection*. Format, ki se pošlje pretvorniku, ima obliko:

  $$[A][CMD][PARAM][CR][LF],$$

  kjer je:

  - *[A]*: 1-znakovni identifikator naprave, ki je podan kot argument konstruktorju *a_ axis*;
  - *[CMD]*: 2-znakovni ukaz za koračni motor. Podprti ukazi so:
    - *FL*: premik koračnega motorja določeno število korakov,
    - *AC*: nastavitev pospeška,
    - *DE*: nastavitev pojemka,
    - *VE*: nastavitev končne hitrosti;
  - *[PARAM]*: parameter ukazu (na primer število korakov pri premiku, pospešek pri ukazu za nastavljanje pospeška, ...);
  - *[CR]*: prvi predefinirani znak za konec prenosa. Znak ima desetiško ASCII-vrednost 13;
  - *[LF]*: drugi predefinirani znak za konec prenosa. Znak ima desetiško ASCII-vrednost 10.

  Ko sta oba predefinirana znaka prenesena, koračni motor to razume kot konec prenosa in izvede zahtevani ukaz. Če je bil ukaz sprejet in pravilen, koračni motor v odgovoru vrne:

  $$[A]\%$$

  Odgovori z identifikacijsko številko naprave in znakom '%'.

- ***SetAcc()***, ***SetDec()***, ***SetVel()***: metode za nastavljanje hitrostnega profila. Kličejo se iz metode *SetParams()*.

Privatna člana razreda sta kopiji identifikacije naprave *a_ axis* in kazalec na objekt razreda za mrežno povezavo *a_ conn*, ki sta podana konstruktorju razreda.

Poglavje 4

Meritve in rezultati

Kot smo videli iz enačbe 2.7 v poglavju 2.2, je pasovna širina votline 127 kilohercev. To pomeni, da ima votlina, ki ima resonančno frekvenco odmaknjeno za 60 kilohercev, le še polovico nominalne amplitude. Hitrost regulacije votline je omejena zaradi različnih delov sistema. Glavna omejitev hitrosti regulacije je fizično premikanje vijaka votline preko koračnih motorjev.

Cilj počasne regulacije resonančne frekvence je, da pri počasnih spremembah temperature v okolju (10 stopinj Celzija na uro) napaka v resonančni frekvenci votline nikoli ne sme biti višja od 2 kilohercev. Če pride do večjih nenadnih odstopanj, mora krmilnik hitro popraviti resonančno frekvenco - nenaden odmik frekvence za 100 kilohercev mora biti popravljen znotraj predpisanih meja v 30 sekundah.

Meritve so razdeljene v tri sklope:

- **Meritev izračuna frekvence**: izračun resonančne frekvence z metodo *ComputeDecay()*. Meritve vključujejo posamezne in povprečene vrednosti, ki so uporabljene za krmiljenje sistema.
- **Določanje parametrov PID**: spremljanje odziva sistema z različnimi parametri krmilnika PID in določanje teh parametrov.
- **Meritev napake resonančne frekvence**: spremljanje napake izračunane resonančne frekvence skozi daljša obdobja. Na začetku je prikazana meritev brez krmiljenja zanke, sledi pa ji več meritev s krmiljenjem v različnih delovnih okoljih.

Na grafih so zaradi večje preglednosti vedno upodobljene le napake izračuna resonančne frekvence. Referenčna vrednost za izračun napake je nosilna frekvenca votline, to je 1300 MHz. Pri pretvorbi je treba upoštevati, da je izhod

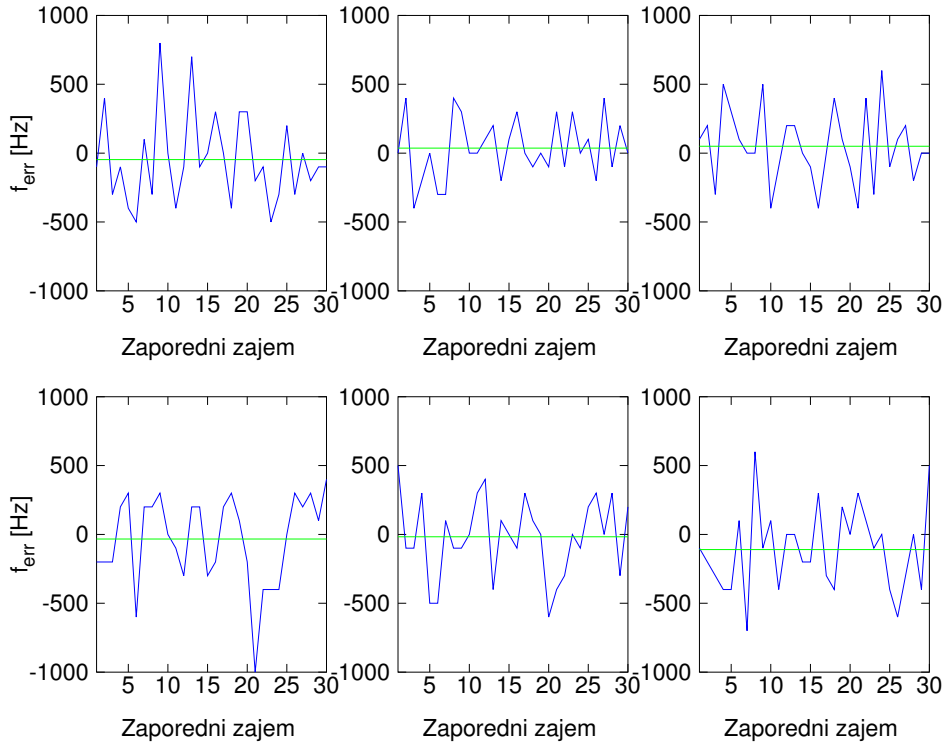
analize v vmesni frekvenci. Napaka je izračunana s pomočjo enačbe

$$f_{err} = f_{LO} - f_{RF} - f_{izhod\_analize}. \tag{4.1}$$

Na primer izračunana frekvenca 29,5454 MHz nam poda vrednost napake

$$f_{err} = 1329{,}5454e6 \text{ Hz} - 1300e6 \text{ Hz} - 29{,}5454e6 \text{ Hz} = 0 \text{ Hz}.$$

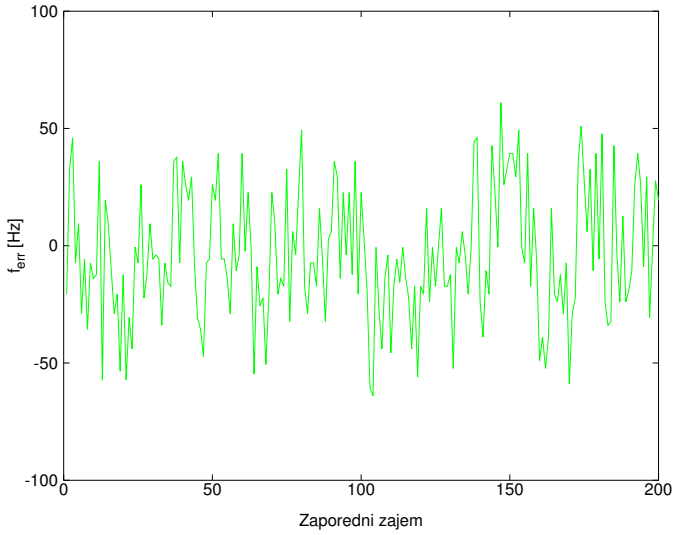
4.1 Meritve izračuna frekvence



Slika 4.1: Napaka v izračunu frekvence

Slika 4.1 je razdeljena na 6 grafov. Vsak graf vključuje 30 zaporednih meritev resonančne frekvence (modra barva) in povprečje istih 30 zajemov (zelena ravna črta). Vsak posamezen izračun frekvence je opravljen v 100 milisekundah (10 izračunov na sekundo). Kot lahko vidimo, je sam izračun resonančne frekvence relativno nenatančen (napaka niha približno ± 500 hercev, lahko tudi

več). S povprečenjem več zaporednih izračunov se ta napaka zelo zmanjša (zelena barva na grafu). Krmiljenje zanke se tako zgodi vsake 3 sekunde, saj se za vhod v krmilnik PID uporabi povprečje 30 zaporednih izračunov.



Slika 4.2: Napaka v izračunu povprečne frekvence

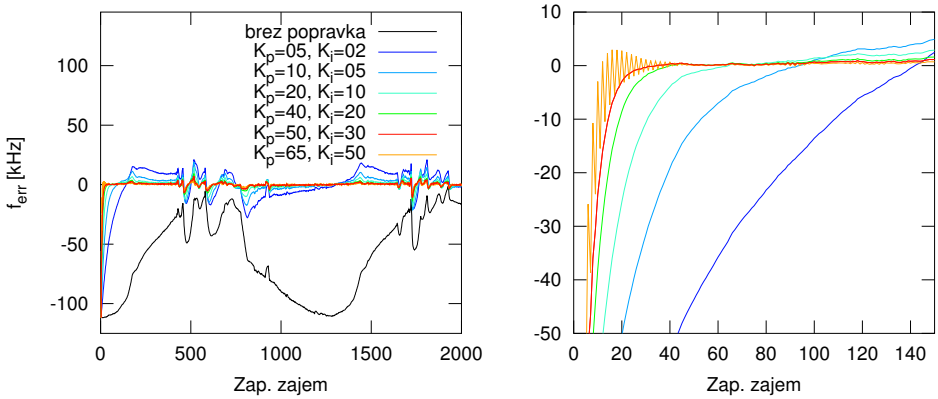
Slika 4.2 prikazuje zajem 200 zaporednih povprečenih rezultatov, kar ustreza merjenju v časovnem oknu 10 minut. Vidimo lahko, da je povprečena meritev frekvence relativno stabilna saj se napaka giblje približno $\pm$ 50 hercev.

4.2 Določanje parametrov PID

Ker je prenos iz krmilnika do votline statičen (se ne spreminja skozi čas), je določanje parametrov potrebno le ob začetni postavitvi sistema. Zato sem se odločil, da je iskanje parametrov ročno in ne avtomatizirano s katerim izmed algoritmov, opisanih v prejšnjih poglavjih. Kljub temu se da določanje parametrov zelo poenostaviti.

Za lažje določanje sem napisal krajšo simulacijo v okolju Matlab v katerem je implementirana osnovna enačba PID 3.25. Za vhodne podatke sem vzel že opravljeno meritev napake z izklopljenim krmilnikom. Treba je bilo še grobo izmeriti sam prenos sistema (kakšno spremembo frekvence lahko pričakujemo z enim korakom na koračnih motorjih). To sem izmeril ročno in prišel do rezultata, da en korak s koračnimi motorji spremeni frekvenco za približno 10

hercev. Vhodne podatke sem simuliral za večji nabor parametrov K_p, K_i in K_d in s tem pridobil primerno področje za vsakega izmed parametrov.



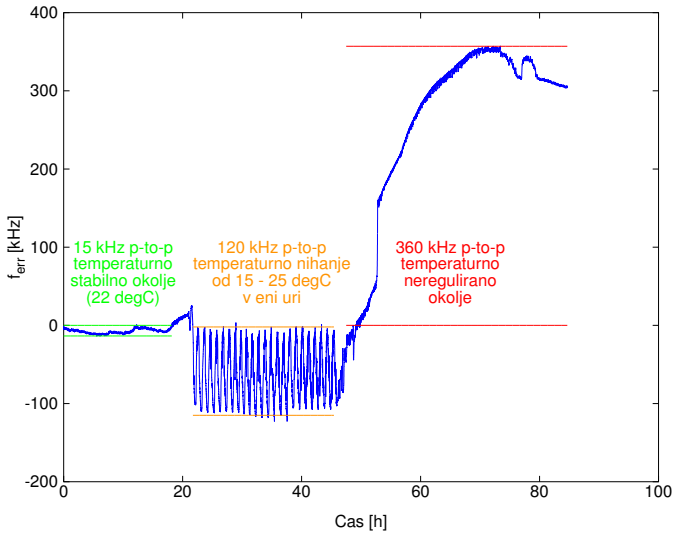
Slika 4.3: Simulacija različnih vrednosti parametrov PID

Rezultat simulacije prikazuje slika 4.3. Na sliki so zaradi preglednosti, prikazani le delni rezultati simulacije. Levi del slike prikazuje daljše časovno obdobje (približno uro in pol), desni del pa prikazuje približek začetka simulacije. Vidimo lahko, da manjši parametri že popravljajo zanko, a je napaka kljub temu velika. Ko višamo parametre, se tudi napaka manjša, dokler ne pridemo do delne nestabilnosti (točke končnega ojačenja). Na oranžni krivulji na desni strani se že vidijo oscilacije, kar nakazuje, da je zanka že na robu stabilnosti. Rdeča krivulja kaže hiter odziv sistema brez oscilacij. S tem lahko določimo približne vrednosti v resničnem sistemu. Za vrednost K_p pričakujemo, da bo okoli 50 (rdeča krivulja), za K_i pa okoli 30. Na prikazani sliki 4.3 je parameter K_d vedno nastavljen na 5. Ne smemo pozabiti, da je ta preprosta simulacija le približek pravega sistema, saj pravi sistem simulira le na grobo.

Po krajšem testiranju parametrov na razvojnem sistemu, so se za najprimernejše izkazali parametri z vrednostmi $K_p = 50$, $K_i = 30$, $K_d = 10$. Te vrednosti so uporabljene tudi za meritve v nadaljevanju.

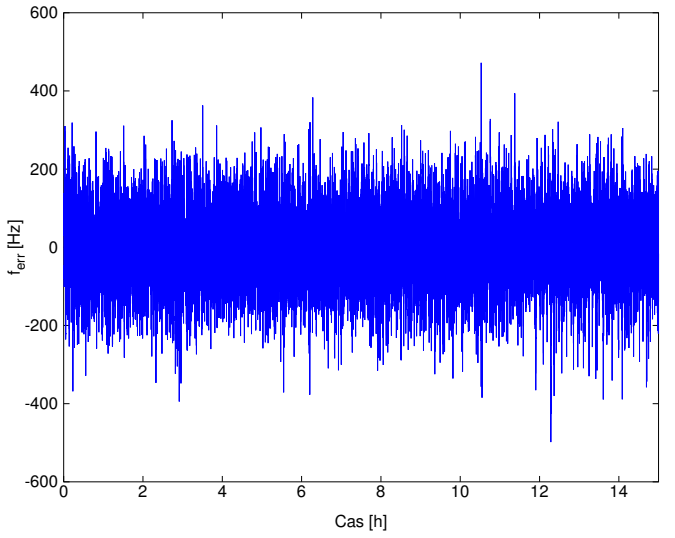
4.3 Meritve krmiljenja radiofrekvenčne votline

Slika 4.4 (zaradi večje preglednosti je ponovljena slika 2.4 iz poglavja 2.2) prikazuje meritev resonančne frekvence votline EMMA v različnih okoljih. Meritev je bila opravljena v temperaturni komori v treh režimih - prvi del je temperaturno stabilno okolje (22 °C), sledi 24 ur dolgo obdobje izmenjajoče se temperature okolja med 15 in 25 °C. Temperaturni cikel je dolg 1 uro. Za



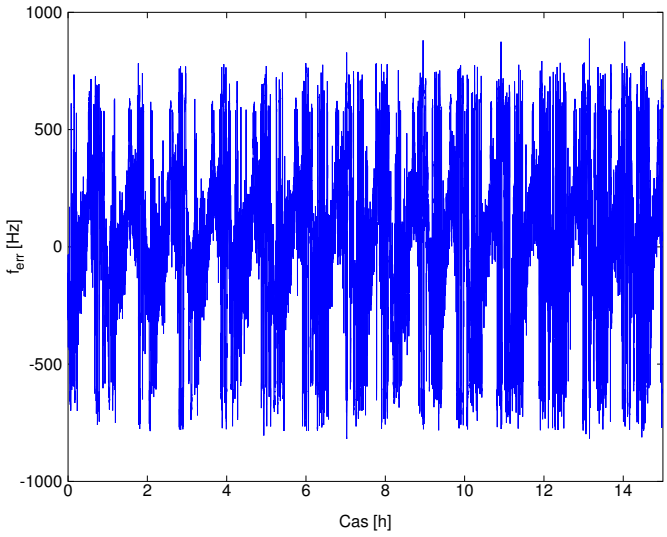
Slika 4.4: 84-urna meritev frekvence brez krmiljenja votline

tem sledi še približno 40 ur dolgo temperaturno neregulirano obdobje. Kot lahko vidimo, je nihanje resonančne frekvence votline relativno veliko, ko se spreminja temperatura okolice.



Slika 4.5: Meritev v temperaturno stabilnem okolju s krmiljenjem votline

Sledijo meritve z vključeno zanko za krmiljenje. Slika 4.5 prikazuje krmiljenje zanke v 15-urnem obdobju pri stabilni temperaturi 22 °C. Resonančna frekvenca votline je v tem obdobju zelo stabilna, saj odmik ni višji od 500 hercev. Zanimivo je, da je vrednost napake višja kot na krajši meritvi 200 zaporednih vzorcev (slika 4.2). To gre pripisati dejstvu, da je bilo okolje v temperaturni komori bolj stresno - kljub temperaturni stabilizaciji sama komora povzroča stres celotnega sistema v obliki tresenja. Kljub temu je zanka stabilna in napaka majhna skozi daljše obdobje.

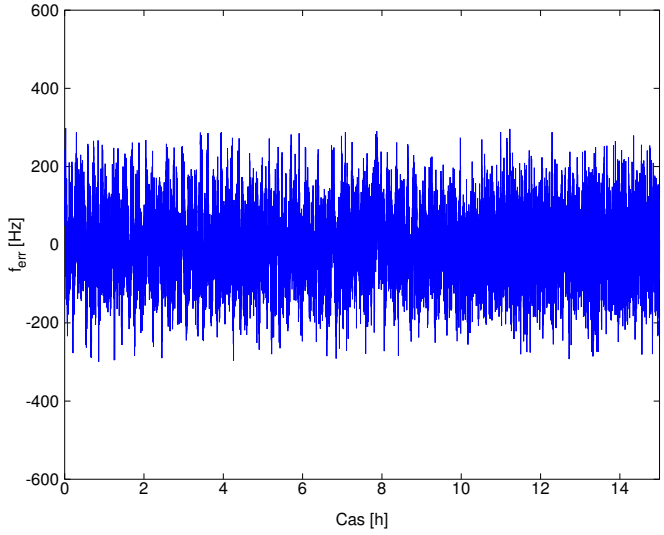


Slika 4.6: Meritev v temperaturno spreminjajočem se okolju s krmiljenjem votline

Naslednja meritev je opravljena v temperaturno izmenjajočem se okolju. V obdobju 15 ur se je temperatura vsako uro izmenjevala med 15 in 25 °C. Podatki so prikazani na sliki 4.6. Vidimo, da je sistem stabilen skozi daljše obdobje. Kljub močnemu spreminjanju okolja zanka uspešno krmili resonančno frekvenco votline, da ta nikoli ne odstopa več kot nekaj sto hercev. V podatkih lahko vidimo ponavljajoči se vzorec, ki natančno sledi spreminjanju temperature okolice. To napako bi se dalo še zmanjšati z dodatnim optimiziranjem parametrov ali hitrejšim odzivom krmilnika PID. Namesto povprečja 30 vzorcev bi lahko uporabili povprečje 10 vzorcev in tako popravljali votlino vsako sekundo namesto vsake tri sekunde. Kljub temu da bi obe rešitvi lahko pripomogli k še manjši napaki, se za dodatno optimizacijo nisem odločil, saj za to ni bilo potrebe. Z dodatno optimizacijo bi lahko zanka postala tudi manj

stabilna.

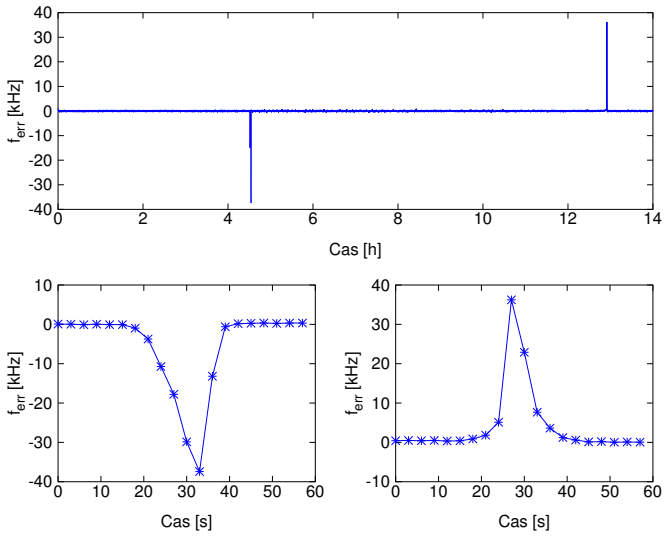
Sledi 15-urna meritev v temperaturno nereguliranem okolju. Kot lahko vidimo na sliki 4.7, je napaka razlike manjša kot pri meritvi v temperaturno stabiliziranem okolju. Razlog je po vsej verjetnosti, kot že omenjeno, dejstvo, da temperaturna komora ni izvajala fizičnega stresa na sistem. Resonančna frekvenca pri tej meritvi odstopa od nastavljene do približno 200 hercev.



Slika 4.7: Meritev v temperaturno nereguliranem okolju s krmiljenjem votline

Sledi še ena meritev, dolga 14 ur. Meritev se je začela ob približno 16:20 in zanka je normalno krmilila sistem z napako, podobno prejšnim meritvam. Po približno 4 urah in pol, torej okoli 23:00 zvečer, pa je v okolju prišlo do neznanega dogodka, ki je povzročil premik resonančne frekvence votline za več 10 kilohercev. Najvišja izmerjena napaka je bila skoraj 40 kilohercev. Krmilni algoritem je napako hitro odpravil. Po približno 8 urah in pol (okoli 7:30 naslednje jutro) je spet prišlo do podobnega, a obratnega dogodka. Kot se vidi na sliki 4.8, je takrat ponovno prišlo do napake v višini skoraj 40 kilohercev, a tokrat v drugi smeri. Zanka je napako ponovno hitro odpravila.

Na sliki 4.8 vidimo tri grafe. Na zgornjem je graf napake skozi celoten čas meritve, na spodnjih dveh grafih pa vidimo približane meritve ob močnem premiku votline v obe smeri. Kot vidimo na spodnjem levem grafu je premik resonančne frekvence trajal približno 20 sekund in je bil tako močan, da ga zanka ni zmogla sproti popravljati. Ko se je močan premik zaključil, je zanka resonančno frekvenco votline hitro (v manj kot 10 sekundah) na nastavljeno.



Slika 4.8: Druga meritev v temperaturno nereguliranem okolju

Na grafu spodaj desno vidimo obraten dogodek, ki se je zgodil 8 ur in pol kasneje - premik se je zgodil zelo hitro (približno 10 sekund), zanka pa je resonančno frekvenco votline popravila v približno 12 sekundah.

Klimatski sistemi v prostorih meritve preklopijo med dnevnim in nočnim načinom delovanja ob 18:00 popoldne in 6:00 zjutraj, tako da to ni bil razlog za premik votline. Meritve so bile opravljene v industrijski coni v Solkanu, kjer je v bližnji okolici kar nekaj težke industrije. Najverjetneje so v nočni izmeni opravljali kakšno težko delo, ki je preko električnega omrežja vplivalo na celoten sistem. Kljub temu da razlog za premik ni znan, lepo prikaže občutljivost sistema in dobro krmiljenje votline z izdelanim sistemom.

Poglavje 5

Zaključek

Kot smo videli v prejšnih poglavjih, sem za implementacijo skušal izbrati dovolj prožno arhitekturo, ki bi jo bilo mogoče uporabiti za krmiljenje več različnih vrst sistemov. Pri tem sem poizkušal za vsak posamezen del izbrati čim bolj preprosto rešitev, ki pa bi omogočala rezultat znotraj predpisanih meja. Skozi meritve na testnem okolju v laboratoriju je bilo prikazano, da sistem deluje zadovoljivo in da zanka hitro popravi resonančno frekvenco votline tudi pri ostrih spremembah okolice.

Prototipni pospeševalnik EMMA je bil uspešno zagnan poleti 2010. Pred zagonom pospeševalnika smo v 4 dneh uspešno priključili tudi napravo Libera LLRF, nastavili potrebne parametre za krmiljenje zanke in uspešno krmilili vseh 19 radiofrekvenčnih votlin. Pospeševalnik EMMA je dokazal možnost uporabe neskalirnih FFAG-tipov pospeševalnikov. Naslednji korak je izgradnja prototipnega pospeševalnika PAMELA (angl. Particle Accelerator for MEdical Applications), ki bo potrdil tudi pospešitev težjih protonov in ionov.

Slike

Literatura

[1] Instrumentation Technologies, d.d.
Libera LLRF product page
http://www.i-tech.si/products.php?meni1=12 5

[2] Wikipedia, the free encyclopedia
COM Express form factor
http://en.wikipedia.org/wiki/COM_Express 6

[3] C.D.Beard, N.Bliss, P.A.Corlett, D.M.Dykes, P.Goudket, C.Hill, P.A.McIntosh, A.J.Moss J.F.Orrett, J.H.P.Rogers, A.Wheelhouse, STFC Daresbury Laboratory, Warrington, WA4 4AD, UK A.Bogle, T.Grimm, A.Kolka, Niowave Inc, Lansing MI, USA E.Wooldridge, EDFA-JET, Culham Science Centre, Abingdon, UK
EMMA RF Cavity design and prototype testing at daresbury
http://accelconf.web.cern.ch/AccelConf/e08/papers/thpp002.pdf
9

[4] Dejan Trbojevic, Vasily Morozov
Non-scaling Fixed Field Alternating Gradient Permanent Magnet Cancer Therapy Accelerator
http://www.cockcroft.ac.uk/events/ffag11/FFAG_talks/15/pm/8.Trbojevic.pdf
9

[5] Applied Motion Products, inc.
Host Command Reference
http://www.applied-motion.com/ampinfo/manuals/Host_Command_Ref_D.pdf
11

[6] McLennan - Precision Motion Control
http://www.mclennan.co.uk/stepper.html 11

[7] Industrial Circuits Application Note
Stepper Motor Basics
http://www.solarbotics.net/library/pdflib/pdf/motorbas.pdf 11

[8] Wikipedia, the free encyclopedia
Harmonic oscillator
http://en.wikipedia.org/wiki/Harmonic_oscillator 21

[9] Wikipedia, the free encyclopedia
Q factor
http://en.wikipedia.org/wiki/Q_factor 21

[10] Wikipedia, the free enccyclopedia
PID controller
http://en.wikipedia.org/wiki/PID_controller 27