TARTU ÜLIKOOL

Füüsika-keemiateaduskond

Eksperimentaalfüüsika ja tehnoloogia instituut

JOEL KUUSK

VÄLISPEKTROMEETRI JUHTIMINE JA ANDMEHÕIVE

Magistritöö
Rakendusfüüsika eriala

Juhendajad:
TÜ dotsent, f.-m.knd. ANDO OTS
TO insener, f.-m.knd. MATTI PEHK

Tartu 2005

SISUKORD

Internetist pärinev kasutatud kirjandus

1 Hand Held Remote Sensing Instrumentation.

2 FieldSpec® Pro - Product Specifications.

3 FieldSpec® Pro. User’s Guide.

4 GER 2600 Specifications.

5 GER-Series Field Portable Spectroradiometers.

6 Birch - Betula pendula.

7 ASD FieldSpec® Pro System.

8 AD7712 Data Sheet.

9 Constant-Current Chopper Drive UPS Stepper-Motor Performance.

10 L6506 Data Sheet.

Spektromeetri juhtarvuti programmi lähtetekstid ja konfiguratsioonifail.

Spektromeetri kontrolleri programmi lähtetekst.

Magistritöö PDF-failina.

1 SISSEJUHATUS

Taimkattelt peegeldunud optiline (nähtav ja lähis- ning keskmine infrapuna, 400-2500 nm) kiirgus on peamine taimkatte seisundi vahendaja optilises kaugseires. Seni on peamiselt tehnilistel põhjustel vähe mõõtmistulemusi Päikese spektri lähis- ja kesk-infrapunases osas (800-2500 nm), ehkki just selles spektripiirkonnas kujundavad mitmed biokeemilised komponendid (vesi, proteiin, ligniin, tselluloos, pooltselluloos, suhkrud, tärklis) lehtede ja okaste optilised omadused. Siit tuleneb vajadus laiendada nii laboratoorsete kui välimõõtmiste spektripiirkonda sellesse spektrialasse, mis võib teha võimalikuks kiirguslevi mudelite jaoks vajalike komponentide optilise määramise senise väga töömahuka keemilise analüüsi asemel.

Tartu Observatooriumi taimkatte kaugseire töörühmas on valmimas 800-2500 nm piirkonnas töötav spektromeeter FIRS (*Field Infrared Scanner*), mis on ette nähtud taimkattelt tagasi peegelduva päikesekiirguse spektraalse jaotuse registreerimiseks maapealsete vahenditega kättesaadaval kõrgusel. Sisendoptika muutmisega on võimalik teda kohandada ka taimede üksikute osade mõõtmisteks sobivaks.

Spektromeetri optiline skeem oli varem välja töötatud, osaliselt oli realiseeritud optiline ja mehaaniline osa. Välja olid valitud kiirgusvastuvõtjad (InGaAs PIN-fotodioodid) ning analoog-digitaalmuundur (AD-muundur). Esmaseid proovimõõtmisi tehti AD-muunduri tootja poolt kirjutatud äärmiselt piiratud võimalustega *MS Windows* keskkonnas töötava demo-programmi abil. Käesoleva magistritöö käigus lahendamist vajavaks ülesandeks püstitati spektromeetri juhtimiseks ja AD-muundurilt andmete lugemiseks vajaliku elektroonika ja vabavaralisel *Linux* platvormil töötava tarkvara väljatöötamine ning realiseerimine.

Töö teises peatükis on esitatud lühiülevaade infrapunases piirkonnas töötavatest välispektromeetritest, kolmandas on toodud spektromeetri FIRS projekteerimisülesanded. Neljandas osas on toodud ülevaade spektromeetri optilisest ja mehaanilisest osast ning kirjeldatud signaaliahela analoogelektroonika osa. Viiendas peatükis on põhjalikumalt kirjeldatud spektromeetri juhtimiseks ja andmehõiveks valmistatud elektroonikat ja tarkvara, on antud ülevaade spektromeetri juhtprogrammi kasutamisest. Töö kuuendas peatükis on kirjeldatud laboritingimustes läbiviidud esmaseid mõõtmisi ja nende tulemusi ning nende põhjal on analüüsitud spektromeetri kvaliteeti. Seitsmendas peatükis on toodud spektromeetri edasise täiustamise käigus lahendamist vajavad ülesanded. Lisades on ära toodud spektromeetri ja kontrolleri elektrilise osa skeemid, töö käigus valminud programmide lähtetekstid ja XXXV Eesti füüsikapäevade raames esitatud stendiettekanne

”Välispektromeetri juhtimine ja andmehõive”. Töös on lisana kaasa pandud ka CD, millele on kirjutatud kõik kasutatud kirjanduse loetelus viidatud internetist pärinevad materjalid, magistritöö käigus loodud spektromeetri juhtimise tarkvara lähtetekstid ja käesolev töö PDF-failina.

2 ÜLEVAADE VÄLISPEKTROMEETRITEST

Välispektromeetritele esitatavad nõudmised on palju rangemad laboritingimustes töötamiseks mõeldud mõõteriistade omadest. Muutlike valgustustingimuste tõttu tuleb mõõtmised sooritada võimalikult kiiresti. Vahelduva pilvisusega võib objekti valgustatus muutuda sekunditega kümneid ja mõnes spektripiirkonnas rohkemgi kordi. Samuti võib näiteks tuulehoog objekti liigutada ja muuta varjude asukohta ning intensiivsust. Välimõõtmisteks mõeldud seadmed vajavad autonoomset toiteallikat. Seetõttu peab nende energiatarve olema nii väike kui vähegi võimalik, sest akud on kallid ja rasked, aga välimõõtmistel on seadme kerge kaal väga oluline.

Tööpõhimõttelt võib välispektromeetrid jaotada kaheks - jadavastuvõtjaga ja skaneerivad (vahel kasutatakse ühes seadmes ka mõlemat lahendust korraga). Jadavastuvõtjaga spektromeetril on dispergeeriv element (tänapäeval reeglina difraktsioonivõre) fikseeritud ja iga mõõdetava lainepikkuse jaoks on eraldi fotoelement. Sellise konstruktsiooni eeliseks on asjaolu, et kogu spektri saab mõõta samal ajahetkel ja seetõttu valgustustingimuste muutus ei riku olulisel määral mõõtmistulemusi. Skaneerival spektromeetril on üks fotovastuvõtja mitme lainepikkuse jaoks ja dispergeeriva elemendi pööramisega suunatakse vastuvõtjale see lainepikkus, mida mõõta soovitakse. Skaneeriva riista eeliseks on hetkel mõnevõrra parem signaal-müra suhe [1], puuduseks aga asjaolu, et kogu spektrit ei saa mõõta samal ajahetkel. Kiiremad skaneerivad spektromeetrid suudavad kogu spektri mõõta millisekunditega, aeglasematel kulub selleks sekundeid.

Kuna mõõdetava kiirguse intensiivsus võib sõltuvalt mõõtmistingimustest väga tugevalt varieeruda, siis on vajalik mõõtmispiirkonna muutmise võimalus. Jadavastuvõtjaga seadmetel muudetakse selleks integreerimisaega, skaneerivatel spektromeetritel on reeglina valitav AD-muundurisse siseneva signaali võimendusaste.

Spektromeetreid toodavad paljud firmad, kuid põhiline osa nendest on ränidetektoriga ja mõõdetav lainepikkuste vahemik ei ulatu üle 1100 nm. Selliseid seadmeid, mis mõõdaks kuni 2500 nm-ni, on väga vähe ja nad on väga kallid. Järgnevalt on esitatud mõnede konkureerivate spektromeetrite lühikirjeldused.

2.1 FieldSpec® Pro NIR

Firma Analytical Spectral Devices Inc. pakub FieldSpec® Pro tooteseerias erinevaid välispektromeetreid. FieldSpec® Pro NIR on spektraalse ulatusega 1000-2500 nm ja eraldusvõimega 10 nm [2], lainepikkuste intervall kahe lugemi vahel on 2 nm. Tegu on ska-

neeriva spektromeetriga, millel on kaks termoelektriliselt jahutatavat InGaAs fotodioodi ja 16-bitine AD-muundur. Spektromeetri müraga ekvivalentse kiirguse spektraalne heledus on $2,4{\cdot}10^{-9}\ \frac{W}{cm^2{\cdot}nm{\cdot}sr}$ lainepikkusel 1400 nm ja $8,8{\cdot}10^{-9}\ \frac{W}{cm^2{\cdot}nm{\cdot}sr}$ lainepikkusel 2100 nm. Spektri mõõtmise aeg on 100 ms. Dispergeeriva elemendina on kasutusel nõgus holograafiline peegelvõre. Sisendiks on 25° vaateväljaga valguskaabel, võimalik on kasutada erinevaid vaatevälja piirajaid. Spektromeeter kaalub 7,2 kg, aku tööaeg peaks jääma vahemikku 3-5 tundi. Seadme juhtimiseks kuulub komplekti sülearvuti, juhtprogramm töötab *MS Windows* keskkonnas. Spektromeetriga on sülearvuti ühendatud rööpvärati kaudu.

Selle spektromeetri eeliseks on kiudoptiline sisend, mistõttu ei ole vaja rasket mõõteriista paigutada objekti kohale.

FieldSpec® Pro tooteseeriast on Tartu Observatooriumi taimkatte töörühm sooritanud mõõtmisi spektromeetriga FieldSpec®Pro VNIR. Seadme puuduseks on väga pikk soojenemisaeg. Juhendi kohaselt on seade kaliibritud peale 75-90 minutilist töötamist [3]. Selle aja jooksul muutub pimevool üsna märgatavalt. Sisendi väljapiirajad on valmistatud ebakorrektselt, mistõttu esinevad peegeldused väljapiiraja silindrilistelt seintelt ja seepärast jõuab vastuvõtjani kiirgust ka vaateväljast kõrvale jäävatelt objektidelt. Peale väljapiiraja ümberkonstrueerimist (vaateväli jäi samaks) vähenes signaali nivoo 5%. Esinenud on ka mõningaid ebastabiilsusi juhtprogrammi töös, neist üks isegi sedavõrd tõsine, et nõudis tarkvara üleinstalleerimist. Samuti on ebamugav asjaolu, et spektrid salvestatakse binaarkujul ja edaspidiseks andmetöötluseks tuleb nad teise programmiga tekstifailideks konverteerida.

On alust arvata, et samad puudused on ka mudelil FieldSpec® Pro NIR.

2.2 GER2600

GER2600 on firma GER poolt pakutav välispektromeeter lainepikkuste skaalaga 350-2500 nm [4, 5]. Spektraalne lahutusvõime on 1,5 nm vahemikus 350-1050 nm ja 11,5 nm vahemikus 1050-2500 nm. Kasutusel on 512-elemendiline (Si) ja 128-elemendiline (PbS) jadavastuvõtja, dispergeerivateks elementideks on kaks liikumatult kinnitatud difraktsioonivõret. Spektromeetri müraga ekvivalentse kiirguse spektraalne heledus on $(3...6) \cdot 10^{-9}\ \frac{W}{cm^2{\cdot}nm{\cdot}sr}$ nähtavas ja $9 \cdot 10^{-9}\ \frac{W}{cm^2{\cdot}nm{\cdot}sr}$ infrapunases diapasoonis. Standardne vaateväli on 3°, kuid soovi korral on kasutatavad ka 10° ja 23° vaateväljad. AD-muundamine on 16-bitine, spektri registreerimiseks kulub minimaalselt 50 ms. Spektromeeter kaalub 5,3 kg, aku tööaeg on neli tundi. Seadet juhitakse sülearvutiga jadavärati kaudu, juhtprogramm töötab *MS DOS* keskkonnas.

GER2600-ga on Tartu Observatooriumi taimkatte kaugseire töörühmas sooritatud mõõtmisi ja esile tulid mitmed vajakajäämised seadme ja tema juhtprogrammi disainis. GER2600 puuduseks on asjaolu, et tal ei ole valguskaabli sisendit. Seetõttu tuleb ta seada tervenisti vaadeldava objekti kohale, kuid suure kaalu tõttu on seda väliolukorras ebamugav teha, kuna vajatakse vastukaaluga statiivi, et vältida ümberkukkumist. Kuigi GER2600 on varustatud lasersihikuga, et oleks võimalik täpselt määrata vaatevälja jäävat piirkonda, ei ole see funktsionaalsus siiski reaalselt välimõõtmistel kasutatav, kuna puudub võimalus laserit juhtarvuti vahendusel sisse ja välja lülitada. Laseri lüliti asub seadme korpuse küljes, aga seda enne mõõtmist lülitada ei saa, kuna siis tallataks ära mõõteobjekt.

Väga suureks puuduseks oli ka seadme juhtprogramm. Esiteks osutus ta äärmiselt ebastabiilseks, muutes tihti seadme ja juhtarvuti kontaktivõimetuks. Teiseks ei ole teada programmi lähtekoodi, mistõttu puudub igasugune info temas tehtava kaliibrimise ja esmase andmetöötluse kohta. PbS vastuvõtja on aga väga temperatuuritundlik, seetõttu on korrektsioon vajalik. Kahjuks selgus, et see korrektsioon ei olnud kuigi kvaliteetne ja mõõtmispiirkonna üleminekul ühelt kiirgusvastuvõtjalt teisele esines mõõdetud signaalis märgatav hüpe. Lisaks muutusid vastavalt juhendile sooritatud mõõtmiste korral kehvades valgustustingimustes pikematel lainepikkustel näidud negatiivseteks (põhjuseks halb pimesignaali korrektsioon).

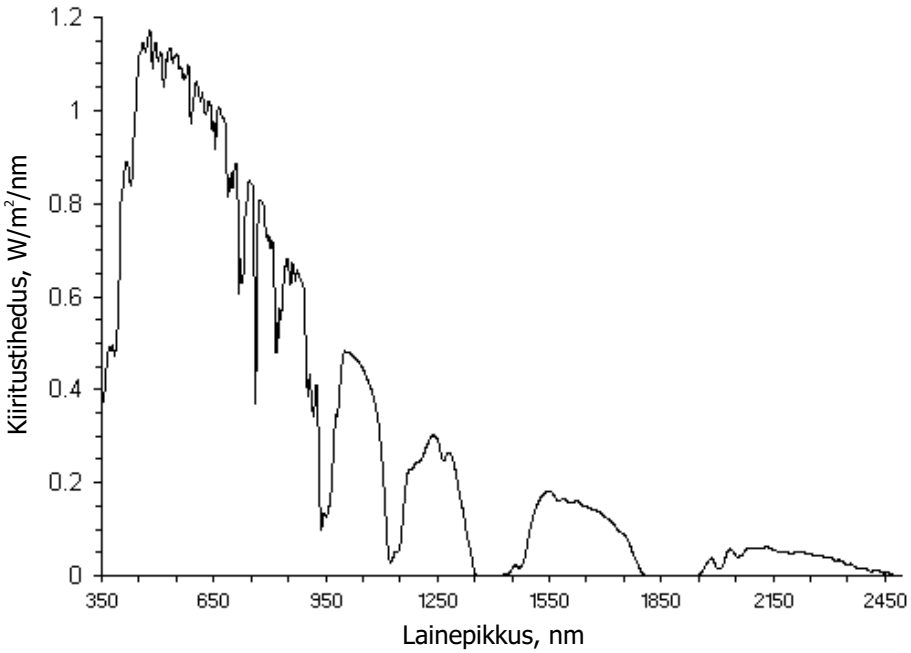
3 SPEKTROMEETRI PROJEKTEERIMISÜLESANDED

Spektromeetrile FIRS esitatavad nõudmised olid järgmised.

- Seadme eesmärgiks välitingimustes taimkatte peegeldumisspektri (spektraalse heleduskoefitsiendi) mõõtmine.
- Mõõtmispiirkond 800-2500 nm.
- Spektraalne lahutusvõime 10 nm.
- Lugemite spektraalintervall 5 nm.
- Vaateväli 10° lisavõimalusega seda hiljem muuta sisendoptika vahetusega.
- Tööaeg autonoomse toiteallikaga vähemalt 8 tundi.
- Spektromeetri ja juhtarvuti maksimaalne võimalik vahemaa vähemalt 5 meetrit.
- Tekstirežiimis töötamist võimaldav juhtprogramm.
- Juhtprogrammi töökeskkonnaks vabavaraline operatsioonisüsteem *Linux*.
- Tabuleeritud kujul mõõtmistulemuste salvestamine faili arvuti kõvakettale koos lisainformatsiooniga mõõtmise aja ja spektromeetri tööparameetrite kohta.

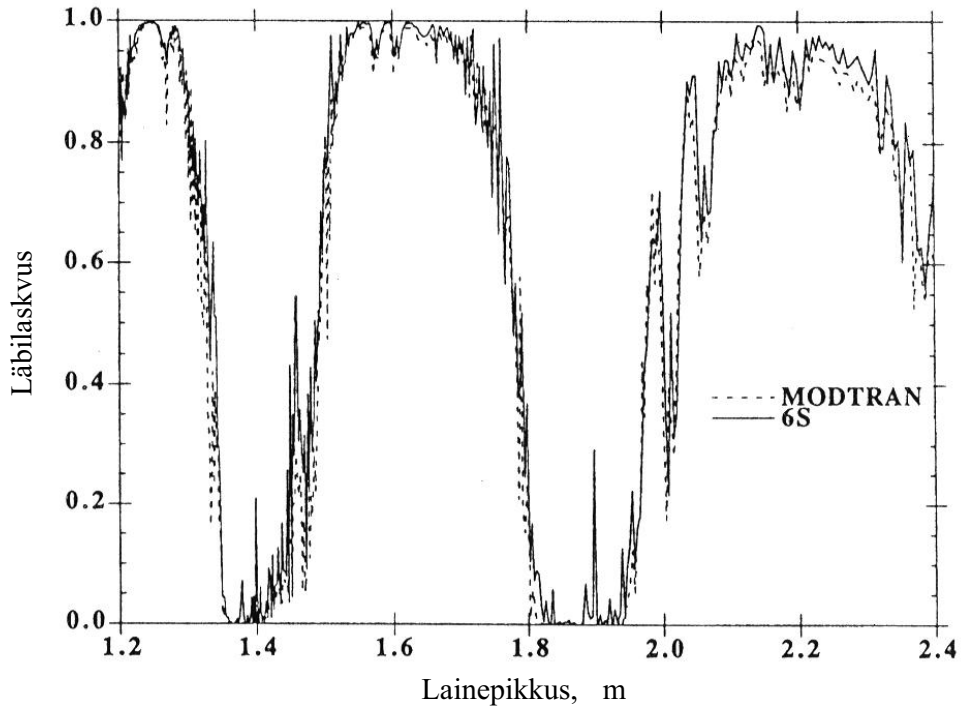
Kahest esimesest nõudmisest tulenevalt on spektromeetriga registreeritavad signaalid väga nõrgad. Päikese kiiritustihedus maapinnal kahaneb kiiresti infrapunases lainealas (vt. joonis 1), lisaks on 1400 nm ja 1900 nm ümbruses ning üle 2400 nm veeauru poolt põhjustatud tugevad neeldumisribad, kus kiiritustihedus kahaneb praktiliselt nullini. Joonisel 2 on kujutatud tüüpilise Eestimaa suvise atmosfääri läbilaskvus arvutatuna kahe erineva teoreetilise mudeli abil [6].

Peegeldumisspekter saadakse objektilt ja etalonilt (kalibreeritud peegeldumisspektriga plaat) peegeldunud kiirguse mõõtmistulemuste suhtest. Kuna mõõdetav signaal läbib mõlemal juhul sama trakti, siis sel juhul taanduvad välja kõik valgusallika kiirgusspektrist ja seadme tundlikkusest tingitud efektid. Veeauru neeldumisribade kohal, kus signaal on väga nõrk ja signaal-müra suhe seetõttu väike, saadakse kehvemates valgustustingimustes peegeldumisspektris tulemuseks põhiliselt müra. Joonisel 3 on näitena toodud noorte kaskede peegeldumisspekter mõõdetuna tööstusliku spektromeetriga GER2600 [7]. Selgelt on näha müra veeauru neeldumisribade kohal.

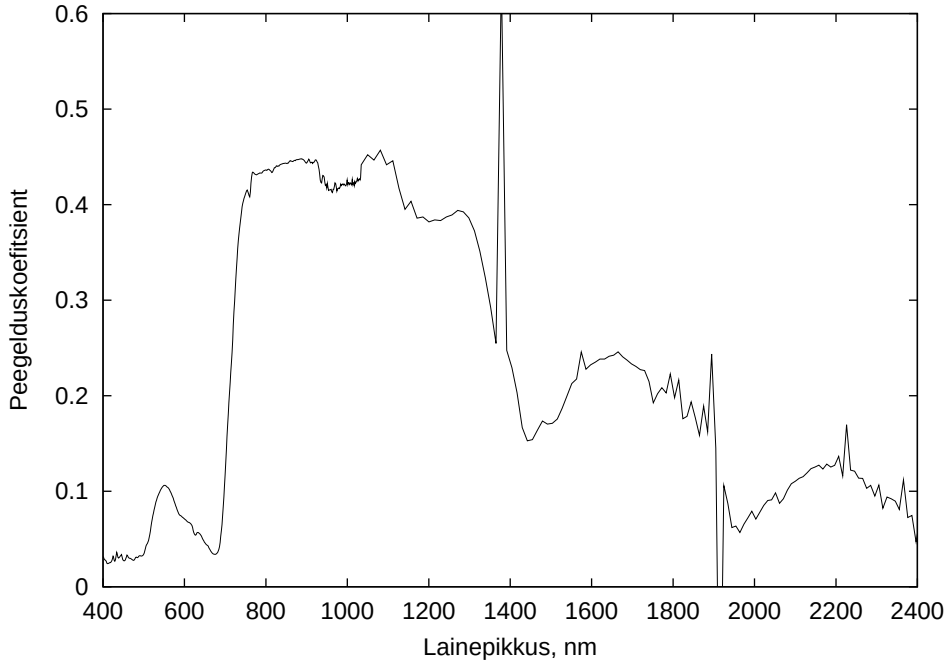


Joonis 1: Atmosfäärikihti läbinud päikesekiirguse energiaspekter maapinnal [8].

Mõõtmised on näidanud, et tihedas alustaimestikuta kuusikus on lainepikkusel 1000 nm mõõdetava kiirguse spektraalne heledus suurusjärgus $2,2 \cdot 10^{-6}\ \frac{W}{nm \cdot cm^2 \cdot sr}$. Arvestades päikesekiirguse energiaspektri langemist lainepikkuse suurenedes, aga ka peegeldusteguri spektraalsetest variatsioonidest põhjustatud kiirgusenergia kahanemist, saame 2500 nm kohal mõõdetava kiirguse spektraalse heleduse suurusjärguks $4 \cdot 10^{-9}\ \frac{W}{nm \cdot cm^2 \cdot sr}$. Kiiritustiheduse muutuseks ühe mõõtmise jooksul saame seega umbes 500 korda, mis nõuab vähemalt 9-bitist AD-muundamist. Kuna aga tulemuseks on vaja saada peegeldumisspekter, mis on kahe mõõtmistulemuse jagatis, siis kvantimismüra minimaalse signaali korral ei tohi olla väga suur. Hinnanguliselt vajame mõõtmisteks 14-16-bitist AD-muundurit. Lisaks ühe mõõtmise jooksul esinevale kiirguse intensiivsuse muutusele peab arvestama, et lageda platsi peal pilvitu ilmaga on kiirguse intensiivsus umbes kaks suurusjärku suurem kui pilves ilmaga tihedas metsas. Seega tuleb kasutada vähemalt 22-24-bitist AD-muundurit või kasutada muudetava võimendusteguriga võimendusastet AD-muunduri ees.



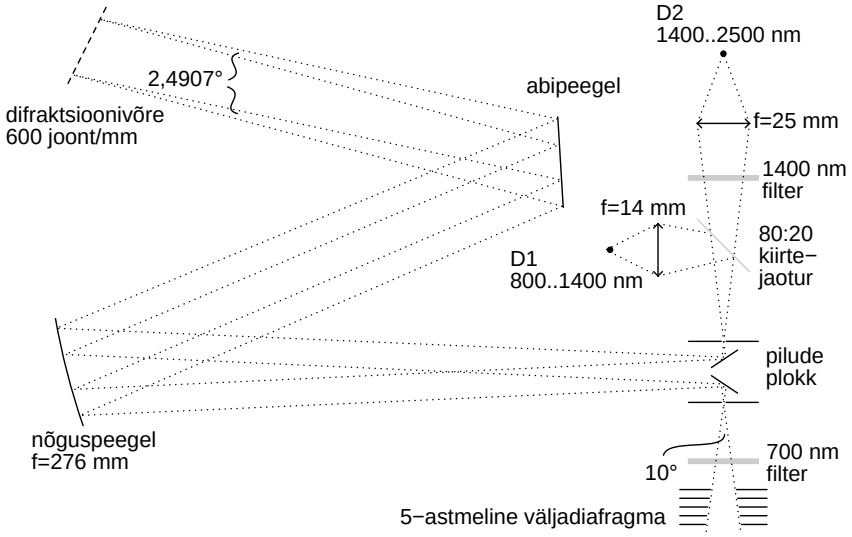
Joonis 2: Tüüpilise Eestimaa suvise atmosfäärikihi läbilaskvus arvutatuna kahe erineva mudeliga [6].



Joonis 3: Noorte kaskede peegeldumisspekter mõõdetuna spektromeetriga GER-2600 [7].

4 SPEKTROMEETRI OSADE KIRJELDUS

4.1 Spektromeetri optiline skeem



Joonis 4: Kiirte käik spektromeetris FIRS.

Kasutusel on klassikaline autokollimatsioonisüsteem (tuntud ka Littrow süsteemi nime all [9]), mis tähendab seda, et difraktsioonivõrele langev kiir ja sealt difrageerunud vaadeldav kiir on kollineaarsed. Kuna sisend- ja väljundpilu kohakuti panek pole küll võimatu, kuid siiski üsna keeruline, siis on kasutatud lahendust, kus pilud on kõrvuti. Seetõttu ei ole võrele langev ja sellelt difrageeruv vaadeldav kiir täielikult kollineaarsed, vaid nende vaheline nurk on $2,4907°$.

Joonisel 4 on kujutatud kiirte käik spektromeetris. Kiirgus siseneb spektromeetrisse läbi viieastmelise väljadiafragma, mis piirab vaatevälja $10°$ peale. Enne sisendpilu asub esimene klaasist järgueraldusfilter, mis neelab alla 700 nm lainepikkusega valguse. Peale sisendpilu läbimist suunatakse kiirgus diagonaalpeegli abil kollimeerivale nõguspeeglile, mille ristlõikeks on teljeväline parabool ja valgusjõud on 1:5,7. Kollimeeritud kiirtekimp peegeldub tasapinnaliselt abipeeglilt ja jõuab difraktsioonivõrele (peegelvõre), millel on 600 joont/mm ja efektiivsuskõvera maksimum lainepikkusel 1320 nm (efektiivsus 70%). Tasapinnaline abipeegel on vajalik kiirte teepikkuse suurendamiseks, sest sel juhul on paremini täidetud autokollimatsiooni tingimus. Pärast võrelt difrageerumist peegeldub kiirgus uuesti abipeeglilt ja fokuseeritakse nõguspeegli ja diagonaalse tasapeegli abil väljundpilule. Väljundpilu taha on kiirte teele asetatud kiirtejaotur, mis umbes 80% kiirgusest laseb läbi pikalainelise kiirgusvastuvõtja suunas ja 20% suunab lühilainelise

kiirgusvastuvõtja poole. Kuna tegu on dielektrilise kattega peegliga, siis neeldumise võib arvestamata jätta. Pikalainelise vastuvõtja ees asub klaasfilter, mis neelab alla 1400 nm lainepikkusega kiirguse. Edasi läbib mõlema vastuvõtja suunas liikuv kiirgus kondensoroptika ja jõuab lõpuks vastuvõtjani. Lühilainelise vastuvõtja ees on kasutusel 14 mm, pikalainelise vastuvõtja ees 25 mm fookuskaugusega kondensor.

Pilude plokk, kollimeeriv peegel ja abipeegel on pärit spektrofotomeetrist ИКС-29.

Difraktsioonivõre korral määrab difrageerunud valguse intensiivsuse peamaksimumide asukohad valem

$$d(\sin\phi + \sin\theta) = m\lambda, \tag{1}$$

kus d on võrekonstant, θ on võrele langeva kiire ja võre pinnanormaali vaheline nurk, ϕ on võrelt difrageerunud kiire ja võre pinnanormaali vaheline nurk, m on peamaksimumi järk ja λ on lainepikkus [9]. Difrageerunud kiire nurk loetakse positiivseks, kui kiir jääb võre pinnanormaalist samale poole kui võrele langev kiir. Sellest valemist on näha, et samasse suunda difrageeruvad näiteks esimest järku peamaksimum lainepikkusel λ_1 ja teist järku peamaksimum lainepikkusel $\lambda_2 = 0,5\lambda_1$. Selle vältimiseks tuleb difraktsioonivõrega spektromeetrites kasutada filtreid, millega takistatakse kõrgemat järku peamaksimumide tekkimine vaadeldavas spektrialas. Neid nimetatakse järgueraldusfiltriteks, spektromeetris FIRS on neid kasutusel kaks.

Difraktsioonivõre joondispersioon on määratud valemiga [10]

$$\frac{\mathrm{d}l}{\mathrm{d}\lambda} = \frac{f\left(\sin\phi + \sin\theta\right)}{\lambda\cos\phi}, \tag{2}$$

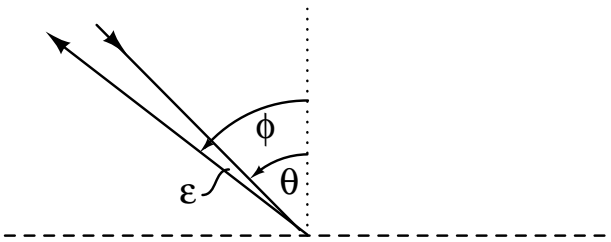
kus $f = 276$ mm on fokuseeriva peegli fookuskaugus. Arvestades, et

$$\phi = \theta + 2,4907^\circ \tag{3}$$

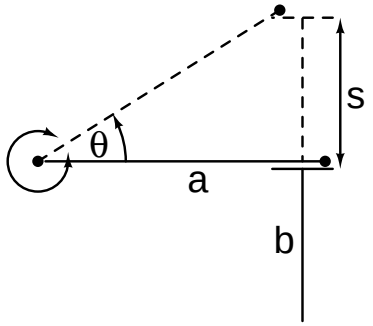
(vt. joonis 5), saab valemist (1) leida, et

$$\sin\theta = \frac{\lambda}{2d} - \sqrt{\left(\frac{\lambda}{2d}\right)^2 - \frac{\lambda^2 - d^2\left(\sin\varepsilon\right)^2}{2d^2\left(1+\cos\varepsilon\right)}}. \tag{4}$$

Selle valemi järgi arvutades vastab lainepikkusele 800 nm nurk $\theta = 12,6445^\circ$ ja lainepikkusele 2500 nm nurk $\theta = 47,3604^\circ$. Seega valemi (2) kohaselt on lineaarsed dispersioonid vastavalt 0,1716 mm/nm ja 0,2568 mm/nm. Tagamaks spektraalse lahutusvõime vähemalt 10 nanomeetrit, tuleb pilude laiuseks seada 1,72 mm. Lainepikkusel 2500 nm annaks see lahutusvõimeks 6,7 nm.



Joonis 5: Kiirte difraktsioon peegelvõrelt.



Joonis 6: Siinusmehhanism.

4.2 Difraktsioonivõret pöörava mehhanismi kirjeldus

Difraktsioonivõre pööramiseks kasutatakse siinusmehhanismi. Selle tööpõhimõtet aitab selgitada joonis 6. Hoob a toetub tõukurmehhanismile b. Kui tõukurmehhanismi b liigutada teepikkuse s võrra, siis hoob a pöördub nurga

$$\theta = \arcsin \frac{s}{a} \tag{5}$$

võrra, kus a on hoova a pikkus. Hoob a on kinnitatud difraktsioonivõre pöörlemistelje külge.

Difrageeruva valguse intensiivsuse peamaksimumide asukohad määrab ära valem (1). Selle valemi võib teisendada kujule

$$2d \sin \frac{\theta + \phi}{2} \cos \frac{\theta - \phi}{2} = m\lambda \,. \tag{6}$$

Autokollimatsioonilise süsteemi korral on langev kiir ja vaadeldav difrageerunud kiir kollineaarsed. Sel juhul $\varepsilon = 0$ ja $\phi = \theta$ (vt. joonis 5). Arvestades seda asjaolu ning valemit

(5), saame valemist (6) esimest järku peamaksimumi korral

$$\lambda = 2d \sin\theta = 2d\frac{s}{a}\,. \tag{7}$$

Nagu näha, sõltub siinusmehhanismi kasutamise korral lainepikkus lineaarselt tõukurmehhanismi b liigutamisest.

Tõukurmehhanismina b kasutatakse poolemillimeetrise keermesammuga mikromeeterkruvi. Kasutatav samm-mootor ДШИ-200-1-1teeb ühe täispöörde jooksul 200 sammu, seega ühe sammuga liigub kruvi edasi 2,5 μm. Difraktsioonivõrel on 600 joont/mm, millest saame võrekonstandiks 1,67 μm. Hoova a pikkus on 41,67 mm. Kõike seda arvestades leiame, et ühe sammuga muutub lainepikkus

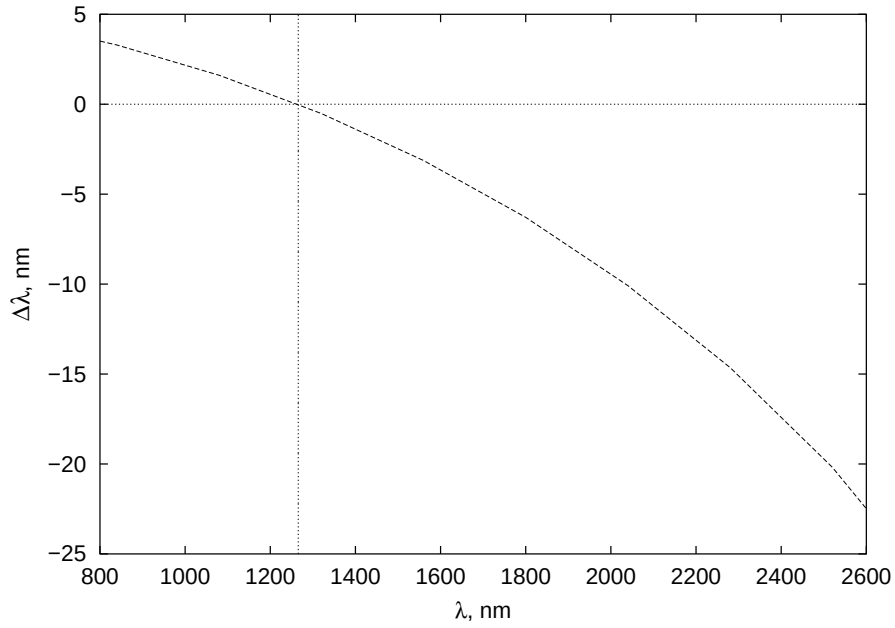
$$\delta\lambda = \frac{2d}{a}\delta s = \frac{2 \cdot 1,67 \cdot 10^{-6}}{41,67 \cdot 10^{-3}} 2,5 \cdot 10^{-6}\ \text{m} = 2 \cdot 10^{-10}\ \text{m} = 0,2\ \text{nm}. \tag{8}$$

Kuna sisend- ja väljundpilu ei ole kohakuti, siis ei ole tegemist ideaalselt autokollimatsioonilise süsteemiga st. võrele langev ja sealt difrageeruv vaadeldav kiir ei ole kollineaarsed, vaid nende vahel on nullist erinev nurk ε - antud juhul $\varepsilon = 2,4907°$. Sel juhul arvestades valemit (5) saab esimest järku peamaksimumi korral valemi (1) kirja panna kujul

$$\lambda = d\left(\frac{s}{a} + \sin\left(\arcsin\left(\frac{s}{a}\right) + \varepsilon\right)\right). \tag{9}$$

Nagu näha, ei sõltu lainepikkus enam lineaarselt varda b liigutamisest. Joonisel 7 on kujutatud lainepikkuste skaala erinevust ideaalsest autokollimatsioonilisest süsteemist juhul, kui skaalad on nihutatud kohakuti lainepikkusel 1265,6 nm, mis vastab HeNe laseri valguse teisele difraktsioonijärgule.

Joonisel 7 esitatud sõltuvus kehtib hoova a pikkuse 41,67 mm korral. Hoova konstruktsiooni tõttu on tema valmistamine ideaalse pikkusega keerukas ja reaalne erinevus autokollimatsioonilisest süsteemist tuleb määrata katseliselt. Selleks sobib HeNe laseri valgus, mille lainepikkus on 632,8 nm. Kui eemaldada kiirte teelt järgueraldusfiltrid, siis on võimalik jälgida 2., 3. ja 4. järku peamaksimumi asukohta, mis peaks jääma vastavalt lainepikkustele 1265,6 nm, 1898,4 nm ja 2531,2 nm. Neist viimane on vaadeldav, kui lainepikkuste piirkonna pikemalainelist piirajat nihutada kaugemale 2500-le nanomeetrile vastavast asukohast. Eelmainitud mõõtmistest on täpsemalt kirjutatud peatükis 6.1.



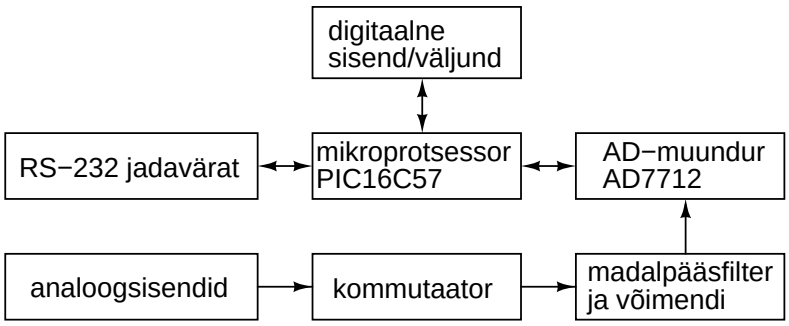
Joonis 7: Lainepikkuste skaala erinevus ideaalsest autokollimatsioonilisest süsteemist.

4.3 Signaaliahela analoogelektroonika ja AD-muunduri kirjeldus

Kiirgusvastuvõtjatena kasutatakse kahte firmas Hamamatsu valmistatud InGaAs PIN-fotodioodi - parandatud lühilainetundlikkusega G5125-10 ($\lambda\lambda$ 700-1700 nm, Ø1 mm) ja pikalainelist dioodi G5853-23 ($\lambda\lambda$ 900-2500 nm, Ø3 mm), mis on ka termoelektrilise jahutusega. Praeguseks ei ole veel lõplikult valminud vastuvõtjate optika ja analoogelektroonika osa. Esialgsete mõõtmiste läbiviimiseks on kasutusel elementaarne vool-pinge-muundur, mis baseerub operatsioonivõimendil 544УД1А (IC3 joonisel 23 lk 47). Häirete vähendamiseks toidetakse vool-pinge-muundurit eraldi kahelt 9-voldiselt patareilt, mitte kontrollerit ja samm-mootorit toitvatelt akudelt.

Fotodioode kasutatakse spektromeetris fotogalvaanilises (voolugeneraatori) režiimis. Võrreldes fotojuhtiva režiimiga, kus fotodioodi kasutatakse eelpingestatud lülituses, on fotogalvaanilises režiimis reageerimise kiirus küll väiksem, kuid puudub pimevool ja pimevoolust põhjustatud müra ning seetõttu on signaal-müra suhe suurem [11]. Kuna InGaAs vastuvõtja on väga kiire reageerimisajaga, siis ei ole see mõningane kaotus töökiiruses antud signaalide mõõtmise juures probleemiks.

AD-muunduriks on kasutusel M201 firmalt Lawson Labs Inc, mille plokkskeem on kujutatud joonisel 8. Tegu on 6 kommuteeritava diferentsiaalse sisendkanaliga 24-bitise andmehõivesüsteemiga. Analoogsignaali digitaliseerimiseks kasutatakse sigma-delta tehni-



Joonis 8: Andmehõivesüsteemi Lawson M201 plokkskeem.

kal põhinevat AD-muundurit AD7712 firmalt Analog Devices[1]. AD-muunduri skaleerimise vea ja nullnivoo nihke kompenseerimise kalibratsioonivõimalus on seadmesse sisse ehitatud, sisendsignaali võimendustegur on astmeliselt muudetav vahemikus 1-128. Andmevahetus käib optiliselt isoleeritud RS-232 liidese vahendusel kiiruste vahemikus 300-9600 boodi. Seadmel on lisaks 8 digitaalset sisend- ja 12 (neist 4 on optiliselt isoleeritud) digitaalset väljundviiku. AD-muunduri toiteks kasutatakse kahte 9-voldist patareid.

AD7712 sisaldab endas digitaalset madalpääsfiltrit sageduskostega [12]

$$H(f) = \left(\frac{\sin(cf)}{cf}\right)^3, \qquad c = \frac{512C\pi}{F_{CLK}}, \qquad C = 19 \ldots 2000, \tag{10}$$

kus f on sagedus hertsides, $F_{CLK} = 10$ MHz on kasutatav taktsagedus ja C on programselt valitav parameeter, mis määrab filtri lõikesageduse[2]. Filtri sageduskoste esimene nullkoht on valitav sageduste vahemikus

$$f_0 = \frac{F_{CLK}}{512C} = 9,76 ... 1027,96 \text{ Hz}. \tag{11}$$

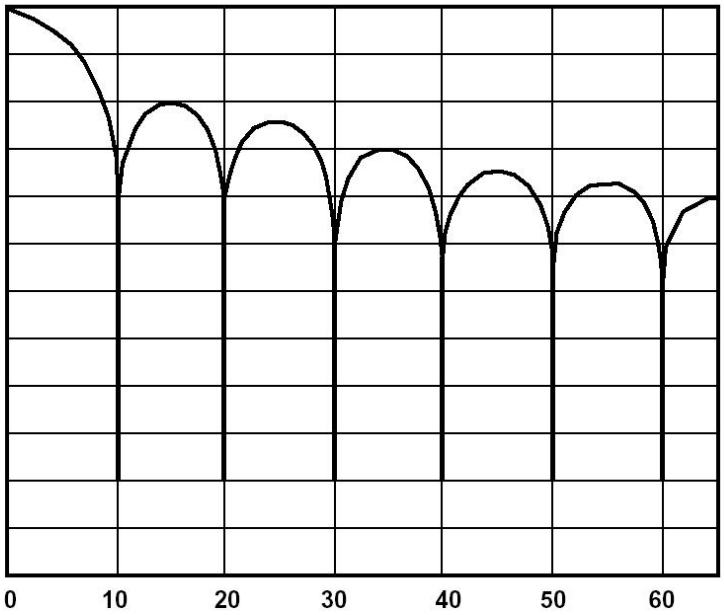
Filtri lõikesagedus on esimese nullkoha sagedusega seotud vastavalt valemile

$$f_c = 0,262 f_0, \tag{12}$$

mis annab võimalikuks lõikesageduste vahemikuks $f_c = 2,56 ... 269$ Hz. Joonisel 9 on näitena toodud AD7712 digitaalse madalpääsfiltri sageduskoste, kui parameetriks on valitud $C = 1953$, millest valemi (11) kohaselt saame sageduskoste esimese nullkoha asu-

[1]Lawson M201 on tegelikult mikroprotsessoriga PIC16C57 juhitav andmehõivesüsteem, mille sees asub 24-bitine AD-muundur AD7712. Käesolevas töös pole selline vahetegemine tarvilik ning lihtsuse mõttes käsitletakse M201 mõnevõrra laiendatud võimalustega AD-muundurina ja AD-muundurile viidates peetakse üldjuhul silmas tervet Lawson M201 andmehõivesüsteemi.

[2]Sagedus, mille juures on filtri sageduskarakteristik langenud 3dB võrra.



Joonis 9: AD7712 sisemise digitaalse madalpääsfiltri sageduskoste, kui esimese nullkoha sageduseks on valitud 10 Hz [12].

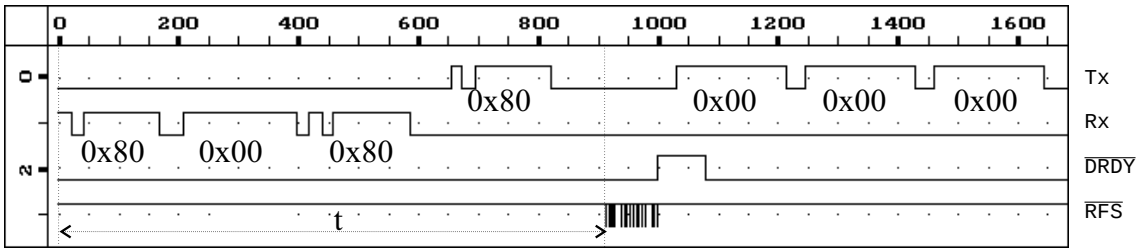
kohaks sageduse 10 Hz.

Efektiivne muundamise aeg on seotud digitaalse filtri parameetri C valikuga, uus andmesõna on saadaval filtri sageduskoste esimese nullkoha asukohale vastava sagedusega (kui esimese nullkoha sageduseks on valitud 10 Hz, siis uus andmesõna on saadaval iga 100 ms tagant). Efektiivne lahutusvõime sõltub valitud võimendusest ja filtri lõikesagedusest.

AD7712 teeb pidevalt AD-muundamist kindla diskreetimissagedusega $\frac{F_{CLK}}{512} = 19531,25$ Hz ja digitaalse filtri parameeter C määrab ära efektiivse diskreetimissageduse. Otseselt näitab ta lugemite arvu, üle mille keskmistatud tulemus andmepuhvrisse kirjutatakse. Kui muundamise tulemust andmepuhvrist välja ei loeta, siis kirjutatakse ta järgmise muundamise tulemusega üle. Lugemi võtmise käsu peale tagastatakse viimase lõpetatud muundamise tulemus, seega reaalne mõõtmine toimub pisut enne lugemi võtmise käsu saabumist.

Lisaks AD7712 sees olevale digitaalsele madalpääsfiltrile on Lawson M201 sees muudetava lõikesagedusega esimest järku analoogfilter, mille lõikesagedus võib olla 4, 40 või 400 hertsi.

Lawson M201 kasutab Microchipi PIC16C57 mikrokontrollerit, millel ei ole riistvaralist jadaväratit. Seetõttu on jadavärati funktsionaalsus realiseeritud tarkvaraliselt, kasutades



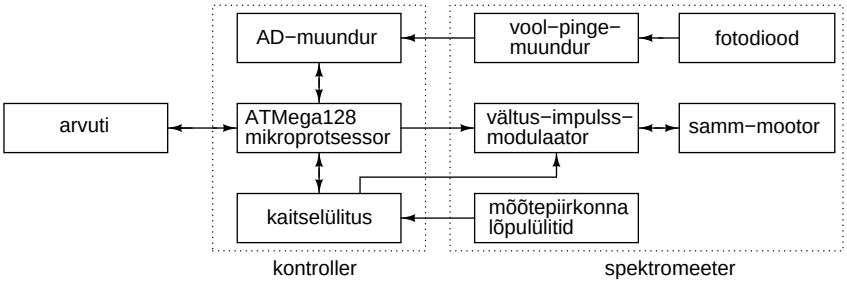
Joonis 10: AD-muunduri signaalide aegdiagramm. Loogikaanalüsaatori BlackStar 3332 diskreetimissagedus oli 200 kHz.

sisend-väljundviike, ja ei vasta seetõttu 100-protsendiliselt RS-232 standardile. Nimelt hakkab protsessor tegelema muude asjadega juba andmevahetuse viimase biti saatmise või vastuvõtmise ajal.

Joonisel 10 on näha M201 andmevahetuse signaalide aegdiagramm. Tx on spektromeetri kontrollerist M201-le saabuvad andmed, Rx M201-st kontrollerile saadetavad andmed. $\overline{\text{DRDY}}$ madal nivoo näitab, et AD7712 on lõpetanud muundamise ja seal on lugemata andmeid. $\overline{\text{RFS}}$ on AD7712 andmeside kontrollsignaal - andmeside toimub vaid madala nivoo korral.

Kõigepealt saadetakse spektromeetri kontrollerist M201-le kolmebaidine lugemi võtmise käsk 0x800080. Seejärel saadab M201 kontrollerile tagasi käsukoodi 0x80, võtab AD7712-lt lugemi ja saadab tulemuse spektromeetri kontrollerile. Kontrolleriga andmevahetuse parameetrid on 8 andmebitti, 0 paarsusbitti, 1 startbitt ja 1 stoppbitt, seega ühe andmebaidi saatmiseks kulub 10 bitiaega. Mõõtmised loogikaanalüsaatoriga erinevatel andmevahetuskiirustel andsid tulemuseks, et M201 alustab käsukoodi tagasi saatmist 39,25 bitiaja pluss 180 mikrosekundi möödudes esimese kontrollerist temale saadetud biti algusest. Käsukoodi tagasi saatmise algusest kuni AD7712-lt lugemi võtmiseni kulub veel 9,97 bitiaega pluss 244 mikrosekundit. Seega tuleb arvestada asjaoluga, et reaalne lugemi võtmine toimub 49,22 bitiaega pluss 424 mikrosekundit pärast spektromeetri kontrollerist käsu saatmise alustamist.

5 SPEKTROMEETRI JUHTIMINE JA ANDMEHÕIVE



Joonis 11: Spektromeetri ja kontrolleri plokkskeem.

Spektromeeter FIRS on mõeldud välimõõtmisteks. Seetõttu kasutatakse tema juhtimiseks sülearvutit ja teda toidetakse 12-voldise NiMH akupatarei pealt (8,3 Ah). Sülearvuti ja spektromeetri kontroller ning akud on paigutatud eraldi kohvrisse ja kontroller on spektromeetriga ühendatud kahe kaabli abil. Üks neist on mootori juhtsignaalide ja toite jaoks, teine on vool-pinge-muunduri toite ja mõõdetava signaali jaoks. Joonisel 11 on kujutatud spektromeetri ja tema kontrolleri plokkskeem.

5.1 Digitaalse juhtelektroonika kirjeldus

Spektromeetri kontrolleri keskseks osaks on Atmeli mikroprotsessor ATmega128. Protsessori valikul sai otsustavaks kahe jadavärati olemasolu. Kuna ATmega128 jadaväratid töötavad TTL nivooga signaalidega (signaalile 0 vastab 0 V ja signaalile 1 vastab +5 V), siis on skeemi lisatud kummagi jadavärati jaoks nivookonverter ST3232CN (IC1 ja IC2 joonisel 22 lk 46), mille abil on võimalik suhelda RS-232 standardiga (signaalile 1 vastab -15 V ja signaalile 0 vastab +15 V) kooskõlas olevate seadmetega.

Samm-mootori juhtimise ahelasse on lisatud kaitselülitus (IC7, IC8 ja IC9 joonisel 22 lk 46), mis takistab difraktsioonivõre pööramist kaugemale lubatud lainepikkuste vahemiku piiridest. Tavaolukorras seda lülitust ei vajata, sest liiga kaugele pööramise üle peab arvet mikroprotsessori programm. Lülitus on oluline veasituatsioonis, kui mikroprotsessori programmeerimisel on tehtud viga ja programne piirajani jõudmise kontroll mingil põhjusel ei tööta. Iga sammu tegemisel genereerib mikroprotsessori programm sõltuvalt difraktsioonivõre pööramise suunast kas *+1* või *-1* impulsi, mis siseneb kaitselülitusse, lubades vaid ühes suunas liikumise. Pikemate lainepikkuste suunas liikudes sisenevad kaitselülitusse *+1* impulsid. Kui jõutakse piirajani, siis kaitselülitus viib *ENABLE* väljundi

madalaks ja veel edasine difraktsioonivõre pööramine on riistvaraliselt takistatud. Uuesti lubatakse sammumise impulsid mootorile, kui alustatakse lühemate lainepikkuste suunas liikumist ja kaitselülitusse saabub esimene *-1* impulss. Analoogselt toimib kaitselülitus ka lainepikkuste piirkonna lühemalainelises servas asuva piirajani jõudmisel.

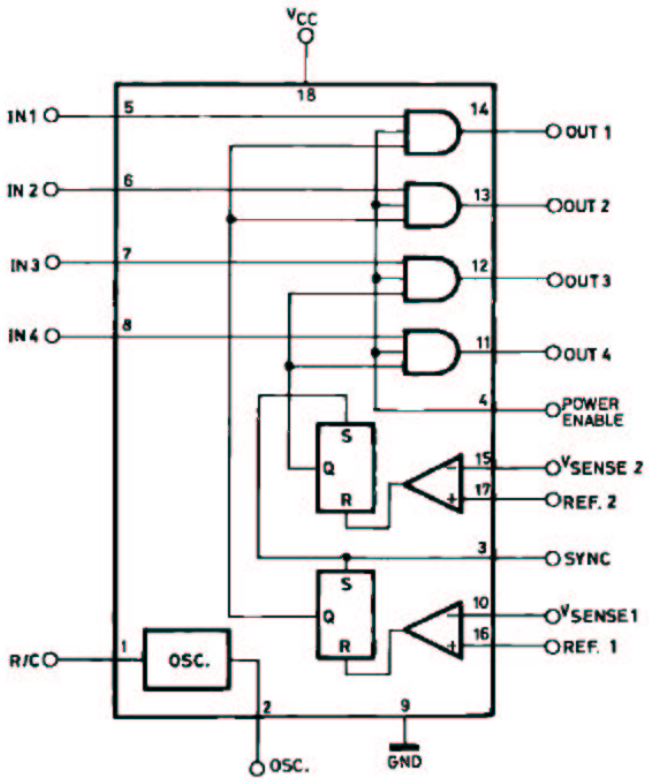
Kuna piirajani jõudes võetakse samm-mootori mähistelt pinge koheselt maha, siis võib juhtuda, et mootor ei jõua sammu sooritada. Kontroller seda aga teada ei saa ja arvestab, et samm on tehtud. Nii võib tekkida viga difraktsioonivõre asukoha määramisel. Selle vältimiseks on piirajad seatud pisut kaugemale mõõdetavate lainepikkuste piirkonna servadest. Kui on valitud spektromeetri juhtprogrammi parameeter *manualstep = 0* (juhtprogrammist ja tema konfiguratsioonifailist on täpsemalt räägitud peatükis 5.3), siis töö käigus ei lubata difraktsioonivõret pöörata piirajateni.

Mähise induktiivsuse tõttu suureneb vool samm-mootori mähises vastavalt valemile

$$i = \frac{U}{R}\left(1 - e^{-\frac{R}{L}t}\right), \tag{13}$$

kus R on mähise takistus, L on mähise induktiivsus, t on voolu sisselülitamisest möödunud aeg ja U on toitepinge [13]. Siit järeldub, et mida lühemad on impulsid ja mida kiiremini mootor pöörleb, seda väiksem on mootorit läbinud keskmine voolutugevus. Tegelikult peaks olukord olema vastupidine. Kuna mootori mähiste induktiivsust vähendada ei ole võimalik, siis tuleb suurema pöörlemissageduse jaoks järelikult suurendada kas ainult toitepinget või ühendada mähisega jadamisi ballasttakisti ja kasvatada toitepinget. Ainult toitepinge suurendamisega tekib see probleem, et väiksema pöörlemissageduse (pikemate sammuimpulsside) korral muutub vool liiga suureks ja võib mähise rikkuda, lisaks kulutatakse väikesel kiirusel asjatult palju energiat. Ballasttakistite lisamisega kahaneb valemis (13) liige $e^{-\frac{R}{L}t}$ kiiremini, millest tulenevalt ka mootorit läbiv vool kasvab kiiremini. Samas tuleb ka tõsta mootori toitepinget, et suhe $\frac{U}{R}$ jääks samaks. Selle lahenduse puuduseks on asjaolu, et ballasttakistil eraldub täiesti tarbetu energiahulk, mida akutoitel seadme korral tuleks vältida.

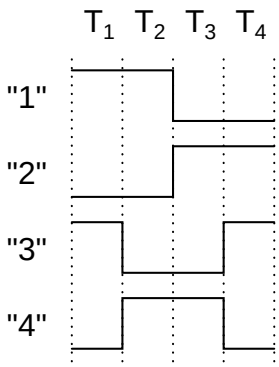
Lahenduseks on energiasäästliku vältus-impulssmodulatsiooni (*PWM - pulse width modulation*) kasutamine [14]. Võrreldes tavalise 1,5-kordse forsseerimisega lülitusega ($R_{ballast} = 1,5R_{mähis}$) osutus sääst enam kui kahekordseks. Keskmine voolutarve skaneerimisel vähenes mõnesaja milliampri võrra, jäädes alla 1,5 A, kuid maksimaalne skaneerimise kiirus suurenes kaks korda, ulatudes nüüd sooja ilmaga 1500 hertsini. Külma ilmaga kasvab liikuvate detailide vahel oleva määrde viskoossus ja maksimaalne skaneerimise kiirus väheneb.



Joonis 12: Vältus-impulssmodulaatori L6506 põhimõtteskeem [15].

Vältus-impulssmodulaatori tööd aitab selgitada joonis 12 [15]. Sisenditesse *IN1-IN4* on ühendatud kontrollerist saabuvad samm-mootori juhtsignaalid, väljunditega *OUT1-OUT4* juhitakse võtmetransistore (T1-T4 joonisel 23 lk 47). Samm-mootori mähistega on järjestikku ühendatud 0,24 Ω mõõtetakistid (R8 ja R9 joonisel 23 lk 47). Modulaator on varustatud taktgeneraatoriga, mille iga perioodi jooksul võrreldakse pinget mõõtetakistitel (modulaatori *SENSE* sisendid) tugipingega (*REF* sisendid). Kui mõõtetakistil on pinge väiksem tugipingest, siis avatakse vastava faasi transistor ja mootori mähist läbib vool. Ka nüüd suureneb vool mähises vastavalt valemile (13), kuid kasutada võib mootorile lubatud toitepingest oluliselt suuremat pinget. Teatud aja möödudes saavutab vool sellise väärtuse, et pingelang mõõtetakistil ületab tugipinge väärtuse ja transistor suletakse. Selliselt moduleerides on võimalik keskmine voolutugevus hoida lubatu piires, samas ei kulutata energiat ballasttakistite soojendamiseks. Kuna transistorid töötavad võtmerežiimis, siis ei soojene ka need olulisel määral. Mootori neljast faasist on kahe faasi juhtsignaalid ülejäänud kahe suhtes inverteeritud (vt. joonis 13). Seetõttu on kasutatud kahe faasi kohta ainult ühte 0,24 Ω mõõtetakistit, sest korraga läbib vool ainult ühte mähist.

Modulaatori *ENABLE* sisendisse saabub signaal kontrolleri kaitselülitusest, mis takistab võre pööramist kaugemale lubatud lainepikkuste vahemiku piiridest. Madala nivoo korral



Joonis 13: Samm-mootori juhtsignaalide aegdiagramm.

sellel sisendil on vältus-impulssmodulaatori väljunditel samuti madal nivoo, võtmetransistorid on suletud ja mootori mähiseid vool ei läbi.

Modulaatori taktgeneraatori sagedus on kasulik valida suurem helisagedusest, et vähendada mootori töötamisel tekkivat müra. L6506 kasutab RC generaatorit (R1 ja C1 joonisel 23 lk 47), mille sagedus on määratud valemiga [15]

$$f = \frac{1}{0,69RC}\,. \tag{14}$$

Antud juhul $R = 18\,\mathrm{k}\Omega$ ning $C = 1,5\,\mathrm{nF}$ ja seega generaatori sagedus peaks olema

$$f = \frac{1}{0,69 \cdot 18 \cdot 10^3 \cdot 1,5 \cdot 10^{-9}} = 53676,9\ \text{Hz}. \tag{15}$$

Kuna reaalselt komponendid ei vasta täpselt neile trükitud nominaalväärtustele, siis tegelikult osutus sagedus olevat arvutatust mõnevõrra väiksem, jäädes pisut alla 40 kHz. See on siiski oluliselt suurem helisagedusest ja kuna täpse ja stabiilse sageduse tagamine pole oluline, siis ei ole ka tarvidust kasutada selles skeemi osas täppiskomponente.

Samm-mootor tarbib suhteliselt suurt voolu ja spektromeetri ning kontrolleri vahelised ühendusjuhtmed on pikad. Seetõttu mootori töötamise ajal juhtmetes tekkiva pingelangu tõttu kontrolleri skeemi nullnivoo muutus spektromeetri skeemi nullnivoo suhtes mitme voldi ulatuses. Sellest tulenevalt hakkasid esinema häired samm-mootori töös, kuna vältus-impulssmodulaatori juhtsignaalid saabuvad kontrollerist ja on fikseeritud kontrolleri skeemi nullnivoo suhtes. Nende häirete vähendamiseks on kõikidele kontrollerist saabuvatele juhtsignaalidele enne vältus-impulssmodulaatorit vahele lisatud RC-madalpääsfilter.

Suurt tähelepanu on pööratud protsessori kaitsmisele. Väljunditesse on ühendatud pool-

juhtdioodid vältimaks liigpinge sattumist protsessori väljundviikudele, sisendid on kaitstud stabilitronidega. Kõik protsessori sisendid ja väljundid, mille signaalid liiguvad pikkade ühendusjuhtmete kaudu kontrolleri ja spektromeetri vahel, on lisaks varustatud puhvritega, mis on välismõjude suhtes mikroprotsessorist vastupidavamad ja riknemise korral lihtsalt ning odavalt asendatavad. Skeemi toiteahelasse on ühendatud stabilitron (D13 joonisel 22 lk 46), mis liigse pinge lühistab ja suurenenud voolu tõttu põletab läbi sulavkaitsme. Kaitse on ülikiire reageerimisajaga ja ette nähtud spetsiaalselt pooljuhtseadiste kaitsmiseks.

5.2 Spektromeetri kontrolleri tarkvara

Kontrolleri programm on kirjutatud assemblerkeeles. Kogu programm on üles ehitatud riistvaraliste ja tarkvaraliste katkestuste töötlemisele. Kui parajasti midagi teha ei ole tarvis, siis läheb kontrolleri protsessor energiasäästurežiimi ja väljub sellest olekust järgmise katkestuse peale. Kokku on kasutusel 10 erinevat katkestust genereerivat olukorda. Need on järgmised.

- Difraktsioonivõret pöörava mehhanismi jõudmine lainepikkuste skaala pikalainelise või lühilainelise piirajani.
- Spektromeetri juhtpaneelil asuvate difraktsioonivõre pööramist võimaldavate lülitite vajutamine (neid kahte katkestust on võimalik lubada või keelata juhtprogrammi konfiguratsioonifaili parameetriga *manualstep*, lisaks keelatakse need katkestused programmi poolt mõõtmise sooritamise ajaks).
- Jadaväratitesse andmete saabumine kas juhtarvutist või AD-muundurist.
- Jadaväratitest andmete juhtarvutile või AD-muundurile saatmise lõpp (katkestus genereeritakse siis, kui jadavärati saatepuhver on tühi ja sinna võib saatmiseks laadida järgmise andmebaidi).
- *Taimer1* katkestus. Selle katkestuse sees tehakse põhiline töö samm-mootori juhtimisel ja mõõtmisel. Taimer nullitakse, mootorile antakse korraldus sammu tegemiseks ja arvutatakse ning seatakse uus väärtus, milleni jõudes genereeritakse järgmine *taimer1* katkestus (vajalik kiirendusega liikumise võimaldamiseks). Kui on jõutud järgmise lugemi võtmiseks sobivale lainepikkusele, saadetakse AD-muundurile lugemi võtmise käsk.

- *Taimer2* katkestus. Siin võetakse samm-mootori mähistelt pinge maha. See on vajalik energia säästmiseks difraktsioonivõre aeglase pööramise korral, kuna püsimagnetitega samm-mootor ühes asendis seismiseks elektrit ei vaja. Sammuimpulsside sagedustel üle 200 Hz seda katkestust ei kasutata, sest *taimer2* katkestus genereeritakse 5 ms pärast taimeri nullimist, aga nullitakse ta iga *taimer1* katkestuse töötlemise ajal.

Kontrolleril on kaks jadaväratit AD-muunduriga ja juhtarvutiga suhtlemiseks. Kuivõrd juhtarvuti otse AD-muunduriga ühendatud ei ole, siis kogu AD-muunduriga suhtlemine käib kontrolleri vahendusel. Initsialiseerimisel ja üksikute lugemite võtmisel toimib kontroller AD-muunduri ja arvuti vahel läbipaistva lüüsina, spektri mõõtmise ajal kontrollib seadme tööd täielikult kontrolleri programm. Kontrollerisse jadaväratite kaudu sisenevat infot käsitletakse erinevalt sõltuvalt sellest, et kas ta saabus AD-muundurist või juhtarvutist. Kui parajasti mõõtmist ei toimu, siis kõik AD-muundurist saabunud baidid saadetakse teise jadavärati kaudu otse edasi juhtarvutile. Mõõtmise sooritamise ajal saadetakse edasi ainult mõõtmistulemused. Enne mõõtmistulemuste saatmist tagastab AD-muundur käsukoodi 0x81 (vt. joonis 10), seda baiti juhtarvutile edasi ei saadeta. Juhtarvutist kontrollerile saadetud infot käsitletakse sõltuvalt jadavärati juhtsignaali RTS olekust. Kui juhtarvuti on seadnud selle signaali nulliks, siis saadetakse kõik saabuvad baidid teise jadavärati kaudu edasi AD-muundurile. RTS signaali kõrge nivoo korral toimitakse vastavalt saadud käsukoodile, mida on kokku 15 erinevat.

0x00 kajatest. Sellele järgnev bait saadetakse tagasi juhtarvutile, et oleks võimalik kontrollida andmeside kvaliteeti.

0x01 difraktsioonivõre pööramise käsk. Järgnevad kaks baiti määravad ära, kui palju tuleb võret pöörata (samm-mootori sammude arv). Kõrgem bait saadetakse esimesena. Kui mõlemad baidid on nullid, siis pööratakse difraktsioonivõret kuni piirajani. Võret pööratakse samas suunas, kuhu ta viimati liikus, kui ei ole eelnevalt suunda muudetud käskudega 0x03 või 0x04. Soovitud lainepikkusele jõudes tagastatakse juhtarvutile käsukood 0x01. Kui jõutakse piirajani, siis tagastatakse käsukoodi asemel 0xF0, kui jõuti lühemalainelise piirajani, või 0xF1, kui jõuti pikemalainelise piirajani.

0x02 difraktsioonivõret pöörava mehhanismi seiskamise käsk. Samm-mootor peatatakse, juhtarvutile saadetavate andmete puhver tühjendatakse ja juhtarvutile saadetakse neli korda käsukood 0x02.

0x03 seab difraktsioonivõre pööramise suunaks lühematelt lainepikkustelt pikemate poole.

0x04 seab difraktsioonivõre pööramise suunaks pikematelt lainepikkustelt lühemate poole.

0x05 spektromeetri mingile kindlale lainepikkusele häälestamise käsk. Järgnevad kaks baiti määravad ära, millisele lainepikkusele tuleb difraktsioonivõre pöörata (samm-mootori sammude arv). Kõrgem bait saadetakse esimesena. Soovitud lainepikkusele jõudes tagastatakse juhtarvutile käsukood 0x05.

0x06 muudab juhtarvutiga ühendatud jadavärati ühenduskiirust. Järgnev kahebaidine arv määrab ära kiiruse, ta leitakse vastavalt valemile

$$S = \frac{F_{OSC}}{16b - 1}\,, \tag{16}$$

kus $F_{OSC} = 14745600$ Hz on protsessori taktsagedus ja b on soovitav kiirus boodides. Kõrgem bait saadetakse esimesena.

0x07 muudab AD-muunduriga ühendatud jadavärati ühenduskiirust. Järgnev kahebaidine arv määrab ära kiiruse, ta leitakse vastavalt valemile (16). Kõrgem bait saadetakse esimesena.

0x08 muudab difraktsioonivõre pööramisega seotud parameetreid. Parameetriteks on 11 baiti, mis tähendavad järgmist.

- Kahebaidine difraktsioonivõre pööramise algkiirus mõõtmise ajal. Tegu on arvuga, milleni loendades genereeritakse *taimer1* katkestus. Ta leitakse vastavalt valemile
$$C = \frac{F_{OSC}}{64f}\,, \tag{17}$$
kus $F_{OSC} = 14745600$ Hz on protsessori taktsagedus ja f on soovitav samm-mootori sammuimpulsside sagedus hertsides.
- Kahebaidine difraktsioonivõre pööramise maksimaalkiirus mõõtmise ajal. Leitakse vastavalt valemile (17).
- Kahebaidine difraktsioonivõre pööramise algkiirus, kui mõõtmist ei toimu (näiteks spektromeetri häälestamisel mingile lainepikkusele). Leitakse vastavalt valemile (17).
- Kahebaidine difraktsioonivõre pööramise maksimaalkiirus, kui mõõtmist ei toimu. Leitakse vastavalt valemile (17).

- Difraktsioonivõre pööramise kiirendus. Selle suuruse võrra vähendatakse iga mootori sammu järel väärtust, milleni jõudes genereeritakse *taimer1* katkestus. Kiirendusega liikumine toimub seni, kuni saavutatakse maksimaalne lubatud kiirus. Enne peatumist võre pööramise kiirust uuesti vähendatakse, nii et soovitud peatumise kohani jõudes oleks saavutatud jälle algkiirus.
- Diskreetimisintervall. Määrab ära samm-mootori sammude arvu, mille möödudes lugemit võetakse. Üks samm vastab lainepikkuse muudule ligikaudu 0,2 nm.
- Viimane bait määrab selle, kas spektromeetri juhtpaneelil asuvate nuppude abil saab difraktsioonivõret pöörata. Kui selle baidi väärtus on 0, siis juhtpaneeli nuppude vajutamisel genereeritavad katkestused keelatakse ja nende nuppude abil difraktsioonivõret pöörata ei saa.

Kõigil eelmainitud juhtudel saadetakse kahebaidise arvu korral kõrgem bait esimesena.

0x09 mõõtmise alustamine. Järgnevad kaks baiti määravad ära, millise lainepikkuseni tuleb skaneerida (samm-mootori sammude arv). Kõrgem bait saadetakse esimesena. Arvutile tagastatakse käsukood 0x09 ja mõõtmistulemused.

0x0A lainepikkuste skaala initsialiseerimine. Difraktsioonivõre pööratakse mootori sammuimpulsside sagedusega 100 Hz lühilainelise piirajani ja seatakse samm-mootori sammude arvu loenduri väärtuseks 10 (nullist suurem, kuna piiraja asukoht on kontaktide konstruktsioonist tulenevalt määratud mõne sammu täpsusega ja loenduri väärtus ei tohi ühelgi juhul muutuda negatiivseks). Juhtarvutile tagastatakse bait väärtusega 0xF0.

0x0B käsule järgnev bait näitab AD-muundurilt saadavate mõõtmistulemuste baitide arvu. See on vajalik AD-muundurilt saadavate andmete tõlgendamiseks ja käsukoodi 0x81 ärajätmiseks mõõtmise sooritamise ajal juhtarvutile saadetavate mõõtmistulemuste hulgast.

0x0C tagastab juhtarvutile samm-mootori sammude arvu loenduri väärtuse. Selle järgi saab teada lainepikkuse, millele spektromeeter häälestatud on.

0x0D mõõdab AD-muundurilt lugemi saamiseks kulunud aja, saadud tulemus saadetakse, kõrgem bait esimesena, juhtarvutile. Selle järgi on võimalik määrata mak-

simaalset lugemite võtmise sagedust hertsides vastavalt valemile

$$f_{max} = \frac{F_{OSC}}{256t}\,, \tag{18}$$

kus $F_{OSC} = 14745600$ Hz on protsessori taktsagedus ja t on juhtarvutile tagastatud tulemus.

0xA0 lähtestamiskäsk. Programmi tööd alustatakse uuesti algusest.

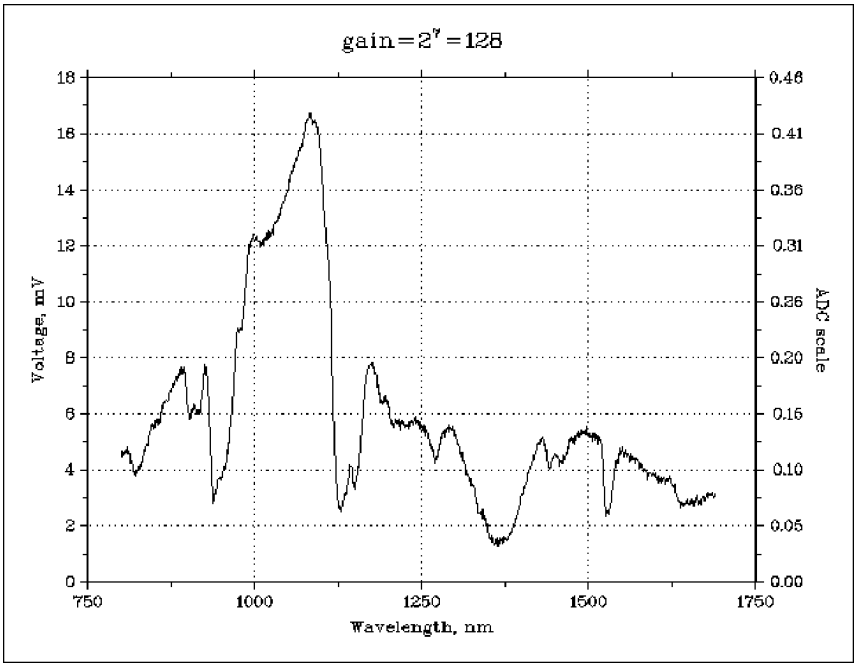
Maksimaalse kiiruse saavutamiseks toimub lugemite võtmine difraktsioonivõre pööramisega samaaegselt. Kontroller suhtleb AD-muunduriga kiirusel 300-9600 boodi sekundis. Enne lugemi võtmist saadab kontroller üle jadaliidese kolm baiti AD-muundurile ja saab ühe baidi tagasi. Selleks kulub teatud aeg δt (vt. joonis 10). Seetõttu on reaalseks lugemi võtmise hetkeks juba mõned sammud edasi astutud. Lihtsuse ja paindlikkuse mõttes tehakse selle ja ka teiste mõõtmisel tekkivate vigade (lainepikkuste skaala erinevus ideaalsest autokollimatsioonilisest süsteemist, viivitus signaaliahela analoogosas) korrektsioon spektromeetri juhtarvuti tarkvaras, mitte ei korrigeerita kontrolleri programmis AD-muundurile lugemi võtmise käsu saatmise ajahetke.

Kontroller võib juhtarvutiga ja AD-muunduriga suhelda erinevatel kiirustel. Seetõttu on kasutusel elavjärjekorra põhimõttel (*FIFO - first in, first out*) töötav puhver mõlemas suunas andmete saatmisel, et läbipaistva lüüsina töötades andmeid kaduma ei läheks kiiremalt seadmelt aeglasemale saates. Lisaks toimub puhvrisse andmete salvestamine kiiresti ja ei teki viivitust programmi töös andmete saatmisel suhteliselt aeglase jadaliidese kaudu. Kuna AD-muunduri suunas väga mahukaid andmeid ei saadeta, siis sellega suheldakse läbi 15-baidise puhvri. Juhtarvuti suunas liiguvad mahukad mõõtmistulemused, millest tulenevalt on selle puhver tehtud nii suureks kui vähegi võimalik ehk veidi alla nelja kilobaidi.

Kontrolleri programmis on mõned lihtsamad veaolukordade kontrollid (juhtarvutiga ühendatud jadavärati saatepuhvri ületäitumise kontroll - veakood 0x10, AD-muunduriga andmeside kiiruse kontroll - veakood 0x20). Vea ilmnedes antakse sellest märku spektromeetri juhtpaneelil asuva valgusdioodi vilgutamisega ja spektromeeter viiakse avariirežiimi - arvutiga suhtlemise kiirus seatakse 300 boodi peale, arvutile saadetakse pidevalt veakoodi ja oodatakse lähtestamiskäsku.

5.3 Spektromeetri juhtarvuti tarkvara

Spektromeetri juhtarvuti programm on kirjutatud *GNU/Linux* platvormile keeles *C++*, mõõtmistulemuste visualiseerimiseks on kasutusel joonestusteek *Dislin*. Programm



Joonis 14: Spektromeetri juhtprogrammi mõõtmistulemuse kuvamise akna ekraanipilt.

võimaldab mõõta kogu spektri või ainult osa sellest, sammuda soovitud lainepikkusele ja võtta ühekordselt või korduvalt lugemeid ühel lainepikkusel. Kasutajaliides on kirjutatud tekstirežiimis kasutamiseks, kuid graafilises keskkonnas töötamise korral on võimalik näidata mõõtmistulemust ekraanil graafikuna (vt. joonis 14), lisaks salvestatakse see tabuleeritud kujul kõvakettale. Programmi konfiguratsioonifailis (vt. Lisa B) on võimalik muuta diskreetimisintervalli ja skaneerimise kiirust. Kuna samm-mootor ei ole kuigi võimekas suure dünaamilise koormuse korral ja kiiruse suurenedes pöördemoment väheneb, siis sammude vahelejätmise vältimiseks toimub sammumise alustamine ja peatumine sujuvalt. Kiirendust kasutamata on maksimaalne sammuimpulsside sagedus, mille juures mootor veel töötada suudab, umbes 500 Hz. Võimalik on eraldi määrata nii algkiirust kui kiirendust mõõtmise ja ühelt lainepikkuselt teisele sammumise jaoks. Ka AD-muunduri tööparameetreid (sisendsignaali võimendus, bittide arv) saab konfiguratsioonifailis seada.

Spektromeetri kontrolleriga suhtleb juhtarvuti jadavärati abil. Kuna tänapäevastel sülearvutitel tihti jadavärat puudub, siis on läbi viidud katsed suhtlemiseks kontrolleriga USB-RS232 muunduri vahendusel. Juhtarvuti poolt vaadates paistab lõppkasutajale selline seade tavalise jadaväratina. Kasutusel oli MCT toodetud muundur U232-P9. Paraku selgus, et selle muunduri draiver 2.6 seeria tuumale on vigane. Kontrolleri poolt juhtarvutisse siseneva andmevoo esimene bait ei jõudnud kohale, samuti ei töötanud mõned

olulised jadavärati kontrollfunktsioonid (*tcdrain(), tcflush(), tcflow()*). Draiveri autorit on vigadest teavitatud ja ta lubas nendega ka esimesel võimalusel tegeleda. Vajaduse korral, tehes mõningaid muudatusi juhtarvuti programmi jadaväratiga suhtlemise osas, on selle muunduri kasutamine siiski võimalik. Hetkel on need muudatused programmi koosseisu kaasamata, kuna tegu ei ole korrektse lahendusega ja on lootus, et muunduri draiver operatsioonisüsteemi tuuma järgnevates versioonides parandatakse.

Mõõtes kitsast spektrijoont kahes suunas skaneerides selgus, et mida suurem oli skaneerimise kiirus, seda enam spektrijoone asukoht nihkus. Selle põhjuseks on signaali viivitus signaalitrakti analoogosas. Lisaks tekib lainepikkuste skaalas viga, kuna ei ole täidetud autokollimatsioonitingimus. Eelmainitud vigasid on püütud spektromeetri juhtprogrammis mõõtmistulemuste registreerimisel arvesse võtta ja vastavalt lainepikkuste skaalat korrigeerida.

Spektromeetri kontroller arvestab difraktsioonivõre asendit loendades samm-mootori samme arvestusega, et tegu on ideaalse autokollimatsioonilise süsteemiga ning ühele sammule vastab lainepikkuse muutus 0,2 nm. Kasutajaliideses on ühikuks nanomeetrid. Kahe taustsüsteemi vahelistel teisendustel võetakse arvesse lainepikkuste skaala kõrvalekallet ideaalsest autokollimatsioonilisest süsteemist ja siinusmehhanismi hoova a (vt. joonis 6 lk 15) pikkuse erinevust planeeritust. Üleminekuks samm-mootori sammudelt nanomeetritele avaldatakse valemist (7) suurus s ja asendatakse see valemisse (9). Reaalse lainepikkuse saab siis leida valemist

$$\lambda = \frac{\lambda_0 a_0}{2\left(a_0 + \Delta a\right)} + d \sin\left(\arcsin\left(\frac{\lambda_0 a_0}{2d\left(a_0 + \Delta a\right)}\right) + \varepsilon\right) + \Delta\lambda, \tag{19}$$

kus λ_0 on lainepikkus ideaalse autokollimatsiooni korral (ühele sammule vastab lainepikkuse muut 0,2 nm), $a_0 = 41,67$ mm on hoova a planeeritud pikkus, Δa on selle hoova pikkuse erinevus planeeritust, $d = 1,667 \cdot 10^{-6}$ m on võrekonstant, $\varepsilon = 2,4907°$ on kõrvalekalle ideaalsest autokollimatsioonist ja $\Delta\lambda$ on lainepikkuste skaala nihe, millega viiakse ideaalne ja reaalne skaala kohakuti lainepikkusel 1265,6 nm. Üleminekul nanomeetritelt samm-mootori sammudele avaldatakse valemist

$$\lambda = d\left(\frac{s}{a + \Delta a} + \sin\left(\arcsin\left(\frac{s}{a + \Delta a}\right) + \varepsilon\right)\right) + \Delta\lambda \tag{20}$$

suurus s. Selleks võib teha muutujavahetuse

$$x = \frac{s}{a_0 + \Delta a} \tag{21}$$

ning lahendiks saadakse sel juhul

$$x = \frac{\lambda - \Delta\lambda}{2d} - \sqrt{\left(\frac{\lambda - \Delta\lambda}{2d}\right)^2 - \frac{1}{2\left(1 + \cos\varepsilon\right)}\left(\left(\frac{\lambda - \Delta\lambda}{d}\right)^2 - \left(\sin\varepsilon\right)^2\right)}. \qquad (22)$$

Seejärel tuleb asendada leitud s valemisse (7). Δa ja $\Delta\lambda$ on määratud katseliselt, nende määramisest on täpsemalt kirjutatud peatükis 6.1.

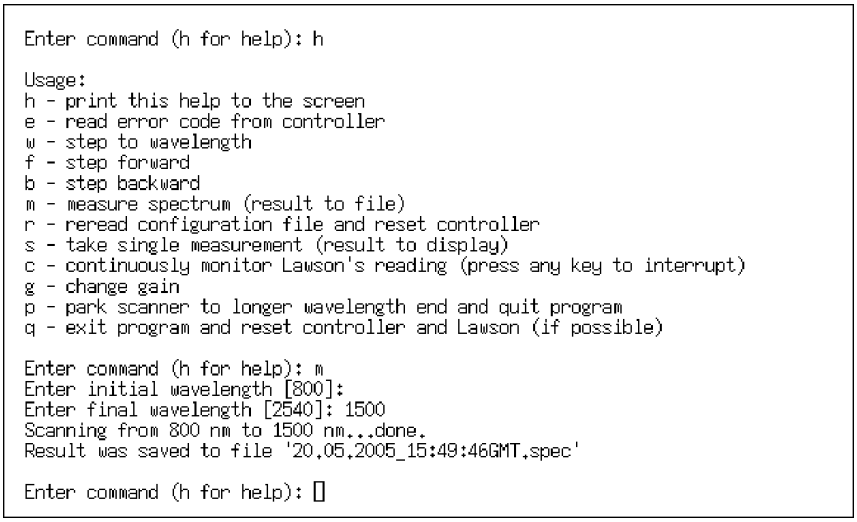
Kasutatav AD-muundur on ette nähtud suhteliselt aeglaste mõõtmiste sooritamiseks. Tema efektiivne lahutusvõime sõltub kasutatavast võimendusest ja efektiivsest diskreetimissagedusest. Efektiivse diskreetimissageduse suurenedes efektiivne lahutusvõime kahaneb, olles spetsifikatsiooni järgi ühikulise võimenduse juures 10 Hz efektiivse diskreetimissageduse korral 22,5 bitti, aga 250 Hz korral 15 bitti ja 1 kHz korral vaid 10,5 bitti[12]. Seetõttu on oluline kasutada võimalikult madalat efektiivset diskreetimissagedust. Teiselt poolt tuleb mõõtmised sooritada võimalikult kiiresti, kuna lugemeid võetakse difraktsioonivõre pööramisega samaaegselt. Kompromissina on võetud kasutusele lahendus, et muundamise aja jooksul ei pööratakse difraktsioonivõret rohkem kui 20% diskreetimisintervallist, mis tähendab, et 5 nm diskreetimisintervalli ja 1500 Hz sammuimpulsside sageduse korral kasutatakse AD-muunduri efektiivset diskreetimissagedust 300 Hz. Kasutatava efektiivse diskreetimissageduse arvutab spektromeetri juhtarvuti programm automaatselt valitud diskreetimisintervalli ja mootori sammuimpulsside sageduse järgi.

Programmi käivitamisel toimub kõigepealt kontrolleri ja AD-muunduri initsialiseerimine. Selle käigus seatakse andmesidekiirused, tehakse kajatest, seatakse sammuimpulsside sagedused, diskreetimisintervall jms parameetrid. Seejärel viiakse läbi AD-muunduri kalibreerimine.

Väikese diskreetimisintervalli korral piirab skaneerimise kiiruse andmevahetuse kiirus AD-muunduri ja kontrolleri vahel. Initsialiseerimise ajal käsib kontroller AD-muunduril võtta ühe lugemi ja mõõdab ära selle lugemi saamiseks kulunud aja. See tagastatakse juhtarvutile, mis leiab antud seadete juures maksimaalse võimaliku skaneerimise kiiruse. Kui kasutaja on valinud suurema kiiruse, siis antakse sellest teada ja seatakse kiirus väiksemaks. Kindluse mõttes valitakse selleks maksimaalsest võimalikust ühe protsendi võrra väiksem kiirus.

Peale initsialiseerimise lõppu jääb programm ootama kasutaja edasisi korraldusi. Kõik käsud on ühetähelised ja neid saab vaadata trükkides 'h' (*help*). Joonisel 15 on näha programmi abimenüü ekraanipilt. Järgnevalt on kirjeldatud kõiki käske lähemalt.

e loeb kontrollerist veakoodi. Kui spektromeetri kontroller on tuvastanud veasituatsioo-



Joonis 15: Spektromeetri juhtprogrammi abimenüü ekraanipilt.

ni, siis läheb ta avariirežiimi. Samm-mootor seisatakse, umbes 20 Hz sagedusega vilgutatakse spektromeetri juhtpaneelil asuvat valgusdioodi, andmevahetuskiirus seatakse 300 boodi peale ja saadetakse pidevalt juhtarvutile veakoodi ning kontrollitakse, ega juhtarvuti poolt pole vastu saadetud lähtestamiskäsku. Tuvastatavad veasituatsioonid on juhtarvutiga suhtleva jadavärati saatepuhvri ületäitumine ja AD-muunduri jaoks liiga kiire spektri skaneerimine, mis väljendub selles, et üritatakse võtta uut lugemit enne eelmise mõõtmise lõppemist.

w häälestab spektromeetri soovitud lainepikkusele. Selle käsu peale küsitakse kasutajalt lainepikkust, millele difraktsioonivõre pöörata. Vaikimisi valikuks pakutakse minimaalset lainepikkust (800 nm). Sisestada lubatakse vaid numbreid ja kui sisestatud arv jääb väljapoole võimalikku lainepikkuste skaalat, siis teavitatakse kasutajat vigasest sisestusest. *Escape* nuppu vajutades on võimalik võre pööramine katkestada.

f pöörab difraktsioonivõret pikemate lainepikkuste suunas. Kasutajalt küsitakse, mitu nanomeetrit tuleb edasi liikuda. Sisestada on võimalik vaid numbreid ja kui sisestatakse liiga suur number, siis antakse sellest kasutajale teada. *Escape* nuppu vajutades on võimalik liikumine katkestada. Vaikimisi valikuks pakutakse väärtuseks 0 nm. Kui on valitud programmi konfiguratsiooniparameeter *manualstep=1*, siis tähendab see, et tuleb liikuda kuni lainepikkuste skaala piirajani.

b sama, mis eelmine, kuid difraktsioonivõret pööratakse lühemate lainepikkuste suunas.

m mõõdab spektri. Kasutajalt küsitakse spektrivahemikku, mida mõõta. Lubatud on vaid numbrite sisestamine, samuti kontrollitakse lainepikkuste sobivust. Mõõtmis-

tulemus salvestatakse tabuleeritud kujul arvuti kõvakettale faili, mille nimekuju on kuupäev_kellaaeg.spec (näiteks *30.09.2004_12:18:19GMT.spec*). Lisaks mõõtmistulemustele on selles failis kirjas ka olulisemad mõõtmisel kasutatud seadme parameetrid nagu andmevahetuskiirus, skaneerimise kiirus, AD-muunduri konfiguratsiooniparameetrid ja skaneerimise suund. Nendele parameetritele on kõigile ette lisatud sümbol "#", sest paljud programmid käsitlevad selle sümboliga algavaid ridasid kommentaaridena ja sel juhul saab ilma faili muutmata kasutada teda kohe andmetöötluses. Juhul, kui konfiguratsioonifailis on seatud parameeter *showgraph = 1*, siis näidatakse kohe peale mõõtmise lõppu tulemust ka ekraanil graafikuna. Graafikul on kirjas kasutatav võimendus ja sekundaarsel ordinaatteljel on esitatud lugem AD-muunduri skaalas. Kui maksimaalne lugem jääb sellel skaalal alla 0,5, siis võib signaali võimendust vähemalt kaks korda tõsta. Kui mõõtmise ajal võetakse maksimaalne võimalik lugem, siis antakse sellest märku helisignaaliga. *Escape* nuppu vajutades on võimalik sammumine katkestada, määrata seejärel nõrgem võimendus ning mõõtmist korrata. Vältimaks keskkonnamõjude (temperatuur, niiskus) muutusest tekkivaid vigasid, viiakse enne iga mõõtmise algust läbi AD-muunduri kalibreerimine. Mehaanilistest lõtkudest tingitud vea ärahoidmiseks liigutakse valitud mõõtepiirkonna algusesse samas suunas, kui toimub edasine spektri skaneerimine.

r loeb uuesti konfiguratsioonifaili ja saadab spektromeetri kontrollerile lähtestamiskäsu. See on sisuliselt samaväärne spektromeetri juhtprogrammi sulgemise ja uuesti avamisega. Seda käsku on tarvis juhul, kui spektromeetri kontroller tuvastab veasituatsiooni ja läheb avariirežiimi või osutub sammumise kiirus, diskreetimisintervall või mõni muu konfiguratsioonifailis seatav parameeter ebasobivaks.

s võtab AD-muundurilt lugemi ja näitab ekraanile selle lugemi ning lainepikkuse, millele on difraktsioonivõre hetkel pööratud.

c pidevalt võtab AD-muundurilt lugemeid ja näitab neid ekraanile. See käsk on kasulik näiteks laseri suunamisel sisendpilule. Lugemite võtmise katkestab suvalise klahvi vajutus.

g muudab AD-muunduri sisendsignaali võimendust. Kasutajalt küsitakse uut signaali võimendustegurit kujul 2^g, st. kui kasutaja sisestab 7, siis seatakse võimenduseks $2^7 = 128$ korda. Muutust ei salvestata konfiguratsioonifaili. Sisestada lubatakse vaid numbreid ja kui valik ei lange lubatud piirkonda (0...7), siis teavitatakse sellest kasutajat.

p pöörab difraktsioonivõre lainepikkuste skaala pikemalainelise serva piirajani ja sulgeb programmi. Sellega on difraktsioonivõre asend kindlalt fikseeritud ja seadme transpordil ei hakka ta liikuma. Võimaluse korral (kui spektromeetri kontroller on kontaktivõimeline) lähtestatakse kontroller ja AD-muundur, et programmi uuesti käivitamisel oleksid nad jälle kontaktsed (andmevahetuse kiirus seatakse 300 boodi peale).

q sama, mis eelmine, kuid programmist väljutakse ilma difraktsioonivõret piirajani pööramata.

5.4 Spektromeetri juhtarvuti programmi konfiguratsioonifail

Järgnevalt on esitatud ülevaade spektromeetri juhtprogrammi konfiguratsioonifailis (vt. lisa B) seatavatest parameetritest.

port määrab ära jadavärati seadme asukoha failisüsteemis.

PC_baud spektromeetri kontrolleri ja juhtarvuti vahelise andmeside kiirus. Lubatud valikud on järgmised.

- 0 - 9600 boodi
- 1 - 4800 boodi
- 2 - 2400 boodi
- 3 - 1200 boodi
- 4 - 600 boodi
- 5 - 300 boodi

measfreq0 mootori sammuimpulsside sagedus hertsides mõõtmise alguses.

measfreq maksimaalne mootori sammuimpulsside sagedus hertsides mõõtmise ajal.

transpfreq0 mootori sammuimpulsside sagedus hertsides alustades spektromeetri häälestamist ühelt lainepikkuselt teisele, kui mõõtmist parajasti ei toimu.

transpfreq maksimaalne mootori sammuimpulsside sagedus hertsides spektromeetri häälestamisel ühelt lainepikkuselt teisele, kui mõõtmist parajasti ei toimu.

dstepsize difraktsioonivõre pööramise kiirendus ja aeglustus. Kehtib nii mõõtmise kui ühelt lainepikkuselt teisele häälestamise kohta. Üheselt määratavat

ühikut sellel parameetril pole, üsna mõistlik väärtus on umbes 8. Sammuimpulsside genereerimine on realiseeritud mikrokontrolleri loenduri katkestustega. Kiireneval liikumisel vähendatakse iga sammu järel katkestuseks vajalike kellataktide arvu parameetri *dstepsize* väärtuse võrra. Selline lahendus annab eksponentsiaalse kiirenduse. Aeglustusel vastavalt liidetakse iga sammu järel parameetri *dsetpsize* väärtus katkestuseks tarvilikule kellataktide arvule.

meassteps lugemite spektraalintervall. Määrab ära sammude arvu, mille möödudes lugemit võetakse. Üks samm vastab lainepikkuse muudule ligikaudu 0,2 nm. Lubatud on väärtused vahemikus 1-255, mis annab diskreetimisintervalli vahemikuks 0,2-51 nm.

channel AD-muunduri kanali valik. Fotodioodi signaalile vastab kanal 0, AD-muunduri sisemisele 5-voldisele etalonile vastab kanal 6 ja 0-voldisele etalonile kanal 7.

M201_baud spektromeetri kontrolleri ja AD-muunduri vahelise andmeside kiirus. Lubatud väärtused on samad, mis parameetri *PC_baud* korral.

gain AD-muunduri sisendsignaali võimendus. Esitatakse kujul 2^{gain}. Lubatud väärtuste vahemik on 0-7, mis annab võimendusteguri võimalikuks vahemikuks 1-128.

filter AD-muunduri sisemise esimest järku madalpääsfiltri lõikesagedus. Väärtus 0 vastab lõikesagedusele 4 Hz, 1 vastab lõikesagedusele 40 Hz ja 2 vastab lõikesagedusele 400 Hz.

wordcount AD-muunduri kasutatav bittide arv. Väärtus 2 vastab 16-bitisele eraldusvõimele ja 3 vastab 24-bitisele eraldusvõimele.

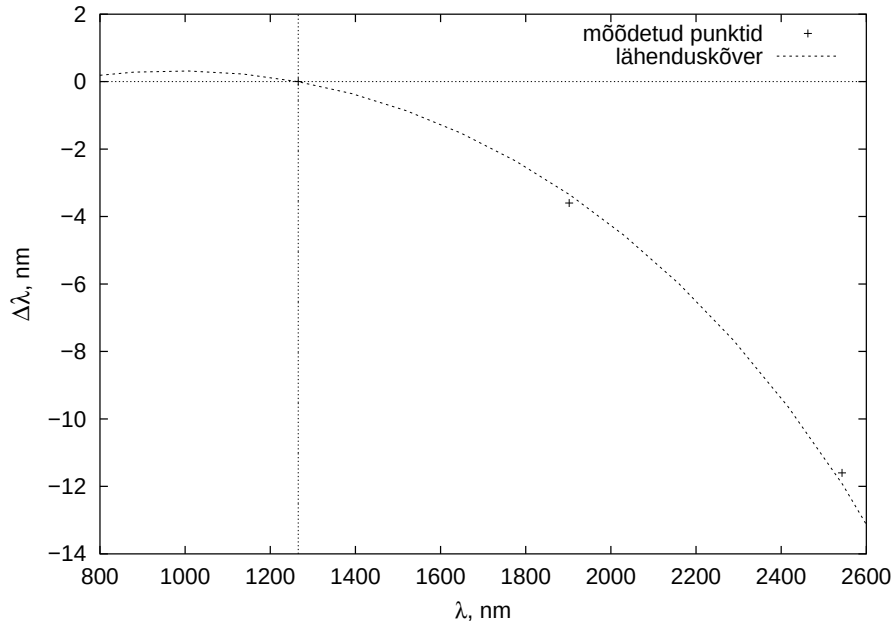
showgraph määrab ära, kas peale mõõtmist näidatakse ekraanil tulemust graafikuna. Väärtus 1 tähendab, et näidatakse, ja 0, et ei näidata.

manualstep määrab selle, kas spektromeetri juhtpaneelil asuvate nuppude abil saab difraktsioonivõret pöörata. 1 tähendab, et saab, ja 0, et ei saa. Lisaks keelatakse viimasel juhul ka programselt käskudega *b* ja *f* difraktsioonivõre pööramine kaugemale lubatud lainepikkuste vahemiku piiridest.

6 MÕÕTMISED JA TULEMUSTE ANALÜÜS

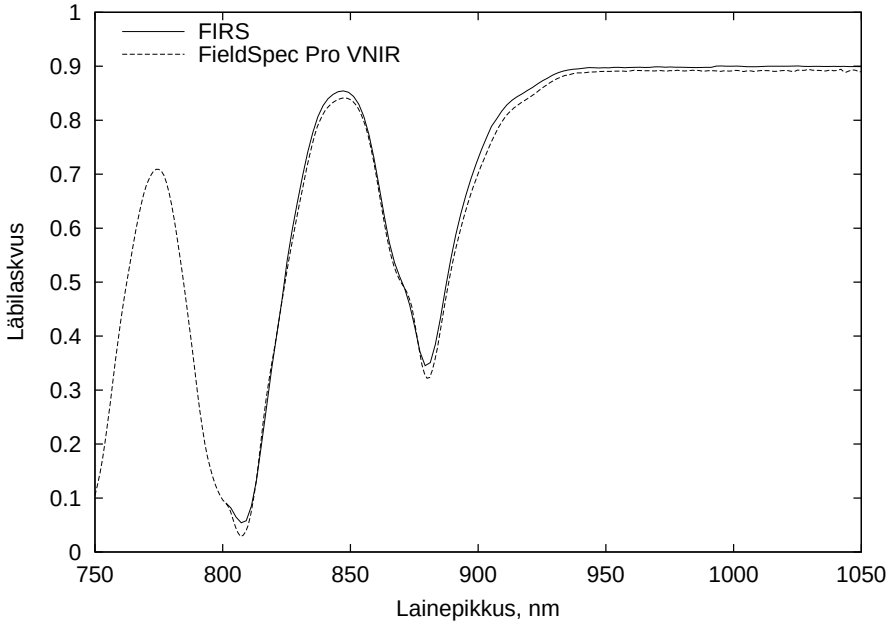
Spektromeetriga on läbi viidud esimesed proovimõõtmised laboritingimustes.

6.1 Optilise süsteemi erinevus autokollimatsioonilisest süsteemist. Lainepikkuste skaala parandi määramine



Joonis 16: Lainepikkuste skaala erinevus ideaalsest autokollimatsioonilisest süsteemist.

Spektromeetri optilise süsteemi erinevust ideaalsest autokollimatsioonilisest süsteemist ja siinusmehhanismi valmistamise täpsust saab hinnata, mõõtes kindla lainepikkusega spektrijoont. Selleks kasutati HeNe laseri valgust, mille lainepikkus on 632,8 nm. Kõrgemate spektrijärkude mõõtmiseks olid kiirte teelt eemaldatud järgueraldusfiltrid ja kõik mõõtmised viidi läbi lühemalainelise vastuvõtjaga, sest see on tundlikum antud lainepikkusega kiirgusele. Vaadeldi 2.-4. järku peamaksimumi asukohtasid, mis teoreetiliselt peaks asuma lainepikkustel 1265,6 nm, 1898,4 nm ja 2531,2 nm. Joonisel 16 on kujutatud lainepikkuste skaala viga sõltuvalt lainepikkusest. Reaalne skaala ja ideaalsele autokollimatsioonilisele süsteemile vastav skaala on kohakuti nihutatud HeNe laseri valguse teist järku peamaksimumi kohal ehk lainepikkusel 1265,6 nm, mistõttu selle koha peal on viga võrdne nulliga. Lähenduskõver vastab juhule, kus difraktsioonivõrele langenud ja sellelt difrageerunud vaadeldava kiire vaheline nurk on $2,4907°$ja siinusmehhanismi hoob a (vt. joonis 6 lk 15) on plaanitust 0,3 mm lühem ehk pikkusega 41,37 mm. Valemites (19)-(22)



Joonis 17: Klaasfiltri läbipaistvusspekter mõõdetuna spektromeetritega FIRS ja FieldSpec® Pro VNIR.

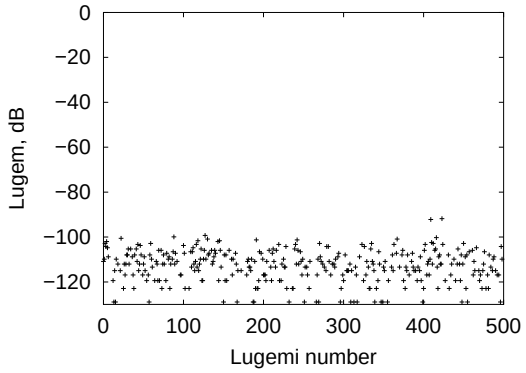
kasutatavaks lainepikkuste skaala nihkeks on määratud $\Delta\lambda = -75,5$ nm.

6.2 Klaasfiltri läbilaskvusspektri mõõtmine. Võrdlus spektromeetriga FieldSpec® Pro VNIR

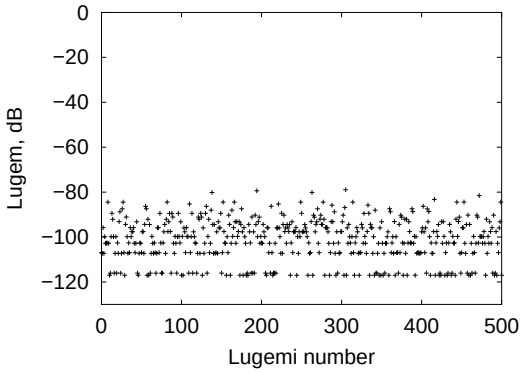
Testobjektiks valiti klaasfilter ΠC-7, millel on lainepikkuste vahemikus 800-1050 nm keeruka kujuga läbilaskvusspekter. Joonisel 17 on mõõdetud klaasfiltri läbilaskvus spektromeetriga FIRS ja võrdluseks sama filtri läbilaskvus mõõdetuna spektromeetriga FieldSpec® Pro VNIR firmalt Analytical Spectral Devices, Inc. Lainepikkuste vahemikuks on valitud kahe spektromeetri mõõtepiirkondade ühisosa. Määratud läbilaskvusspektrite väike erinevus võib olla tingitud spektromeetri sees hajunud ja peegeldunud kiirgusest, mis osaliselt jõuab ka vastuvõtjani. Selle vältimiseks tuleb vastuvõtja paigutada selliselt, et temani jõuaks vaid väljundpilu läbinud kiirgus. Ühtlasi tuleb spektromeetri sisemuses vältida heledaid ja peegeldavaid pindasid.

6.3 AD-muunduri kvaliteedi hinnang

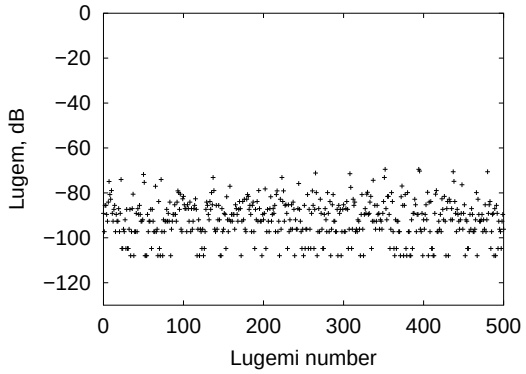
AD-muunduri müra mõõtmiseks kasutati AD-muunduri sisemist 0 V referentskanalit, igasugune nullist erinev lugem oli järelikult müra. Erineva efektiivse diskreetimissagedusega



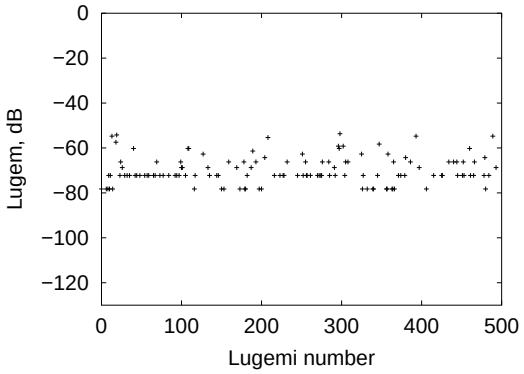
Joonis 18: AD-muunduri müra efektiivsel diskreetimissagedusel 100 Hz. Maksimaalsele võimalikule signaalile (5 V) vastab 0 dB.



Joonis 19: AD-muunduri müra efektiivsel diskreetimissagedusel 200 Hz. Maksimaalsele võimalikule signaalile (5 V) vastab 0 dB.



Joonis 20: AD-muunduri müra efektiivsel diskreetimissagedusel 300 Hz. Maksimaalsele võimalikule signaalile (5 V) vastab 0 dB.



Joonis 21: AD-muunduri müra efektiivsel diskreetimissagedusel 690 Hz. Maksimaalsele võimalikule signaalile (5 V) vastab 0 dB.

registreeriti 501 lugemit. Mõõtmistulemused on kujutatud joonistel 18 kuni 21. Signaali ja müra suure erinevuse tõttu on esitatud müra lugemid suhtena maksimaalsesse võimalikku signaali logaritmilises skaalas.

Erinevate mõõtmisviiside käigus selgus, et AD-muunduri müra on aditiivne, mis liitub AD7712 sisendisse jõudvale signaalile ja ei sõltu signaalist endast. Seetõttu on signaal-müra suhte parandamiseks vajalik, et muunduri sisendsignaal oleks võimalikult suur. Suuremate efektiivsete diskreetimissageduste korral ei sõltunud müranivoo muundamisel kasutatud bittide arvust, väikeste efektiivsete diskreetimissageduste korral jäi ta 16-bitise muundamise korral väiksemaks ühe kvandi väärtusest.

Saadud tulemustest järeldub, et üle 300 Hz efektiivsete diskreetimissageduste kasutamine pole soovitatav, sest siis jääb AD-muunduri efektiivne lahutusvõime alla 15 biti.

See on aga mõeldav vaid juhul, kui kompromissina kasutatakse lahendust, kus lugemi võtmise kestel difraktsioonivõret pööratakse, millest tulenevalt muutub ka lainepikkus. Spektromeetri projekteerimisülesannetest lähtuvalt on valitud lahenduseks olukord, kus ühe lugemi võtmise jooksul muutub lainepikkus 20% lugemite spektraalintervallist, mis moodustab 5 nm intervalli korral 10% nõutud spektraalsest lahutusvõimest. Maksimaalne kasutatav efektiivne diskreetimissagedus on sellise valiku korral 690 Hz, selle piirab ära AD-muunduri andmesidekiirus. Sel juhul on AD-muunduri efektiivne lahutusvõime umbes 11,5 bitti. Ühtlasi on tulemustest näha, et AD-muunduri 24-bitist režiimi on mõtet kasutada vaid väga aeglase mõõtmise korral.

7 SPEKTROMEETRI EDASINE TÄIUSTAMINE

Lõpetada tuleb spektromeetri optilise osa konstrueerimine, mõlemale vastuvõtjale tuleb paigaldada eraldi kvaliteetsetest madala müranivooga komponentidest koostatud vool-pinge-muundur.

Proovimõõtmised on näidanud, et samm-mootor põhjustab vähesel määral difraktsioonivõre vibreerimist ja seetõttu tekitab signaalis müra. Selle müra kõrvaldamiseks on plaanis lisada vool-pinge-muunduri järele kolmandat järku Besseli filter lõikesagedusega 200-300 Hz. See peaks vähendama võre vibreerimisest tingitud müra, aga ka vastuvõtja ja vool-pinge-muunduri müra. Veel tuleks lisada filtri järele muudetava võimendusteguriga pingevõimendi, et oleks võimalik erineva intensiivsusega valgustatuse tingimustes mõõtmisi läbi viia. Spektromeetris signaali võimendamine aitab ühtlasi vähendada pikas ühenduskaablis tekkiva müra mõju. Hetkel puuduvad ka pikalainelise vastuvõtja termoelektrilise jahutuse toide ja juhtelektroonika.

Ühtlasi on mõõtmised on näidanud, et vastuvõtjateni jõuab kiirgus, mis ei ole läbinud väljundpilu. Selle vältimiseks tuleb paigutada fotovastuvõtjate ümber tumedaid kiirgust neelavaid katteid, et vältida spektromeetri sisemuses peegeldunud ja hajunud valguse jõudmist vastuvõtjateni. Lisaks tuleb poleeritud terasest pilude konstruktsioonid katta võimalikult suures ulatuses kiirgust neelava materjaliga, et vähendada spektromeetri sisemuses hajunud kiirguse intensiivsust.

Spektromeetri eeldatav kaal pärast eelmainitud täiustuste sisseviimist on ligi 9 kg. Nii rasket mõõteriista on äärmiselt tülikas paigutada välimõõtmiste ajal objekti kohale. Seetõttu tuleb kiirguse sisend lahendada valguskaabliga. See kõrvaldaks ühtlasi kõik mõõdetava kiirguse polarisatsioonist tingitud mõõtmisvead, kuna valguskaabel ei säilita korduvate sisepeegelduste tõttu kiirguse polarisatsioonitasandit.

Magistritöö käigus valmistatud elektroonika ja tarkvara on kooskõlas hetkeks valminud spektromeetri riistvaraga. Peale loetletud täiustuste sisseviimist tuleb pisut muuta ka spektromeetri juhtprogrammi, võimaldamaks lugemite võtmist kahelt vastuvõtjalt, samuti ei ole teada signaaliahela analoogosas tekkiva viivituse pikkust. See ei ole siiski kuigi mahukas töö, kuna AD-muunduri sisendkanali valiku võimalus on juba programmis realiseeritud.

8 KOKKUVÕTE

Käesoleva magistritöö käigus lahendamist vajavaks ülesandeks oli Tartu Observatooriumi taimkatte kaugseire töörühmas valmistatava välispektromeetri FIRS juhtimiseks ja AD-muundurilt andmete lugemiseks vajaliku elektroonika ja vabavaralisel *Linux* platvormil töötava tarkvara väljatöötamine ning realiseerimine. Töö põhilised tulemused on järgmised.

- Valminud on mikroprotsessori ATmega128 baasil konstrueeritud spektromeetri kontroller, mille ülesandeks on difraktsioonivõret pöörava samm-mootori juhtimine ja AD-muundurilt Lawson M201 andmete lugemine. Kontrolleri tööd juhib assemblerkeeles kirjutatud programm.
- Samm-mootori juhtimine on lahendatud vältus-impulss-modulaatori L6506 baasil.
- Spektromeetri juhtimiseks on keeles C++ kirjutatud programm, mille kasutajaliides võimaldab ka tekstirežiimis töötamist. Graafilise töökeskkonna kasutamisel on mõõtmistulemus võimalik koheselt kuvada ekraanil graafiku kujul joonestusteegi *Dislin* vahendite abil.
- Mõõtmistulemused salvestatakse tabuleeritud kujul arvuti kõvakettale koos lisainformatsiooniga mõõtmise toimumise aja ja spektromeetri konfiguratsiooniparameetrite kohta.
- Laboritingimustes on läbi viidud esmased proovimõõtmised lainepikkuste skaala parandi arvutamiseks kasutatavate empiiriliste parameetrite määramiseks. Lisaks on püütud hinnata seadme kvaliteeti ja võrreldud teda tööstusliku spektromeetriga FieldSpec® Pro VNIR.
- Proovimõõtmiste tulemusi analüüsides on leitud vajakajäämised ning korrigeerimist vajavad sõlmed ja lahendused spektromeetri disainis ja konstruktsioonis.

Tänan Matti Pehki ja Ando Otsa, kelle juhendamisel valmis käesolev magistritöö. Samuti tänan Tartu Ülikooli keskkonnafüüsika instituudi töötajaid osutatud abi ja vahendite eest Lawson M201 AD-muunduriga seonduvate probleemide lahendamise osas.

Joel Kuusk

9 KASUTATUD KIRJANDUS

[1] *Hand Held Remote Sensing Instrumentation.*
http://www.ictinternational.com.au/appnotes/ICT702.htm

[2] *FieldSpec® Pro - Product Specifications.*
http://www.asdi.com/products_specifications-FSP.asp

[3] *FieldSpec® Pro. User's Guide.*
http://asnerlab.stanford.edu/dataccess/asnerlab/files/fsproman.pdf

[4] *GER 2600 Specifications.*
http://www.ger.com/2600.html

[5] *GER-Series Field Portable Spectroradiometers.*
http://www.spectrapartners.nl/manufacturers/ger/ger_page.htm

[6] Vermote, E. F., Tanré, D., Deuzé, J. L., Herman, M., Morcrette, J.-J. Second Simulation of the Satellite Signal in the Solar Spectrum, 6S: An Overview. *IEEE Transactions on Geoscience and Remote Sensing*, 1997, **35**, 3, 675-686.

[7] *Birch - Betula pendula.*
http://www.aai.ee/bgf/ger2600/birch.html

[8] *ASD FieldSpec® Pro System.*
http://fsf.nerc.ac.uk/instruments/asd_fieldspec.shtml

[9] Pedrotti, F. L. S.J., Pedrotti, L. S. *Introduction to Optics. Second Edition.* Prentice-Hall, Inc., 1993.

[10] Busch, K. W, Busch, M. A. *Multielement Detection Systems for Spectrochemical Analysis.* John Wiley & Sons, Inc., 1990.

[11] Fraden, J. *Handbook of Modern Sensors: Physics, Designs, and Applications. Second Edition.* New York, Springer-Verlag, 1996.

[12] *AD7712 Data Sheet.*
http://www.analog.com/UploadedFiles/Data_Sheets/399787711AD7712_f.pdf

[13] Saveljev, I. *Füüsika üldkursus 2*. Tallinn, Valgus, 1978.

[14] *Constant-Current Chopper Drive UPS Stepper-Motor Performance.* `http://www.powerdesigners.com/InfoWeb/design_center/Appnotes_Archive/1650.pdf`

[15] *L6506 Data Sheet.* `http://www.alltronics.com/download/L6506D.pdf`

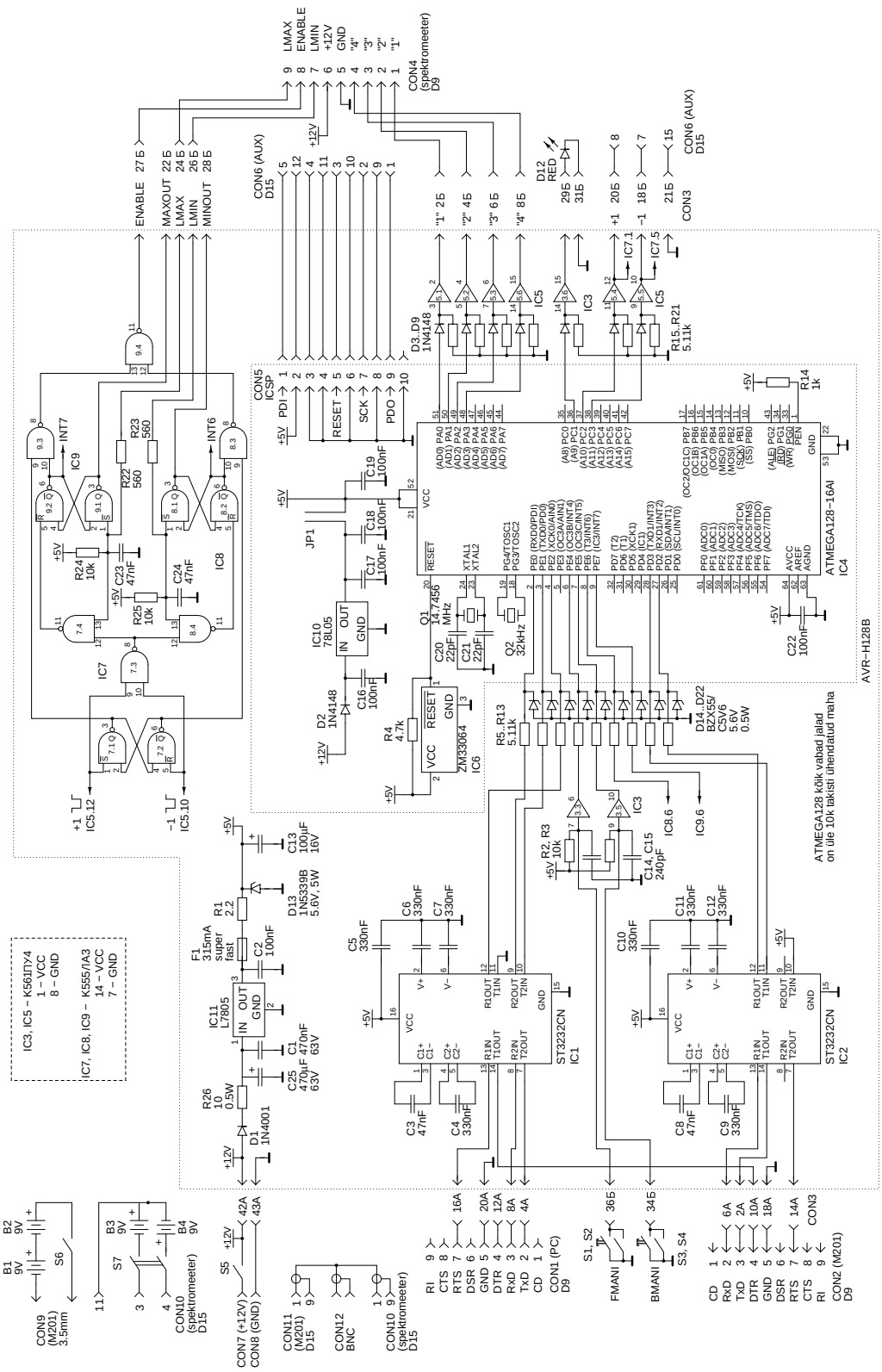
10 SUMMARY

Data Acquisition and Control of Field Spectrometer

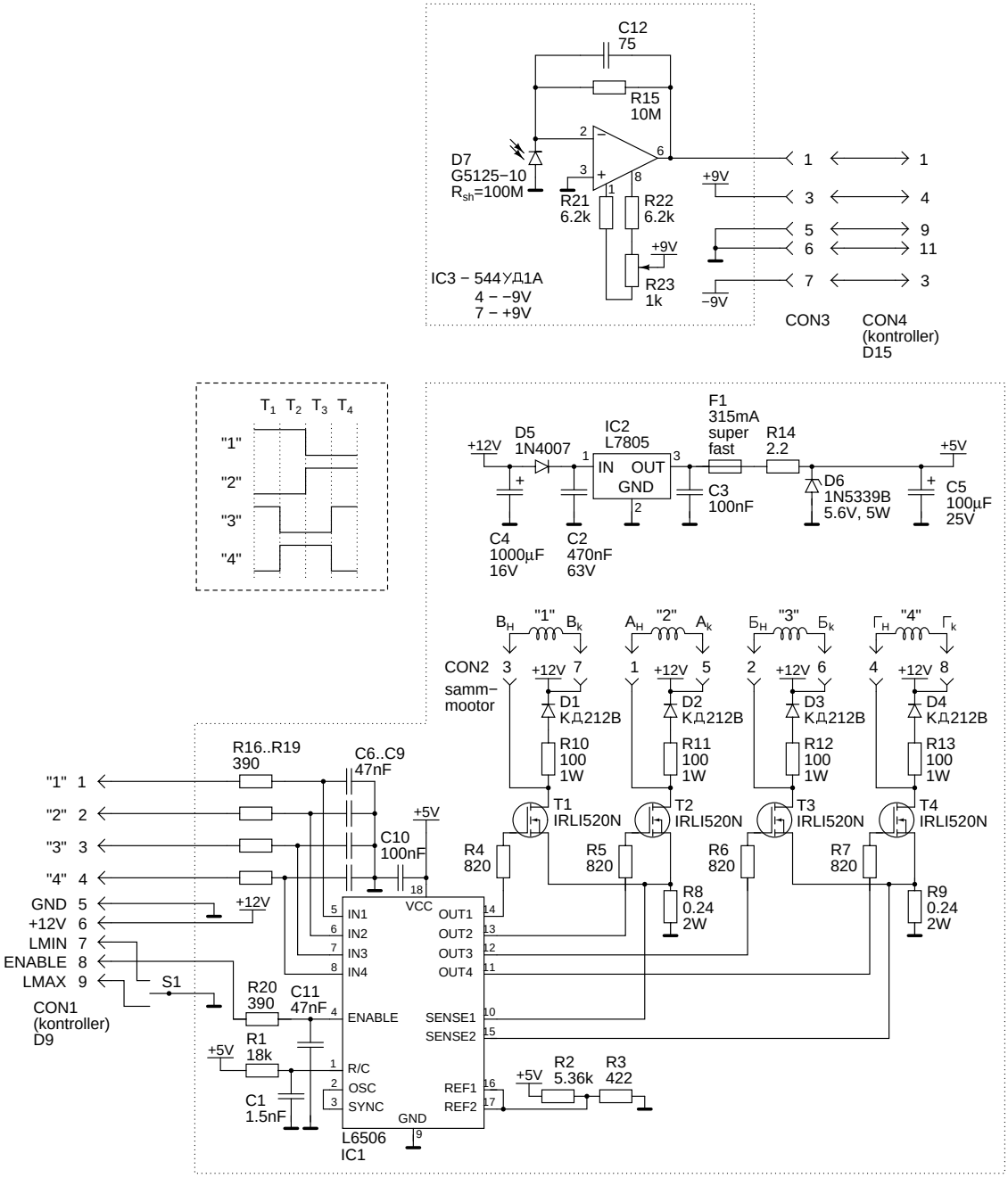
This work has been carried out at the Group of Vegetation Remote Sensing of the Tartu Observatory, where the field spectrometer FIRS (Field Infra Red Scanner) is under construction. The goal of the thesis was to develop electronics and *Linux*-based software for control of the spectrometer and for data acquisition from its analog-to-digital converter. The main results of the work are following.

- The construction of the ATmega128-based controller of the spectrometer is completed. Its task is to operate the stepper-motor which turns the diffraction grating and to read data from Lawson M201 ADC. The controller's program is written in assembly language.
- The operating of the stepper-motor is designed on the basis of pulse width modulator chip L6506.
- The spectrometer is operated by a computer program written in C++, which has a text-based user interface. In case of working in graphic desktop environment result of a measurement can be viewed at once by means of Dislin plotting library.
- The result of a measurement is written on the hard disk of the computer incorporated with additional information about the time of the measurement and the configuration parameters of the spectrometer.
- The first test measurements have been carried out at laboratory for calculating empirical parameters of the correction of the wavelength scale. An attempt is made to estimate the quality of the device and to compare it with the spectrometer FieldSpec® Pro VNIR manufactured by Analytical Spectral Devices, Inc.
- The test measurements have indicated shortcomings in the design and construction of the spectrometer which need further development.

Lisa A SPEKTROMEETRI SKEEMID



Joonis 22: Spektromeetri kontrolleri ja juhtpaneeli skeem.



Joonis 23: Spektromeetri skeem.

Lisa B SPEKTROMEETRI JUHTPROGRAMMI KONFIGURATSIOONI-FAIL

```
# port
port = /dev/ttyS1

# PC_baud
# PC baudrate
# 0 - 9600
# 1 - 4800
# 2 - 2400
# 3 - 1200
# 4 - 600
# 5 - 300
PC_baud = 0

# measfreq0
# initial scanning stepping frequency (Hz)
# reasonable value is somwhere around 250
measfreq0 = 250

# measfreq
# maximum scanning stepping frequency (Hz)
# reasonable value is somwhere below 1500
measfreq = 1000

# transpfreq0
# initial transport stepping frequency (Hz)
# reasonable value is somwhere around 250
transpfreq0 = 250

# transpfreq
# maximum transport stepping frequency (Hz)
# reasonable value is somwhere below 1500
transpfreq = 1300

# dstepsize
# stepping acceleration  coefficent
# affects transport and measuring speed
# reasonable value is somwhere around 8
dstepsize = 8

# meassteps
# number of steps between measurements (1 step = 0.2 nm)
meassteps = 25

# channel
# 0 - 800-1400 nm
# 1 - +5V
# 2 - 0V
# 6 - +5V internal reference
# 7 - 0V internal reference
channel = 0

# M201_baud
# Lawson’s baudrate
# 0 - 9600
# 1 - 4800
# 2 - 2400
# 3 - 1200
# 4 - 600
# 5 - 300
M201_baud = 0

# gain
# actual gain = 2^gain, gain = 0..7
```

```
gain = 2

# filter
# cut-off frequency of 1-st order analog filter
# 0 - 4 Hz
# 1 - 40 Hz
# 2 - 400 Hz
filter = 2

# wordcount
# Lawson’s word count
# 2 - 16 bitti
# 3 - 24 bitti
wordcount = 2

# showgraph
# 0 - don’t show graph after scan
# 1 - show graph after scan
showgraph = 1

# manualstep
# 0 - disable manual stepping (buttons on controller)
# 1 - enable manual stepping
manualstep = 1
```

Lisa C SPEKTROMEETRI KONTROLLERI PROGRAMMI LÄHTETEKST

```
; spectrometer.asm                              (C) Joel Kuusk 2004-2005
;
; FIRS (Field InfraRed Scanner) controller's program
;
; This program was designed, compiled, and written into
; controller's flash memory with AVR Studio and AVR ISP
; in-system programmer by Atmel Corporation

.listmac                ;Show macros in list file

.include "m128def.inc"

;*** registers ***
.def MAXSTEPH   = R2
.def MAXSTEPL   = R3
.def STEPNO     = R4    ;Number of steps since previous measurement
.def U1TXBUF    = R5    ;15 byte FIFO buffer; upper nibble - start, lower nibble - end
.def U1RXCOUNT  = R6
.def U1TXCOUNT  = R7
.def ACCELL     = R8    ;number of steps while accelerating
.def ACCELH     = R9    ;   /
.def CURSTEP    = R15
.def temp       = R16
.def temp2      = R17
.def buf        = R18
.def FLAGS      = R19
.def GOTOL      = R20
.def GOTOH      = R21   ;if GOTOH==0xFF, then goto is disabled
.def POSL       = R22
.def POSH       = R23
.def STEPL      = R24
.def STEPH      = R25
; X register (R27:R26) - USART0 Tx buffer start pointer
; Y register (R29:R28) - USART0 Tx buffer end pointer
; Z register (R31:R30) - temporary register

;*** memory map ***
.equ u1txbuf    = 0x100         ;15-byte USART1 (M201) transmit FIFO buffer
.equ vars       = 0x110         ;Variables in SRAM (max 64)
.equ memstart   = 0x150         ;Almost 4-kbyte USART0 (PC) transmit FIFO buffer
.equ memend     = RAMEND - 50   ;leave 50 bytes for stack

;*** variables in SRAM (displacement from 'vars') ***
.equ measteps   = 0     ;number of steps after which measurements are taken
.equ sstep0h    = 3     ;initial step count high byte
.equ sstep0l    = 4     ;initial step count low byte
.equ ssteph     = 5     ;minimum allowed STEPH value (limits stepping freqency)
.equ sstepl     = 6     ;   /
.equ dstep      = 7     ;stepping speed incremet value (0 = const step size)
.equ wordcount  = 8     ;M201's word count (2 or 3)
.equ return     = 9     ;return value to PC (0xFF means no byte is returned)
.equ tstep0h    = 10    ;max transport stepping speed
.equ tstep0l    = 11    ;   /
.equ tsteph     = 12    ;initial transport stepping speed
.equ tstepl     = 13    ;   /

;*** flags in FLAGS register ***
.equ SCANNING   = 0     ;scanning in progress
.equ INIT       = 1     ;controller initialization in progress
.equ MANSTEP    = 3     ;manual stepping enable flag (1 - enabled)
.equ DIRECTION  = 6     ;stepping direction (1 - forward)
.equ NEXTSTEP   = 7     ;sort of direction of next step...
;For different steps 9, A, 6, or 5 has to be written to PortA
;forward stepping: 9->A->6->5->9...
```

```
;backward stepping 9->5->6->A->9...
;9 XOR 3 = A (3 = 0011)
;A XOR C = 6 (C = 1100)
;6 XOR 3 = 5
;5 XOR C = 9
;in order to take step forward from 9 or 6 we must XOR them with 3
;and to step forward from A or 5 we must XOR them with C
;in order to take step back from 9 or 6 we must XOR them with C
;and to step back from A or 5 we must XOR them with 3
;
;so NEXTSTEP flag indicates if we must XOR with 3 or C to get
;next step. To change stepping direction we must simply invert
;NEXTSTEP flag. 0 means XOR C and 1 means XOR 3

;*** pin numbers ***
;PortE
.equ PCRTS  = 2         ;RTS signal from PC
.equ FMANI  = 4         ;manual forward stepping button
.equ BMANI  = 5         ;manual backward stepping button
.equ LMIN   = 6         ;minimum wavelength signal
.equ LMAX   = 7         ;maximum wavelength signal

;PortC
.equ LED    = 1         ;LED output
.equ PLUS1  = 2         ;+1 output (active low)
.equ MINUS1 = 3         ;-1 output (active low)

;*** Error codes ***
.equ u0buffull  = 0x10  ;USART0 (PC) Tx FIFO buffer overflow
.equ m201slow   = 0x20  ;M201 too slow for this sampling rate

;*** Constants ***
.equ dbnctime   = 200   ;32000/(8*200) = 20 Hz = 50 ms delay for button debounce
.equ initspeed  = 2304  ;14745600/(64*2304) = 100 Hz
.equ stoptime   = 72    ;14745600/(1024*72) = 200 Hz = 5 ms delay for switching
                        ;off motor current

;*** Macros ***
;store status and temporary registers
.macro  push_sreg
    push    ZH          ;store Z
    push    ZL          ;   /
    push    temp2       ;store temp2
    push    temp        ;store temp
    in      temp,SREG
    push    temp        ;store status register
.endmacro

;restore status and temporary registers
.macro  pop_sreg
    pop     temp        ;pop status register
    out     SREG,temp
    pop     temp        ;restore temp
    pop     temp2       ;restore temp2
    pop     ZL          ;restore Z
    pop     ZH          ;   /
.endmacro

;set forward direction
.macro  dir_forward
    sbrc    FLAGS,DIRECTION             ;if DIRECTION == false...
    rjmp    forward_ok                  ;...then skip this
    push    temp                        ;store temp register
    sbr     FLAGS,(1<<DIRECTION)        ;set DIRECTION flag
    ldi     temp,(1<<NEXTSTEP)          ;invert NEXTSTEP flag
    eor     FLAGS,temp                  ;   /
    pop     temp                        ;restore temp register
forward_ok:
```

```
.endmacro

;set backward direction
.macro  dir_backward
    sbrs    FLAGS,DIRECTION             ;if DIRECTION == true...
    rjmp    backward_ok                 ;...then skip this
    push    temp                        ;store temp register
    cbr     FLAGS,(1<<DIRECTION)        ;clear DIRECTION flag
    ldi     temp,(1<<NEXTSTEP)          ;invert NEXTSTEP flag
    eor     FLAGS,temp                  ;   /
    pop     temp                        ;restore temp register
backward_ok:
.endmacro

;disable sleep
;(in Atmega128 datasheet it is recommended to enable sleep just
;before going to sleep and disable it just after waking up)
.macro  disable_sleep
    push    temp                ;store temp register
    in      temp,MCUCR          ;load MCU control register to temp
    andi    temp,0b11000011     ;disable sleep, select idle mode for sleep
    out     MCUCR,temp          ;write MCU control register
    pop     temp                ;restore temp register
.endmacro

;load variable from SRAM to register
;loadvar    temp,meassteps              ;load (Z+meassteps) to temp register
.macro  loadvar
    push    ZH
    push    ZL
    ldi     ZH,high(vars)               ;Init Z pointer to SRAM location
    ldi     ZL,low(vars)                ;   /
    ldd     @0,Z+@1                     ;load variable (second parameter) to
                                        ;register (first parameter)
    pop     ZL
    pop     ZH
.endmacro

;set variable in SRAM from register
;setvar     meassteps,temp              ;set (Z+meassteps) to value in temp register
.macro  setvar
    push    ZH
    push    ZL
    ldi     ZH,high(vars)               ;Init Z pointer to SRAM location
    ldi     ZL,low(vars)                ;   /
    std     Z+@0,@1                     ;set variable (first parameter) to value
                                        ;in register (second parameter)
    pop     ZL
    pop     ZH
.endmacro

;*** reset and interrupt vectors ***
.org 0x00
    jmp main
.org INT7addr   ;MAX wavelength
    jmp max_wl
.org INT6addr   ;MIN wavelength
    jmp min_wl
.org INT5addr   ;BMANI button
    jmp bmani_int
.org INT4addr   ;FMANI button
    jmp fmani_int
.org URXC0addr  ;USART0 Receive Complete
    jmp U0receive_int
.org URXC1addr  ;USART1 Receive Complete
    jmp U1receive_int
.org UDRE0addr  ;USART0 Data Register Empty
    jmp U0transmit_int
```

```
.org UDRE1addr  ;USART1 Data Register Empty
    jmp U1transmit_int
.org OC1Aaddr   ;TMR1 A compare
    jmp step
.org OC2addr    ;TMR2 compare
    jmp switchoff

;*** program code ***
.org 0x46
main:
    cli                             ;Disable global interrupts

    ldi     temp,high(RAMEND)       ;Set stack pointer to top of RAM
    ldi     temp2,low(RAMEND)
    out     SPH,temp
    out     SPL,temp2

    ldi     temp,(1<<WDCE)|(1<<WDE) ;disable watchdog timer (special procedure)
    out     WDTCR,temp              ;   /
    cbr     temp,(1<<WDE)           ;   /
    out     WDTCR,temp              ;   /

    disable_sleep

    in      temp,ACSR           ;load analog comparator control register to temp
    cbr     temp,(1<<ACIE)      ;disable analog comparator interrupt
    out     ACSR,temp           ;   /
    sbr     temp,(1<<ACD)       ;switch off analog comparator
    out     ACSR,temp           ;(this reduces power consumption)

    ldi     YH,high(memstart)   ;initialize USART0 Tx buffer memory pointers
    ldi     YL,low(memstart)    ;   /
    ldi     XH,high(memstart)   ;   /
    ldi     XL,low(memstart)    ;   /

    clr     POSH                ;clear position high byte
    clr     POSL                ;clear position low byte
    ser     GOTOH               ;disable goto
    ser     GOTOL               ;   /
    setvar  return,GOTOL        ;set return variable to 0xFF
    clr     FLAGS               ;clear FLAGS register
    clr     U1TXCOUNT           ;clear USART1 transmit counter
    clr     U1RXCOUNT           ;clear USART1 receive counter
    clr     U1TXBUF             ;clear U1 transmit buffer counters

    clr     temp                ;clear Lawson word count
    setvar  wordcount,temp      ;   /

;scanning speed
    ldi     temp,high(768)      ;14745600/(64*768) = 300 Hz
    setvar  sstep0h,temp        ;   /
    ldi     temp,low(768)       ;   /
    setvar  sstep0l,temp        ;   /
    ldi     temp,high(768)      ;14745600/(64*768) = 300 Hz
    setvar  ssteph,temp         ;   /
    ldi     temp,low(768)       ;   /
    setvar  sstepl,temp         ;   /

;transport speed
    ldi     temp,high(384)      ;14745600/(64*384) = 600 Hz
    setvar  tsteph,temp         ;   /
    mov     MAXSTEPH,temp       ;   /
    ldi     temp,low(384)       ;   /
    setvar  tstepl,temp         ;   /
    mov     MAXSTEPL,temp       ;   /
    ldi     temp,high(768)      ;14745600/(64*768) = 300 Hz
    setvar  tstep0h,temp        ;   /
    mov     STEPH,temp          ;   /
```

```
    ldi     temp,low(768)       ;   /
    setvar  tstep0l,temp        ;   /
    mov     STEPL,temp          ;   /
    ldi     temp,0x05           ;set stepping freq increase coefficent
    setvar  dstep,temp          ;   /

    ser     temp                            ;
    out     DDRA,temp                       ;Set PORTA as output
    out     DDRC,temp                       ;Set PORTC as output
    clr     temp                            ;
    out     DDRE,temp                       ;Set PORTE as input
    out     PORTA,temp                      ;Clear PORTA
    sbr     temp, (1<<PLUS1)|(1<<MINUS1)    ;Clear PORTC, but set PLUS1 and MINUS1
    out     PORTC,temp                      ;high (active low)

    in      temp,SFIOR                      ;Disable pull-ups globally
    sbr     temp, (1<<PUD)                  ;   /
    out     SFIOR,temp                      ;   /

;USART0 init (PC)
    ldi     temp,high(3071)                 ;Baud rate 300
    ldi     temp2,low(3071)
    sts     UBRR0H,temp
    out     UBRR0L,temp2
    ldi     temp,(1<<RXEN0)|(1<<TXEN0)|(1<<RXCIE0)  ;Enable receiver and transmitter,
    out     UCSR0B,temp                             ;receive complete interrupt
    ldi     temp,(1<<UCSZ00)|(1<<UCSZ01)            ;8 data bits, no parity, 1 stop bit
    sts     UCSR0C,temp

;USART1 init (M201)
    ldi     temp,high(3071)                 ;Baud rate 300
    ldi     temp2,low(3071)
    sts     UBRR1H,temp
    sts     UBRR1L,temp2
    ldi     temp,(1<<RXEN1)|(1<<TXEN1)|(1<<RXCIE1)  ;Enable receiver and transmitter,
    sts     UCSR1B,temp                             ;receive complete interrupt
    ldi     temp,(1<<UCSZ10)|(1<<UCSZ11)            ;8 data bits, no parity, 1 stop bit
    sts     UCSR1C,temp

;Set interrupts
    ldi     temp,(1<<INT4)|(1<<INT5)|(1<<INT6)|(1<<INT7)    ;Enable external
    out     EIMSK,temp                                      ;interrupts 4..7
    ldi     temp,(1<<ISC40)|(1<<ISC50)|(1<<ISC61)|(1<<ISC71)
                                    ;External interrupts 4 and 5 are triggered on
                                    ;any logical change on those pins
                                    ;External interrupts 6 and 7 are triggered on
    out     EICRB,temp              ;falling edge

    ser     temp                    ;Clear all pending external interrupts
    out     EIFR,temp               ;   /

    ldi     temp,dbnctime           ;Set T/C0 compare value
    out     OCR0,temp               ;   /
    ldi     temp,(1<<CS01)|(1<<WGM01)   ;Clear timer on compare match and
    out     TCCR0,temp              ;Set T/C0 prescaler clock/8

    ldi     temp,(1<<WGM12)|(1<<CS11)|(1<<CS10) ;Clear timer on compare match
    out     TCCR1B,temp                         ;T/C1 prescaler Clock/64

    ldi     temp,stoptime                       ;Set T/C2 compare value
    out     OCR2,temp                           ;   /
    ldi     temp,(1<<CS20)|(1<<CS22)            ;Normal mode, OC2 disconnected and
    out     TCCR2,temp                          ;Set T/C2 prescaler clock/1024

    ldi     temp,0x09                           ;load CURSTEP with 0x09 (1001)
    mov     CURSTEP,temp                        ;   /
    sbr     FLAGS,(1<<DIRECTION)                ;set forward direction
    cbr     FLAGS,(1<<NEXTSTEP)                 ;   /
```

```
    sbr     FLAGS,(1<<MANSTEP)                  ;Enable manual stepping
    sei                                         ;Enable global interrupts


;main loop
loop:
    in      temp,MCUCR          ;load MCU control register to temp
    sbr     temp,(1<<SE)        ;enable sleep
    out     MCUCR,temp          ;write MCU control register
    sleep                       ;go to sleep
    rjmp    loop                ;Infinite loop


step:
    push_sreg
    disable_sleep
    cpi     GOTOH,0xFF          ;If GOTOH==0xFF...
    breq    nogoto              ;then goto disabled and we branch
    subi    GOTOL,0x01          ;Decrement step count
    sbci    GOTOH,0x00          ;   /
    brpl    nogoto              ;if not negative...
    rjmp    wearethere          ;...then skip this
nogoto:
    in      temp,PINC           ;Load PINC to temp
    cbr     temp,(1<<PLUS1)     ;Clear +1 (active low)
    sbrc    FLAGS,DIRECTION     ;If stepping backward...
    rjmp    step_plus1          ;...then skip this
    sbr     temp,(1<<PLUS1)     ;Set +1
    cbr     temp,(1<<MINUS1)    ;Clear -1
    subi    POSL,0x02           ;subtract from position
    sbci    POSH,0x00           ;   /
step_plus1:
    subi    POSL,0xFF           ;add 1 to position; we have no add immediate cmd
    sbci    POSH,0xFF           ;so we use sub immediate and two's complement of 1
    out     PORTC,temp          ;Write to PORTC
    sbrc    FLAGS,INIT          ;if INIT == false...
    rjmp    step_f_ok           ;..then skip this, otherwise don't change speed
    loadvar temp,dstep          ;load dstep value from SRAM to temp
    clr     temp2               ;   /
    cp      ACCELL,GOTOL        ;compare acceleration step count...
    cpc     ACCELH,GOTOH        ;...and number of steps to go
    brlo    step_changespeed    ;if ACCEL < GOTO then branch
    neg     temp                ;temp = -temp
    com     temp2               ;   /
step_changespeed:
    sub     STEPL,temp          ;change step size
    sbc     STEPH,temp2         ;   /
    cp      STEPL,MAXSTEPL      ;Compare with current frequency
    cpc     STEPH,MAXSTEPH      ;   /
    brlo    step_maxf           ;Branch if lower
    ldi     temp,0x01           ;load 0x0001 to temp
    clr     temp2               ;
    add     ACCELL,temp         ;increment acceleration step count
    adc     ACCELH,temp2        ;   /
    rjmp    step_f_ok           ;branch
step_maxf:
    mov     STEPH,MAXSTEPH      ;set stepping frequency to maximum
    mov     STEPL,MAXSTEPL      ;   /
step_f_ok:
    out     OCR1AH,STEPH                ;Set T/C1 compare A high value
    out     OCR1AL,STEPL                ;Set T/C1 compare A low value
    ldi     temp,0x03                   ;load 0x03 to temp register
    sbrs    FLAGS,NEXTSTEP              ;if NEXTSTEP == true...
    ldi     temp,0x0C                   ;...then skip this
    eor     CURSTEP,temp                ;CURSTEP = CURSTEP XOR temp
    ldi     temp,(1<<NEXTSTEP)          ;invert NEXTSTEP flag
    eor     FLAGS,temp                  ;   /
    out     PORTA,CURSTEP               ;Output step code to motor
```

```
    clr     temp                        ;Reset T/C2
    out     TCNT2,temp                  ;   /
    ldi     temp,(1<<OCF2)              ;Clear T/C2 interrupt flag
    out     TIFR,temp                   ;   /
    in      temp,TIMSK                  ;Load TIMSK register to temp
    sbr     temp,(1<<OCIE2)             ;Enable T/C2 output compare
    out     TIMSK,temp                  ;   /
    in      temp,PINC                   ;Load PINC
    sbr     temp,(1<<PLUS1)|(1<<MINUS1) ;Set +1 and -1
    out     PORTC,temp                  ;Write to PORTC
    sbrs    FLAGS,SCANNING              ;if SCANNING == true...
    rjmp    alldone                     ;...then skip this
    dec     STEPNO                      ;decrement step count
    breq    step_meas                   ;branch if zero
    rjmp    alldone
step_meas:
    ldi     temp,m201slow               ;load error code to temp
    ser     temp2                       ;load temp2 with 0xFF
    cpse    U1RXCOUNT,temp2             ;if all bytes are received..
    rjmp    error                       ;...then skip this - M201 is fast enough
    ldi     temp, 0x81                  ;load 0x81 to temp
    rcall   U1txqueue                   ;send to M201
    clr     temp                        ;clear temp
    rcall   U1txqueue                   ;send to M201
    ldi     temp, 0x81                  ;load 0x81 to temp
    rcall   U1txqueue                   ;send to M201
    loadvar U1RXCOUNT,wordcount         ;load U1RXCOUNT
    loadvar STEPNO,meassteps            ;load meassteps to STEPNO
    rjmp    alldone                     ;branch
wearethere:
    in      temp,TIMSK                  ;Disable T/C1 output compare A interrupt
    cbr     temp,(1<<OCIE1A)            ;   /
    out     TIMSK,temp                  ;   /
    clr     temp                        ;Clear PORTA
    out     PORTA,temp                  ;   /
    in      temp,PINC                   ;Load PINC
    cbr     temp,(1<<LED)               ;Clear LED
    out     PORTC,temp                  ;Write to PORTC
    in      temp,EIMSK                  ;Load EIMSK to temp
    sbrc    FLAGS,MANSTEP               ;if manual stepping enabled...
    sbr     temp,(1<<INT4)|(1<<INT5)    ;...then enable external interrupts 4 and 5
    out     EIMSK,temp                  ;(BMANI and FMANI)
    ldi     temp,(1<<INTF4)|(1<<INTF5)  ;Clear pending FMANI and BMANI interrupts
    out     EIFR,temp                   ;   /
    loadvar temp,return                 ;if return == 0xFF...
    cpi     temp,0xFF                   ;   /
    breq    noreturn                    ;...then branch
    rcall   U0txqueue                   ;send return to PC
    ser     temp                        ;set return to 0xFF
    setvar  return,temp                 ;   /
noreturn:
    sbrs    FLAGS,SCANNING              ;If SCANNING == true...
    rjmp    alldone                     ;...then skip this
    in      temp,EIMSK                  ;Load EIMSK to temp
    cbr     temp,(1<<BMANI)|(1<<FMANI)  ;Disable external BMANI and FMANI interrupts
    out     EIMSK,temp                  ;   /
    ldi     temp,(1<<INTF4)|(1<<INTF5)  ;Clear pending FMANI and BMANI interrupts
    out     EIFR,temp                   ;   /
    sei                                 ;enable global interrupts
last_reading_loop:
    tst     U1RXCOUNT                   ;test U1RXCOUNT
    brpl    last_reading_loop           ;loop if >= 0
    cli                                 ;disable global interrupts
    in      temp,EIMSK                  ;Load EIMSK to temp
    sbrc    FLAGS,MANSTEP               ;if manual stepping enabled...
    sbr     temp,(1<<INT4)|(1<<INT5)    ;...then enable external interrupts 4 and 5
    out     EIMSK,temp                  ;(BMANI and FMANI)
    ldi     temp,(1<<INTF4)|(1<<INTF5)  ;Clear pending FMANI and BMANI interrupts
```

```
    out     EIFR,temp                   ;   /
    cbr     FLAGS,(1<<SCANNING)         ;clear SCANNING flag
alldone:
    pop_sreg                            ;Return from interrupt
    reti


switchoff:
    push_sreg
    disable_sleep
    in      temp,TIMSK                  ;Disable T/C2 output compare interrupt
    cbr     temp,(1<<OCIE2)             ;   /
    out     TIMSK,temp                  ;   /
    clr     temp                        ;Clear PORTA
    out     PORTA,temp                  ;   /
    pop_sreg
    reti


fmani_int:
    push_sreg
    disable_sleep
    in      temp,EIMSK                  ;Load EIMSK to temp
    cbr     temp,(1<<INT4)|(1<<INT5)    ;Disable external interrupts 4 and 5
    out     EIMSK,temp                  ;(BMANI and FMANI)
    sei                                 ;Enable global interrupts
    rcall   debounce                    ;Debounce
    cli                                 ;Disable global interrupts
    ser     temp                        ;set return to 0xFF (nothing is sent to PC)
    setvar  return,temp                 ;   /
    sbic    PINE,FMANI                  ;If FMANI is pushed then...
    rjmp    fmani_release               ;...skip this
    dir_forward                         ;Set direction to forward
    ser     GOTOH                       ;disable goto
    rcall   startstepping               ;start stepping
    rjmp    fmani_reti                  ;Jump to return
fmani_release:
    clr     GOTOH                       ;set GOTO to 0x0000,
    clr     GOTOL                       ;so that we stop on next step
    in      temp,EIMSK                  ;Load EIMSK to temp
    sbr     temp,(1<<INT5)              ;Enable external interrupt 5 (BMANI)
    out     EIMSK,temp                  ;   /
fmani_reti:
    in      temp,EIMSK                  ;Load EIMSK to temp
    sbr     temp,(1<<INT4)              ;Enable external interrupt 4 (FMANI)
    out     EIMSK,temp                  ;   /
    ldi     temp,(1<<BMANI)|(1<<FMANI)  ;Clear pending FMANI and BMANI interrupts
    out     EIFR,temp                   ;   /
    pop_sreg
    reti


bmani_int:
    push_sreg
    disable_sleep
    in      temp,EIMSK                  ;Load EIMSK to temp
    cbr     temp,(1<<BMANI)|(1<<FMANI)  ;Disable external BMANI and FMANI interrupts
    out     EIMSK,temp                  ;   /
    sei                                 ;Enable global interrupts
    rcall   debounce                    ;Debounce
    cli                                 ;Disable global interrupts
    ser     temp                        ;set return to 0xFF (nothing is sent to PC)
    setvar  return,temp                 ;   /
    sbic    PINE,BMANI                  ;If BMANI is pushed then...
    rjmp    bmani_release               ;...skip this
    dir_backward                        ;Set direction to backward
    ser     GOTOH                       ;disable goto
    rcall   startstepping               ;start stepping
```

```
    rjmp    bmani_reti                  ;Jump to return
bmani_release:
    clr     GOTOH                       ;set GOTO to 0x0000,
    clr     GOTOL                       ;so that we stop on next step
    in      temp,EIMSK                  ;Load EIMSK to temp
    sbr     temp,(1<<INT4)              ;Enable external interrupt 4 (FMANI)
    out     EIMSK,temp                  ;   /
bmani_reti:
    in      temp,EIMSK                  ;Load EIMSK to temp
    sbr     temp,(1<<INT5)              ;Enable external interrupt 5 (BMANI)
    out     EIMSK,temp                  ;   /
    ldi     temp,(1<<BMANI)|(1<<FMANI)  ;Clear pending FMANI and BMANI interrupts
    out     EIFR,temp                   ;   /
    pop_sreg
    reti

debounce:
    push_sreg
    ldi     temp, (1<<AS0)              ;T/C0 clocked from 32kHz crystal
    out     ASSR,temp                   ;   /
    ldi     temp,(1<<OCF0)              ;Clear T/C0 interrupt flag
    out     TIFR,temp                   ;   /
    clr     temp                        ;Reset T/C0
    out     TCNT0,temp                  ;   /
dbnceloop:
    in      temp,TIFR                   ;Copy T/C interrupt flag register to temp
    bst     temp,OCF0                   ;Store OCF0 to T
    brtc    dbnceloop                   ;If Output Compare Flag 0 not set then loop
    pop_sreg
    ret


min_wl:
    push_sreg
    disable_sleep
    sbrc    FLAGS,DIRECTION             ;If stepping backward...
    rjmp    min_wl_ok                   ;...then skip this
    clr     GOTOH                       ;set GOTO to 0x0000,
    clr     GOTOL                       ;so that we stop on next step
    loadvar temp,return                 ;If return == 0xFF...
    cpi     temp,0xFF                   ;   /
    breq    min_wl_ok                   ;...then branch
    ldi     temp,0xF0                   ;load return variable with 0xF0
    setvar  return,temp                 ;   /
    sbrs    FLAGS,INIT                  ;if INIT == true
    rjmp    min_wl_ok                   ;...then skip this
    clr     POSH                        ;clear position high byte
    ldi     temp,0x0A                   ;load 0x0A (10) to temp
    mov     POSL,temp                   ;move temp to POSL (POSL=10)
    cbr     FLAGS,(1<<INIT)             ;clear INIT flag
min_wl_ok:
    pop_sreg
    reti


max_wl:
    push_sreg
    disable_sleep
    sbrs    FLAGS,DIRECTION             ;If stepping forward...
    rjmp    max_wl_ok                   ;...then skip this
    clr     GOTOH                       ;set GOTO to 0x0000,
    clr     GOTOL                       ;so that we stop on next step
    loadvar temp,return                 ;If return == 0xFF...
    cpi     temp,0xFF                   ;   /
    breq    max_wl_ok                   ;...then branch
    ldi     temp,0xF1                   ;load return variable with 0xF1
    setvar  return,temp                 ;   /
max_wl_ok:
```

```
    pop_sreg
    reti


startstepping:
    push_sreg
    sbrc    FLAGS,DIRECTION             ;If stepping backward...
    rjmp    stepping_test_forward       ;...then skip this
    sbis    PINE,LMIN                   ;If already min wavelength then...
    rjmp    stepping_minmax             ;...don't step...
    rjmp    stepping_test_ok            ;...else step
stepping_test_forward:
    sbis    PINE,LMAX                   ;If already max wavelength then...
    rjmp    stepping_minmax             ;...don't step
stepping_test_ok:
    sbrc    FLAGS,INIT                  ;if INIT == false...
    rjmp    stepping_speed_ok           ;...then skip this
    sbrc    FLAGS,SCANNING              ;if not scanning...
    rjmp    stepping_scanning           ;...then skip this
    loadvar MAXSTEPH,tsteph             ;Copy max step size to working register
    loadvar MAXSTEPL,tstepl             ;   /
    loadvar STEPH,tstep0h               ;Copy initial step size to working register
    loadvar STEPL,tstep0l               ;   /
    rjmp    stepping_speed_ok           ;branch
stepping_scanning:
    loadvar MAXSTEPH,ssteph             ;Copy max step size to working register
    loadvar MAXSTEPL,sstepl             ;   /
    loadvar STEPH,sstep0h               ;Copy initial step size to working register
    loadvar STEPL,sstep0l               ;   /
stepping_speed_ok:
    clr     ACCELL                      ;clear acceleration step count
    clr     ACCELH                      ;   /
    clr     temp                        ;Reset T/C1
    out     TCNT1H,temp                 ;   /
    out     TCNT1L,temp                 ;   /
    out     OCR1AH,STEPH                ;Set T/C1 compare A high value
    out     OCR1AL,STEPL                ;Set T/C1 compare A low value
    in      temp,TIMSK                  ;Load TIMSK register to temp
    sbr     temp,(1<<OCIE1A)            ;Enable T/C1 output compare A
    out     TIMSK,temp                  ;   /
    in      temp,PINC                   ;Load PINC
    sbr     temp,(1<<LED)               ;Set LED
    out     PORTC,temp                  ;write to PORTC
    rjmp    stepping_return             ;jump to return
stepping_minmax:
    cbr     FLAGS,(1<<INIT)             ;clear INIT flag
    in      temp,EIMSK                  ;Load EIMSK to temp
    sbrc    FLAGS,MANSTEP               ;if manual stepping enabled...
    sbr     temp,(1<<INT4)|(1<<INT5)    ;...then enable external interrupts 4 and 5
    out     EIMSK,temp                  ;(BMANI and FMANI)
    ldi     temp,(1<<INTF4)|(1<<INTF5)  ;Clear pending FMANI and BMANI interrupts
    out     EIFR,temp                   ;   /
    loadvar temp,return                 ;If return == 0xFF...
    cpi     temp,0xFF                   ;   /
    breq    stepping_return             ;...then branch
    sbis    PINE,LMIN                   ;If already min wavelength then...
    ldi     temp,0xF0                   ;...load temp register with 0xF0
    sbis    PINE,LMAX                   ;If already max wavelength then...
    ldi     temp,0xF1                   ;...load temp register with 0xF1
    setvar  return,temp                 ;save return variable to SRAM
    rcall   U0txqueue                   ;send return to PC
stepping_return:
    pop_sreg
    ret


U0receive_int: ;(from PC, interrupt driven)
    push_sreg
```

```
    disable_sleep
    in      temp,UDR0               ;Load receive buffer to temp
    sbis    PINE,PCRTS              ;If PC RTS set then
    rjmp    to_m201                 ;...skip this

    cpi     temp,0x00               ;if temp == 0x00
    breq    b_cmd_U0echotest        ;USART0 echo test command
    cpi     temp,0x01               ;else if 0x01
    breq    b_cmd_start             ;Start stepping command
    cpi     temp,0x02               ;else if 0x02
    breq    b_cmd_stop              ;Stop scanning and stepping command
    cpi     temp,0x03               ;else if 0x03
    breq    b_cmd_forward           ;forward direction command
    cpi     temp,0x04               ;else if 0x04
    breq    b_cmd_backward          ;backward direction command
    cpi     temp,0x05               ;else if 0x05
    breq    b_cmd_goto              ;goto command
    cpi     temp,0x06               ;else if 0x06
    breq    b_cmd_U0baudrate        ;change USART0 baudrate command
    cpi     temp,0x07               ;else if 0x07
    breq    b_cmd_U1baudrate        ;change USART1 baudrate command
    cpi     temp,0x08               ;else if 0x08
    breq    b_cmd_changestep        ;change step parameters command
    cpi     temp,0x09               ;else if 0x09
    breq    b_cmd_scan              ;Start scanning command
    cpi     temp,0x0A               ;else if 0x0A
    breq    b_cmd_init              ;init position command
    cpi     temp,0x0B               ;else if 0x0B
    breq    b_cmd_setwordcount      ;set lawson word count command
                                    ;(does not affect Lawson itself)
    cpi     temp,0x0C               ;else if 0x0C
    breq    b_cmd_getpos            ;get current position
    cpi     temp,0x0D               ;else if 0x0D
    breq    b_cmd_M201speed         ;measure M201's speed
    cpi     temp,0xA0               ;else if 0xA0
    breq    b_cmd_reset             ;reset controller
    rjmp    cmddone                 ;else unknown cmd
b_cmd_U0echotest:
    rjmp    cmd_U0echotest          ;Jump to command
b_cmd_start:
    rjmp    cmd_start               ;Jump to command
b_cmd_stop:
    rjmp    cmd_stop                ;Jump to command
b_cmd_forward:
    rjmp    cmd_forward             ;Jump to command
b_cmd_backward:
    rjmp    cmd_backward            ;Jump to command
b_cmd_goto:
    rjmp    cmd_goto                ;Jump to command
b_cmd_U0baudrate:
    rjmp    cmd_U0baudrate          ;Jump to command
b_cmd_U1baudrate:
    rjmp    cmd_U1baudrate          ;Jump to command
b_cmd_changestep:
    rjmp    cmd_changestep          ;Jump to command
b_cmd_scan:
    rjmp    cmd_scan                ;Jump to command
b_cmd_init:
    rjmp    cmd_init                ;Jump to command
b_cmd_setwordcount:
    rjmp    cmd_setwordcount        ;Jump to command
b_cmd_getpos:
    rjmp    cmd_getpos              ;Jump to command
b_cmd_reset:
    rjmp    cmd_reset               ;Jump to command
b_cmd_M201speed:
    rjmp    cmd_M201speed           ;Jump to command
```

```
to_m201:
    rcall   U1txqueue                   ;Send received byte to M201
    rjmp    cmddone

cmd_U0echotest:
    rcall   U0readbyte                  ;Read byte to be echoed
    mov     temp,buf                    ;move received byte to temp
    rcall   U0txqueue                   ;Echo received byte back to PC
    rjmp    cmddone

cmd_start:
    in      temp,EIMSK                  ;Load EIMSK to temp
    cbr     temp,(1<<INT4)|(1<<INT5)    ;Disable external interrupts 4 and 5
    out     EIMSK,temp                  ;(BMANI and FMANI)
    rcall   U0readbyte                  ;Read count high byte
    mov     ZH,buf                      ;   /
    rcall   U0readbyte                  ;Read count low byte
    mov     ZL,buf                      ;   /
    ldi     temp,0x01                   ;load return variable with 0x01
    setvar  return,temp                 ;   /
    ser     GOTOH                       ;disable goto
    clr     temp                        ;clear temp
    cp      ZL,temp                     ;if Z == 0...
    cpc     ZH,temp                     ;   /
    breq    start_test_ok               ;...then start stepping
    movw    GOTOL,ZL                    ;move step count to GOTO
start_test_ok:
    rcall   startstepping               ;start stepping
    rjmp    cmddone

cmd_stop:
    clr     GOTOH                       ;set GOTO to 0x0000,
    clr     GOTOL                       ;so that we stop on next step
    ser     temp                        ;disable RETURN (set to 0xFF)
    setvar  return,temp                 ;   /
    movw    XL,YL                       ;empty USART0 transmit buffer
    ldi     temp,0x02                   ;load 0x02 to temp
    rcall   U0txqueue                   ;send to PC
    rcall   U0txqueue                   ;send to PC
    rcall   U0txqueue                   ;send to PC
    rcall   U0txqueue                   ;send to PC
    rjmp    cmddone

cmd_forward:
    dir_forward                         ;Set direction to forward
    rjmp    cmddone

cmd_backward:
    dir_backward                        ;Set direction to backward
    rjmp    cmddone

cmd_goto:
    in      temp,EIMSK                  ;Load EIMSK to temp
    cbr     temp,(1<<INT4)|(1<<INT5)    ;Disable external interrupts 4 and 5
    out     EIMSK,temp                  ;(BMANI and FMANI)
    dir_forward                 ;Set direction to forward
    rcall   U0readbyte          ;Read position high byte
    mov     GOTOH,buf           ;   /
    rcall   U0readbyte          ;Read position low byte
    mov     GOTOL,buf           ;   /
    cp      GOTOL,POSL          ;compare low bytes
    cpc     GOTOH,POSH          ;compare with carry high bytes
    in      temp,SREG           ;store status register
    push    temp                ;   /
    sub     GOTOL,POSL          ;subtract POS from GOTO
    sbc     GOTOH,POSH          ;   /
    pop     temp                ;restore status register
    out     SREG,temp           ;   /
```

```
    brsh    goto_checkscanning  ;if GOTO>=POS, then branch
    dir_backward            ;else set direction to backward
    com     GOTOH           ;one's complement of GOTOH, two's complement of GOTOL
    neg     GOTOL           ;now direction is correct and in GOTO we have step count
goto_checkscanning:
    sbrs    FLAGS,SCANNING      ;if scanning...
    rjmp    goto_stepping       ;...then skip this
    ldi     temp,0x09           ;load return value to temp
    rcall   U0txqueue           ;send to PC
    ldi     temp,0x81           ;load 0x81 to temp
    rcall   U1txqueue           ;send to M201
    clr     temp                ;clear temp
    rcall   U1txqueue           ;send to M201
    ldi     temp,0x81           ;load 0x81 to temp
    rcall   U1txqueue           ;send to M201
    loadvar U1RXCOUNT,wordcount ;load wordcount to U1RXCOUNT
    sei                         ;enable global interrupts
goto_wait:
    tst     U1RXCOUNT           ;test U1RXCOUNT
    brpl    goto_wait           ;loop if >= 0
    cli                         ;disable global interrupts
goto_stepping:
    rcall   startstepping       ;start stepping
    ldi     temp,0x05           ;load temp register with 0x05
    sbrc    FLAGS,SCANNING      ;if not scanning...
    ser     temp                ;...then skip, otherwise disable return
    setvar  return,temp         ;set return variable in SRAM
    rjmp    cmddone

cmd_U0baudrate:
    rcall   U0readbyte          ;Read USART0 baudrate high byte
    mov     temp,buf            ;   /
    rcall   U0readbyte          ;Read USART0 baudrate low byte
    sts     UBRR0H,temp         ;Set USART0 baud rate high byte
    out     UBRR0L,buf          ;Set USART0 baud rate low byte
    rjmp    cmddone

cmd_U1baudrate:
    rcall   U0readbyte          ;Read USART1 baudrate high byte
    mov     temp,buf            ;   /
    rcall   U0readbyte          ;Read USART1 baudrate low byte
    sts     UBRR1H,temp         ;Set USART1 baud rate high byte
    sts     UBRR1L,buf          ;Set USART1 baud rate low byte
    rjmp    cmddone

cmd_changestep:
    rcall   U0readbyte          ;Read step size high byte
    mov     STEPH,buf           ;   /
    setvar  sstep0h,STEPH       ;save to SRAM
    rcall   U0readbyte          ;Read step size low byte
    mov     STEPL,buf           ;   /
    setvar  sstep0l,STEPL       ;save to SRAM
    rcall   U0readbyte          ;Read maximum step frequency high byte
    setvar  ssteph,buf          ;save to SRAM
    rcall   U0readbyte          ;Read maximum step frequency low byte
    setvar  sstepl,buf          ;save to SRAM
    rcall   U0readbyte          ;Read initial transport stepping speed high byte
    setvar  tstep0h,buf         ;save to SRAM
    rcall   U0readbyte          ;Read initial transport stepping speed low byte
    setvar  tstep0l,buf         ;save to SRAM
    rcall   U0readbyte          ;Read max transport stepping speed high byte
    setvar  tsteph,buf          ;save to SRAM
    rcall   U0readbyte          ;Read max transport stepping speed low byte
    setvar  tstepl,buf          ;save to SRAM
    rcall   U0readbyte          ;Read step change
    setvar  dstep,buf           ;save to SRAM
    rcall   U0readbyte          ;Read step count between measurements
    setvar  measteps,buf        ;save step count to SRAM
```

```
    rcall   U0readbyte          ;Read manual stepping flag
    cbr     FLAGS,(1<<MANSTEP)  ;Disable manual stepping
    clr     temp                ;clear temp register
    cpse    buf,temp            ;If buf == 0x00
    sbr     FLAGS,(1<<MANSTEP)  ;...then skip this
    in      temp,EIMSK                  ;Load EIMSK to temp
    sbrc    FLAGS,MANSTEP               ;if manual stepping enabled...
    sbr     temp,(1<<INT4)|(1<<INT5)    ;...then enable external interrupts 4 and 5
    out     EIMSK,temp                  ;(BMANI and FMANI)
    ldi     temp,(1<<INTF4)|(1<<INTF5)  ;Clear pending FMANI and BMANI interrupts
    out     EIFR,temp                   ;   /
    rjmp    cmddone

cmd_scan:
    sbr     FLAGS,(1<<SCANNING)         ;set SCANNING flag
    clr     STEPNO                      ;clear step count
    rjmp    cmd_goto                    ;rest is same as cmd_goto

cmd_init:
    in      temp,EIMSK                  ;Load EIMSK to temp
    cbr     temp,(1<<INT4)|(1<<INT5)    ;Disable external interrupts 4 and 5
    out     EIMSK,temp                  ;(BMANI and FMANI)
    sbr     FLAGS,(1<<INIT)             ;set INIT = true
    ser     GOTOH                       ;disable goto
    clr     POSH                        ;clear position high byte
    clr     POSL                        ;clear position low byte
    dir_backward                        ;Set direction to backward
    ldi     STEPH,high(initspeed)       ;Set initialization speed
    ldi     STEPL,low(initspeed)        ;   /
    ldi     temp,0x0A                   ;load return variable with 0x0A
    setvar  return,temp                 ;   /
    rcall   startstepping               ;start stepping
    rjmp    cmddone

cmd_setwordcount:
    rcall   U0readbyte              ;Read Lawsons wordcount
    setvar  wordcount,buf           ;save to SRAM
    rjmp    cmddone

cmd_getpos:
    mov     temp,POSH               ;Send position high byte to PC
    rcall   U0txqueue               ;   /
    mov     temp,POSL               ;Send position low byte to PC
    rcall   U0txqueue               ;   /
    rjmp    cmddone

cmd_M201speed:
    lds     temp,TCCR3B             ;store T/C3 prescaler
    push    temp                    ;   /
    ldi     temp,(1<<CS32)          ;Normal mode,
    sts     TCCR3B,temp             ;T/C3 prescaler Clock/256
    ldi     temp,0x01               ;load 0x0001 to temp
    clr     temp2                   ;   /
    sts     TCNT3H,temp2            ;Reset T/C3
    sts     TCNT3L,temp             ;   /
    sei                             ;enable global interrupts
    ldi     temp, 0x81              ;load 0x81 to temp
    rcall   U1txqueue               ;send to M201
    clr     temp                    ;clear temp
    rcall   U1txqueue               ;send to M201
    ldi     temp, 0x81              ;load 0x81 to temp
    rcall   U1txqueue               ;send to M201
    loadvar U1RXCOUNT,wordcount     ;load U1RXCOUNT
    clr     temp2                   ;set temp2 to 0xFF
cmd_M201speed_loop:
    tst     U1RXCOUNT               ;test U1RXCOUNT
    brpl    cmd_M201speed_loop      ;loop if >= 0
    cli                             ;disable global interrupts
```

```
    lds     temp2,TCNT3L            ;Load T/C3 value to temp
    lds     temp,TCNT3H             ;   /
    rcall   U0txqueue               ;send high byte to PC
    mov     temp,temp2              ;move low byte to temp
    rcall   U0txqueue               ;send low byte to PC
    pop     temp                    ;restore T/C3 prescaler
    sts     TCCR3B,temp             ;   /
    rjmp    cmddone

cmd_reset:
    cli                             ;disable global interrupts
    rcall   debounce                ;wait 50 ms
    clr     temp                    ;clear temp register
    sts     UDR1,temp               ;Send to M201 (reset)
    ldi     temp,(1<<WDCE)|(1<<WDE) ;enable watchdog timer control register
    out     WDTCR,temp              ;   /   changing (safety level 1)
    ldi     temp,(1<<WDE)           ;enable watchdog timer, timeout in 16.3ms
    out     WDTCR,temp              ;   /
    in      temp,MCUCR              ;load MCU control register to temp
    sbr     temp,(1<<SE)            ;enable sleep
    out     MCUCR,temp              ;write MCU control register
    sleep                           ;go to sleep and wait for watchdog timer
                                    ;to generate reset
cmddone:
    pop_sreg
    reti


;read byte from PC, result to buf
U0readbyte:
    push_sreg
    in      temp,UCSR0B         ;Disable USART0 receive complete interrupt
    cbr     temp,(1<<RXCIE0)    ;   /
    out     UCSR0B,temp         ;   /
    sei                         ;Enable global interrupts
U0read_loop:
    sbis    UCSR0A,RXC0         ;If receive complete bit is set...
    rjmp    U0read_loop         ;...then skip this
    cli                         ;Disable global interrupts
    in      temp,UCSR0B         ;Enable USART0 receive complete interrupt
    sbr     temp,(1<<RXCIE0)    ;   /
    out     UCSR0B,temp         ;   /
    in      buf,UDR0            ;Load receive buffer to buf
    pop_sreg
    ret


;receive byte from M201 (interrupt driven)
U1receive_int:
    push_sreg
    disable_sleep
    lds     temp,UDR1       ;Load receive buffer to temp
    loadvar temp2,wordcount ;load wordcount to temp2
    dec     U1RXCOUNT       ;decrement U1RXCOUNT
    dec     temp2           ;decrement temp2
    sbrc    FLAGS,SCANNING  ;If not scanning...
    cpse    temp2,U1RXCOUNT ;...then skip, otherwise compare U1RXCOUNT and wordcount
    rcall   U0txqueue       ;send received byte to PC
                            ;skip echoed command byte 0x81 if U1RXCOUNT == wordcount
    pop_sreg
    reti


;put byte in temp to FIFO buffer (to PC)
U0txqueue:
    push    temp2               ;store temp2
    in      temp2,SREG          ;load status register to temp2
    cli                         ;disable global interrupts for atomic write
```

```
    push    temp2               ;store status register
    push    ZH                  ;store Z
    push    ZL                  ;   /
    push    temp                ;store temp (byte to transmit)
    movw    ZL,YL               ;copy Y (end pointer) to Z
    subi    ZL,low(memend)      ;if not at the end of memory...
    sbci    ZH,high(memend)     ;   /
    brne    U0tx_pointer_ok     ;then branch
    ldi     YL,low(memstart)    ;set end pointer to beginning of buffer
    ldi     YH,high(memstart)   ;   /
U0tx_pointer_ok:
    movw    ZL,YL               ;copy Y (end pointer) to Z
    adiw    ZL,0x01             ;add 1 to end pointer
    cp      XL,ZL               ;compare start and end pointer
    cpc     XH,ZH               ;   /
    brne    U0queueok           ;branch if not equal
                                ;here we are in trouble...
    ldi     temp,u0buffull      ;load error code to temp
    rjmp    error               ;branch
U0queueok:
    pop     temp                ;load byte from stack
    st      Y+,temp             ;store indirect and post-inc
    push    temp                ;store temp to stack
    in      temp,UCSR0B         ;Enable USART0 Data Register Empty interrupt
    sbr     temp,(1<<UDRIE0)    ;   /
    out     UCSR0B,temp         ;   /
    pop     temp                ;restore temp
    pop     ZL                  ;restore Z
    pop     ZH                  ;   /
    pop     temp2               ;restore status register (including interrupt bit)
    out     SREG,temp2          ;   /
    pop     temp2               ;restore temp2
    ret


;put byte in temp to FIFO buffer (to M201)
U1txqueue:
    push    temp2               ;store temp2
    in      temp2,SREG          ;load status register to temp2
    cli                         ;disable global interrupts for atomic write
    push    temp2               ;store status register
    push    ZH                  ;store Z
    push    ZL                  ;   /
    push    temp                ;store temp (byte to transmit)
    mov     temp,U1TXBUF        ;move U1TXBUF to temp
    mov     temp2,temp          ;move temp to temp2
    swap    temp2               ;swap nibbles
    inc     temp                ;increment end counter
    andi    temp,0x0F           ;mask upper nibble
    andi    temp2,0x0F          ;mask upper nibble
    cp      temp,temp2          ;compare temp and temp2
    brne    U1queueok           ;branch if not equal
U1write_loop:
    lds     temp,UCSR1A         ;Load UCSR1A to temp
    sbrs    temp,TXC1           ;If transmit complete bit is set...
    rjmp    U1write_loop        ;...then skip this
    rcall   U1sendbyte          ;send one byte from queue to M201
U1queueok:
    ldi     ZH,high(u1txbuf)    ;Init Z pointer to buffer in SRAM
    ldi     ZL,low(u1txbuf)     ;   /
    mov     temp,U1TXBUF        ;move U1TXBUF to temp
    andi    temp,0x0F           ;mask higher nibble (now end in temp)
    add     ZL,temp             ;add counter value to pointer
    pop     temp                ;load byte from stack
    st      Z,temp              ;store indirect
    push    temp                ;store temp to stack
    mov     temp,U1TXBUF        ;move U1TXBUF to temp
    mov     temp2,temp          ;move temp to temp2
```

```
    andi    temp2,0xF0          ;mask lower nibble (now start<<4 in temp2)
    inc     temp                ;increment end counter
    andi    temp,0x0F           ;mask higher nibble (now end in temp)
    add     temp,temp2          ;add temp and temp2, result in temp
    mov     U1TXBUF,temp        ;move temp to U1TXBUF
    lds     temp,UCSR1B         ;Enable USART1 Data Register Empty interrupt
    sbr     temp,(1<<UDRIE1)    ;   /
    sts     UCSR1B,temp         ;   /
    pop     temp                ;restore temp
    pop     ZL                  ;restore Z
    pop     ZH                  ;   /
    pop     temp2               ;restore status register (including interrupt bit)
    out     SREG,temp2          ;   /
    pop     temp2               ;restore temp2
    ret


;send one byte from FIFO buffer to M201
U1sendbyte:
    push_sreg
    ldi     ZH,high(u1txbuf)    ;Init Z pointer to buffer in SRAM
    ldi     ZL,low(u1txbuf)     ;   /
    mov     temp,U1TXBUF        ;move U1TXBUF to temp
    andi    temp,0xF0           ;mask lower nibble (end)
    swap    temp                ;swap nibbles
    add     ZL,temp             ;add counter value to pointer
    ld      temp,Z              ;load indirect
    sts     UDR1,temp           ;Send to M201
    ldi     temp,0x10           ;load 0x10 to temp
    add     U1TXBUF,temp        ;increment start counter
    lds     temp,UCSR1A         ;Load UCSR1A to temp
    sbr     temp,(1<<TXC1)      ;Clear transmit complete bit
    sts     UCSR1A,temp         ;Write UCSR1A
    pop_sreg
    ret


;send byte from FIFO buffer to PC (interrupt driven)
U0transmit_int:
    push_sreg
    disable_sleep
    cp      XL,YL               ;compare start and end pointer
    cpc     XH,YH               ;   /
    brne    U0notdone           ;branch if not equal
    in      temp,UCSR0B         ;Disable USART0 Data Register Empty interrupt
    cbr     temp,(1<<UDRIE0)    ;   /
    out     UCSR0B,temp         ;   /
    pop_sreg
    reti
U0notdone:
    movw    ZL,XL               ;copy X (start pointer) to Z
    subi    ZL,low(memend)      ;if not at the end of memory...
    sbci    ZH,high(memend)     ;   /
    brne    U0tx_ok             ;then branch
    ldi     XL,low(memstart)    ;set start pointer to beginning of buffer
    ldi     XH,high(memstart)   ;   /
U0tx_ok:
    ld      temp,X+             ;load indirect and post-inc
    out     UDR0,temp           ;Send to PC
    sbi     UCSR0A,TXC0         ;Clear transmit complete bit
    pop_sreg
    reti


;send byte from FIFO buffer to M201 (interrupt driven)
U1transmit_int:
    push_sreg
    disable_sleep
```

```
    mov     temp,U1TXBUF        ;move U1TXBUF to temp
    mov     temp2,temp          ;move temp to temp2
    swap    temp2               ;swap nibbles
    cpse    temp,temp2          ;compare start and end
    rjmp    U1notdone           ;skip this if equal
    lds     temp,UCSR1B         ;Disable USART1 Data Register Empty interrupt
    cbr     temp,(1<<UDRIE1)    ;   /
    sts     UCSR1B,temp         ;   /
    pop_sreg
    reti
U1notdone:
    rcall   U1sendbyte          ;send byte to M201
    pop_sreg
    reti


;error handling (infinite loop), error code is in temp
error:
    clr     temp2               ;Clear PORTA
    out     PORTA,temp2         ;   /
    ldi     buf,high(3071)      ;Set USART0 (PC) baud rate to 300bps
    ldi     temp2,low(3071)
    sts     UBRR0H,buf
    out     UBRR0L,temp2
error_loop:
    rcall   debounce            ;sleep for 50 ms
    in      temp2,PINC          ;Load PINC
    ldi     buf,(1<<LED)        ;set LED flag in buf
    eor     temp2,buf           ;invert LED
    out     PORTC,temp2         ;Write to PORTC
    out     UDR0,temp           ;Send error code to PC
    sbic    UCSR0A,RXC0         ;If USART0 receive complete bit is not set...
    rjmp    error_readcmd       ;...then skip this
    rjmp    error_loop          ;loop
error_readcmd:
    in      buf,UDR0            ;Load receive buffer to buf
    cpi     buf,0xA0            ;if received byte != 0xA0...
    brne    error_loop          ;...then loop
    rjmp    cmd_reset           ;branch
```

Lisa D SPEKTROMEETRI JUHTPROGRAMMI LÄHTETEKSTID

D.1 linuxserial.cpp

```
/* linuxserial.cpp                  (C) Joel Kuusk, 2003-2005
 *
 * Communication with serial port
 */

#include "linuxserial.h"

#define BUFSIZE 4096

int BaudRate[] = { 9600, 4800, 2400, 1200, 600, 300 };

int fd;
struct termios oldtio, newtio;
unsigned char serial_buf[BUFSIZE];
int buf_start, buf_end;
fd_set input, output;
struct timeval timeout;


//Retrun true if RTS is on and false if it is off
bool getRTS()
{
  int status;

  ioctl(fd, TIOCMGET, &status);

  return TIOCM_RTS & status ? false : true;
}

// true - to controller
// false - to M201
void setRTS(bool state)
{
  unsigned long i = TIOCM_RTS;

  // return if we don't have to change anything
  if (state == getRTS())
    return;

  tcdrain(fd);
  ioctl(fd, state ? TIOCMBIC : TIOCMBIS, &i);
}

void openPort(int baud, const char *port)
{
  unsigned char baudrate;

  switch (baud)
  {
    case 9600:
      baudrate = B9600;
      break;
    case 4800:
      baudrate = B4800;
      break;
    case 2400:
      baudrate = B2400;
      break;
    case 1200:
      baudrate = B1200;
      break;
    case 600:
      baudrate = B600;
```

```
      break;
    case 300:
      baudrate = B300;
      break;
    default:
      baudrate = B9600;
      break;
  }

  fd = open(port, O_RDWR | O_NOCTTY | O_NONBLOCK);
  // read will return imediately and it is a tty

  // if we can't open serial port then there is no
  // point in running program any further
  if (fd < 0)
  {
    perror(port);
    fprintf(stderr, "\n%s: serial port open failed.\n",
            __PRETTY_FUNCTION__);
    exit(-1);
  }

  tcgetattr(fd, &oldtio);
  bzero(&newtio, sizeof(newtio));
  newtio.c_cflag = baudrate | CS8 | CLOCAL | CREAD;
  newtio.c_iflag = IGNBRK | IGNPAR;
  newtio.c_oflag = 0;
  newtio.c_lflag = 0;
  newtio.c_cc[VMIN] = 0;
  newtio.c_cc[VTIME] = 0;
  tcflush(fd, TCIFLUSH);
  tcsetattr(fd, TCSANOW, &newtio);
  buf_start = 0;
  buf_end = 0;
  fprintf(stdout, "Serial port opened with baud rate %dbps.\n", baud);

  emptyInputBuf();
}

int serialRead(unsigned char *buf, int count, int timeout_sec)
{
  int n;

  // set timeout
  timeout.tv_sec = timeout_sec;
  timeout.tv_usec = 0;

  // set up input descriptor
  FD_ZERO(&input);
  FD_SET(fd, &input);

  n = select(fd + 1, &input, NULL, NULL, &timeout);

  // select failed, some serious error must have ocurred
  if (n < 0)
  {
    perror("select");
    throw eSerialError();
  }
  // timeout
  else if (n == 0)
  {
    // Only print error message if we have default timeout.
    // On special occasions we can suppress error message
    // by specifying different timeout value
    if (timeout_sec == READTIMEOUT)
      fprintf(stderr,
              "\n%s: could not read from serial port for %d seconds.\n",
```

```
            __PRETTY_FUNCTION__, timeout_sec);
    throw eSerialReadTimeout();
  }
  // data in receive buffer
  else
  {
    n = read(fd, buf, count);

    return n;
  }
}

unsigned char serialRead(int timeout_sec)
{
  unsigned char c;

  serialRead(&c, 1, timeout_sec);

  return c;
}

// Empty serial port's receive buffer
void emptyInputBuf(void)
{
  usleep(100000);
  tcflush(fd, TCIFLUSH);
}

void serialWrite(unsigned char *buf, int count, int timeout_sec)
{
  int n;

  // repeat until all data is transferred
  while (count > 0)
  {
    // set timeout (we must reset it after every select() call)
    timeout.tv_sec = timeout_sec;
    timeout.tv_usec = 0;

    // set up output descriptor
    FD_ZERO(&output);
    FD_SET(fd, &output);

    n = select(fd + 1, NULL, &output, NULL, &timeout);

    // select failed, some serious error must have ocurred
    if (n < 0)
    {
      perror("select");
      throw eSerialError();
    }
    // timeout
    else if (n == 0)
    {
      fprintf(stderr,
              "\n%s: could not write to serial port for %d seconds.\n",
              __PRETTY_FUNCTION__, timeout_sec);
      throw eSerialWriteTimeout();
    }
    // transmit buffer is empty
    else
    {
      n = write(fd, buf, count);
      count -= n;
      buf += n;
    }
  }
}
```

```
void serialWrite(unsigned char c, int timeout_sec)
{
  serialWrite(&c, 1, timeout_sec);
}

void closePort(void)
{
  if (fd >= 0)
  {
    tcdrain(fd);
    close(fd);
  }
}
```

D.2 m201.cpp

```
/* m201.cpp                  (C) Joel Kuusk, 2003-2005
 *
 * Communication with Lawson M201
 */

#include "m201.h"

M201::M201(void)
{
  buildModeWords();
}

void M201::signOn(void)
{
  int i;
  bool permission;
  unsigned char c;

  //get current RTS state
  bool state = getRTS();

  setRTS(false);             //to M201

  permission = false;

  for (i = 0; i < 10; i++)
  {
    serialWrite(0x00);

    try
    {
      // 1 second timeout
      c = serialRead(1);

      if (c == 0x03 || c == 0x80)
      {
        printf("Lawson woke up.\n");
        permission = true;
        break;
      }
      else
        fprintf(stderr,
                "Got 0x%.2X instead of 0x03 or 0x80 from Lawson.\n", c);
    }
    catch(eSerialReadTimeout)
    {
      fprintf(stderr, "Did not get answer from Lawson for %d. time.\n",
              i + 1);
    }
  }
```

```
  if (!permission)
  {
    fprintf(stderr, "%s: could not contact Lawson.\n",
            __PRETTY_FUNCTION__);
    throw eM201Error();
  }

  usleep(200000);

  // sign on
  serialWrite(0x88);

  usleep(100000);

  // restore RTS state
  setRTS(state);
}


void M201::setBaudRate(unsigned char baud)
{
  unsigned char c;

  //get current RTS state
  bool state = getRTS();

  setRTS(false);             //to M201

  //set baud rate
  serialWrite(baud);

  if ((c = serialRead()) != baud)
  {
    fprintf(stderr, "%s: baud rates does not match.\n",
            __PRETTY_FUNCTION__);
    throw eM201Error();
  }
  else
    printf("Lawson's baud rate set to %dbps.\n", BaudRate[baud]);

  // restore RTS state
  setRTS(state);
}

void M201::echoTest(void)
{
  int i;
  unsigned char c, ch;

  //get current RTS state
  bool state = getRTS();

  setRTS(false);             //to M201

  for (i = 0; i < 10; i++)
  {
    c = random() & 0xFF;

    serialWrite(c);

    if (c != (ch = serialRead()))
    {
      fprintf(stderr, " 0x%.2X != 0x%.2X\n", c, ch);
      fprintf(stderr, "%s: Lawson's echo test failed.\n",
              __PRETTY_FUNCTION__);
      throw eM201Error();
    }
```

```
  }
  printf("Lawson's echo test passed.\n");

  // stop echo
  serialWrite(0x00);

  // restore RTS state
  setRTS(state);
}

void M201::initMode(void)
{
  bool modeok;
  unsigned char c;

  //get current RTS state
  bool state = getRTS();

  setRTS(false);             //to M201

  // send mode words
  sendPacket(modereghi, moderegmid);
  sendPacket(modereglo, PLACEHOLDER);
  sendPacket(M201_AVERAGE, parms.filter);
  sendPacket(PLACEHOLDER, 0x01);    //0x01 - polled mode

  usleep(200000);

  modeok = true;

  try
  {
    if (((c = serialRead()) & 0x1F) != (modereghi & 0x1F))
      modeok = false;
    if (c = serialRead() != moderegmid)
      modeok = false;
    if (c = serialRead() != modereglo)
      modeok = false;
  }
  catch(eSerialReadTimeout)
  {
    fprintf(stderr, "\n%s: Lawson did not respond.\n",
            __PRETTY_FUNCTION__);
    throw eM201Error();
  }

  if (!modeok)
  {
    fprintf(stderr, "\n%s: Lawson returned wrong byte.\n",
            __PRETTY_FUNCTION__);
    throw eM201Error();
  }

  // restore RTS state
  setRTS(state);
}

void M201::sendPacket(unsigned char param1, unsigned char param2)
{
  unsigned char packet[3];

  packet[0] = param1;
  packet[1] = param2;
  packet[2] = (param1 + param2) & 0xFF;

  serialWrite(packet, 3);
}
```

```
void M201::sysCal(bool verbose)
{
  int i;
  unsigned char c[4], original_gain, original_channel;

  original_gain = parms.gain;
  original_channel = parms.channel;

  //get current RTS state
  bool state = getRTS();

  setRTS(false);             // to M201

  if (parms.gain)
  {
    parms.gain = 0;
    modereghi = parms.gain << 2;
    setMode();
  }

  // offset calibration
  parms.channel = 7;
  sendPacket(0x82, parms.channel << 4); // suppose there is no extension

  // discard return code
  i = 0;
  do
    i += serialRead(c + i, parms.wordcount + 1 - i);
  while (i < parms.wordcount + 1);

  // full scale calibration
  parms.channel = 6;
  sendPacket(0x83, parms.channel << 4); // suppose there is no extension

  // discard return code
  i = 0;
  do
    i += serialRead(c + i, parms.wordcount + 1 - i);
  while (i < parms.wordcount + 1);

  if (parms.gain != original_gain)
  {
    parms.gain = original_gain;
    modereghi = parms.gain << 2;
    setMode();

    // offset calibration
    parms.channel = 7;
    sendPacket(0x82, parms.channel << 4);   // suppose there is no extension

    // discard return code
    i = 0;
    do
      i += serialRead(c + i, parms.wordcount + 1 - i);
    while (i < parms.wordcount + 1);
  }

  // restore channel selection
  setChannel(original_channel, verbose);

  usleep(200000);

  if (verbose)
    printf("System calibration done.\n");

  // restore RTS state
  setRTS(state);
}
```

```
void M201::buildModeWords(void)
{
  int modecode;

  modereghi = parms.gain << 2;  //gain, normal operation;
  modecode = (int) (ADCLOCK / parms.samplingrate);
  modereglo = modecode & 0xFF;
  moderegmid = (modecode & 0x700) >> 8;

  if (parms.wordcount == 3)
    moderegmid |= 0x80;

  if (M201_UNIPOLAR)
    moderegmid |= 0x10;
}

void M201::setMode(void)
{
  bool modeok;
  unsigned char c;

  //get current RTS state
  bool state = getRTS();

  setRTS(false);             // to M201

  sendPacket(0x84, PLACEHOLDER);

  if ((c = serialRead()) != 0x84)
  {
    fprintf(stderr, "%s: Lawson returned wrong byte (0x%.2X).\n",
            __PRETTY_FUNCTION__, c);
    throw eM201Error();
  }

  usleep(20000);

  sendPacket(modereghi, moderegmid);
  sendPacket(modereglo, PLACEHOLDER);


  modeok = true;

  if ((serialRead() & 0x1F) != (modereghi & 0x1F))
    modeok = false;

  if (serialRead() != moderegmid)
    modeok = false;

  if (serialRead() != modereglo)
    modeok = false;

  if (!modeok)
  {
    fprintf(stderr, "%s: setmode failed.\n", __PRETTY_FUNCTION__);
    throw eM201Error();
  }

  // restore RTS state
  setRTS(state);
}

void M201::setChannel(unsigned char channel_number, bool verbose)
{
  if (parms.channel == channel_number)
    return;
```

```
  //get current RTS state
  bool state = getRTS();

  setRTS(false);             // to M201

  parms.channel = channel_number;

  // set channel
  // suppose there is no extension so optically isolated outputs are zeroed
  sendPacket(0x01, parms.channel << 4);

  //wait for settling
  if (parms.filter == 0)
    if (parms.wordcount == 2)
      usleep(3000000);
    else
      usleep(4300000);
  else if (parms.filter == 1)
    if (parms.wordcount == 2)
      usleep(2 * 300000);
    else
      usleep(2 * 430000);
  else if (parms.wordcount == 2)
    usleep(3 * 30000);
  else
    usleep(3 * 43000);

  if (verbose)
    printf("Channel %d selected.\n", parms.channel);

  // restore RTS state
  setRTS(state);
}

void M201::standby(void)
{
  unsigned char c;

  //get current RTS state
  bool state = getRTS();

  setRTS(false);             // to M201

  modereghi |= 0x01;         //set standby mode
  setMode();

  // restore RTS state
  setRTS(state);
}

void M201::wake(void)
{
  unsigned char c;

  //get current RTS state
  bool state = getRTS();

  setRTS(false);             // to M201

  modereghi &= 0xFE;         //set normal mode
  setMode();

  // restore RTS state
  setRTS(state);
}

void M201::zeroDigitalOutput(void)
{
```

```
  //get current RTS state
  bool state = getRTS();

  setRTS(false);             // to M201

  sendPacket(0x02, 0);       // zero digital output

  // restore RTS state
  setRTS(state);
}

int M201::getVersion(void)
{
  unsigned char c;

  //get current RTS state
  bool state = getRTS();

  setRTS(false);             // to M201

  sendPacket(0x86, PLACEHOLDER);

  if ((c = serialRead()) != 0x86)
  {
    fprintf(stderr, "%s: Lawson returned wrong byte (0x%.2X).\n",
            __PRETTY_FUNCTION__, c);
    throw eM201Error();
  }

  // restore RTS state
  setRTS(state);

  return (int) serialRead();
}

M201::~M201(void)
{
}
```

D.3 controller.cpp

```
/* controller.cpp                  (C) Joel Kuusk, 2003-2005
 *
 * Communication with spectrometer's controller
 */

#include "controller.h"
namespace dislin
{
#include <dislin.h>
}

// convert position in steps to position in wavelength
double stepsToWl(int steps)
{
  double wl, correctedwl;

  wl = MINWL + (steps - MINWLPOS) * STEPWL;

  correctedwl = wl * LEVERA / (2 * (LEVERA + DELTAA)) +
          GRATING * sin(asin(wl * 1e-9 * LEVERA /
                  (2 * GRATING * (LEVERA + DELTAA)))
          + EPSILON) * 1e9 + SHIFT;

  return correctedwl;
}
```

```
// convert position in wavelength to position in steps
int wlToSteps(double wl)
{
  double x, s, correctedwl;

  x = (wl - SHIFT) * 1e-9 / (2 * GRATING) -
          sqrt(pow((wl - SHIFT) * 1e-9 / (2 * GRATING), 2) -
          1 / (2 * (1 + cos(EPSILON))) *
          (pow((wl - SHIFT) * 1e-9 / GRATING, 2) - pow(sin(EPSILON), 2)));

  s = x * (LEVERA + DELTAA);

  correctedwl = 2 * GRATING * s * 1e9 / LEVERA;

  return (int) rint((correctedwl - MINWL) / STEPWL) + MINWLPOS;
}

controller::controller(void)
{
  lawson = new M201();
}

void controller::signOn(void)
{
  int i;
  unsigned char c, ch, speed[2];
  double scanfreq;

  openPort(300, parms.port.c_str());

  setRTS(true);              //to controller

  // echo test
  for (i = 0; i < 10; i++)
  {
    c = random() & 0xFF;

    serialWrite(CMD_U0ECHOTEST);
    serialWrite(c);

    try
    {
      if (c != (ch = serialRead()))
      {
        fprintf(stderr, "\n 0x%.2X != 0x%.2X\n", c, ch);
        fprintf(stderr, "%s: controller's echo test error.\n",
                __PRETTY_FUNCTION__);
        throw eSerialError();
      }
    }
    catch(eSerialReadTimeout)
    {
      fprintf(stderr, "\n%s: controller does not respond to echo test.\n",
              __PRETTY_FUNCTION__);
      fprintf(stderr, "Check cables, power, serial port settings, etc.\n");
      throw eControllerError();
    }
  }

  printf("Controller's echo test passed.\n");

  if (BaudRate[parms.PC_baud] != 300)
  {
    // set controller's baudrate
    serialWrite(CMD_U0BAUDRATE);
    serialWrite(ubrr_high(BaudRate[parms.PC_baud]));
    serialWrite(ubrr_low(BaudRate[parms.PC_baud]));
```

```
  // reopen port
  closePort();
  openPort(BaudRate[parms.PC_baud], parms.port.c_str());

  setRTS(true);            //to controller

  // echo test
  for (i = 0; i < 10; i++)
  {
    c = random() & 0xFF;

    serialWrite(CMD_U0ECHOTEST);
    serialWrite(c);

    if (c != (ch = serialRead()))
    {
      fprintf(stderr, "\n0x%.2X != 0x%.2X\n", c, ch);
      fprintf(stderr, "%s: controller's echo test failed.\n",
              __PRETTY_FUNCTION__);
      throw eSerialError();
    }
  }
  printf("Controller's echo test passed.\n");
}

//set Lawson's baud rate
serialWrite(CMD_U1BAUDRATE);
serialWrite(ubrr_high(300));
serialWrite(ubrr_low(300));

printf("Controller's initialization done.\n");

// initialize M201
lawson->signOn();

// change Lawson's baudrate
setM201BaudRate();

usleep(200000);

lawson->echoTest();
lawson->initMode();

//change wordcount (only in controller)
serialWrite(CMD_SETWORDCOUNT);
serialWrite(low(parms.wordcount));

printf("Lawson's sign-on done.\n");

lawson->zeroDigitalOutput();  //reduce power consumption
lawson->sysCal();

// measure M201's speed
serialWrite(CMD_M201SPEED);

//read M201's response (we don't need that);
for (i = 0; i < (parms.wordcount + 1); i++)
  serialRead();

try
{
  // read two bytes from controller, high byte first
  i = 0;

  do
    i += serialRead(speed + i, 2 - i);
  while (i < 2);
}
```

```
// if we can’t read anything, then the culprit is
// probably M201, not serial link
catch(eSerialReadTimeout)
{
  throw eM201Error();
}

// put M201 to standby mode
lawson->standby();

scanfreq = F_OSC / (256 * (double) ((speed[0] << 8) + speed[1]));

// check for scanning speed, leave 1% error margin
if (parms.measfreq > scanfreq * parms.meassteps * 0.99)
{
  //if possible, change stepping frequency so, that other
  //parameters remain the same
  parms.measfreq = (int) floor(scanfreq * parms.meassteps * 0.99);

  if ((int) (F_OSC / (64.0 * parms.measfreq)) <= 0xFFFF)
  {
    if (parms.measfreq < parms.measfreq0)
      parms.measfreq0 = parms.measfreq;
  }
  else
  {
    parms.measfreq0 = 250; // Hz
    parms.measfreq = 250;  // Hz
    parms.M201_baud = 5;   // 300bps
    parms.wordcount = 2;   // 16 bit
    parms.meassteps = 50;  // 50 steps = 10 nanometers resolution
  }

  printf("\nStepping frequency too high for given sampling frequency, \n");
  printf("Lawson’s baud rate, and word count settings.\n");
  printf("Applying following settings:\n");
  printf("measfreq0 = %d\n", parms.measfreq0);
  printf("measfreq = %d\n", parms.measfreq);
  printf("meassteps = %d\n", parms.meassteps);
  printf("M201_baud = %d (%d bps)\n", parms.M201_baud,
         BaudRate[parms.M201_baud]);
  printf("wordcount = %d\n\n", parms.wordcount);
}
else if (parms.measfreq0 > parms.measfreq)
{
  parms.measfreq0 = parms.measfreq;

  printf("\nmeasfreq0 > measfreq. Applying following settings:\n");
  printf("measfreq0 = %d\n", parms.measfreq0);
  printf("measfreq = %d\n\n", parms.measfreq);
}

//set ADC sampling rate
parms.samplingrate =
        (int) ceil(parms.measfreq / (parms.meassteps *
                M201_SAMPLINGCOEF));
if (parms.samplingrate < 10)
  parms.samplingrate = 10;
else if (parms.samplingrate > 1027)
  parms.samplingrate = 1027;

lawson->buildModeWords();
lawson->setMode();

serialWrite(CMD_CHANGESTEP);
//set initial stepping frequency while measuring
serialWrite(stepfreq_high(parms.measfreq0));
serialWrite(stepfreq_low(parms.measfreq0));
```

```
  //set stepping frequency while measuring
  serialWrite(stepfreq_high(parms.measfreq));
  serialWrite(stepfreq_low(parms.measfreq));
  //set initial transport stepping frequency
  serialWrite(stepfreq_high(parms.transpfreq0));
  serialWrite(stepfreq_low(parms.transpfreq0));
  //set transport stepping frequency
  serialWrite(stepfreq_high(parms.transpfreq));
  serialWrite(stepfreq_low(parms.transpfreq));
  //set acceleration
  serialWrite(parms.dstepsize);
  //set resolution (number of steps between measurements)
  serialWrite(low(parms.measssteps));
  //set manual stepping flag
  serialWrite(parms.manualstep & 0xff);

  printf("Controller's parameters set.\n");

  initPosition();

}

void controller::initPosition(void)
{
  unsigned char c;

  //get current RTS state
  bool state = getRTS();

  setRTS(true);              //to controller

  printf("\nInitializing minimum wavelength position...");
  fflush(stdout);

  //step to min wavelength
  privateStepBackward(0, false);

  //initialize min wavelength position
  serialWrite(CMD_INIT);

  try
  {
    // wait for response (already at minimum wavelength)
    c = serialRead(TIMEOUTBUF);
  }
  catch(eSerialReadTimeout)
  {
    fprintf(stderr, "\n%s: controller did not respond in %d seconds.\n",
            __PRETTY_FUNCTION__, TIMEOUTBUF);
    throw eControllerError();
  }

  if (c != 0xF0)             //min_wl
  {
    fprintf(stderr, "%s: controller returned wrong byte (0x%.2X).\n",
            __PRETTY_FUNCTION__, c);
    throw eControllerError();
  }

  //step forward 20 steps
  privateStepForward((int) (20 * STEPWL), false);

  //initialize min wavelength position
  serialWrite(CMD_INIT);

  try
  {
    // wait for response (stepping speed 100 Hz)
```

```
    c = serialRead((int) (20.0 / 100.0 + TIMEOUTBUF));
  }
  catch(eSerialReadTimeout)
  {
    fprintf(stderr, "\n%s: controller did not respond in %d seconds.\n",
            __PRETTY_FUNCTION__,
            (int) (20.0 / parms.transpfreq + TIMEOUTBUF));
    throw eControllerError();
  }

  if (c == 0xF0)             //min_wl
    printf("done.\n\n");
  else
  {
    fprintf(stderr, "%s: controller returned wrong byte (0x%.2X).\n",
            __PRETTY_FUNCTION__, c);
    throw eControllerError();
  }

  //restore RTS state
  setRTS(state);
}

void controller::goTo(double wavelength, bool verbose)
{
  int pos;
  double curpos;
  unsigned char c;

  if (wavelength < MINWL || wavelength > MAXWL)
  {
    printf("\nWavelength must be in the range [%d..%d].\n\n",
            MINWL, MAXWL);
    return;
  }

  //get current RTS state
  bool state = getRTS();

  setRTS(true);              //to controller

  //get current position
  curpos = getPosition();

  //calcualte distance in steps
  curpos -= wavelength;
  if (curpos < 0)
    curpos *= -1;
  curpos /= STEPWL;

  //convert from wavelength to step
  pos = wlToSteps(wavelength);

  serialWrite(CMD_GOTO);
  serialWrite(high(pos));
  serialWrite(low(pos));

  if (verbose)
  {
    printf("\nStepping to %.1f nm...", wavelength);
    fflush(stdout);
  }

  try
  {
    // wait for respomse
    c = serialRead((int) (curpos / parms.transpfreq + TIMEOUTBUF));
  }
```

```
  catch(eSerialReadTimeout)
  {
    fprintf(stderr, "\n%s: controller did not respond in %d seconds.\n",
            __PRETTY_FUNCTION__,
            (int) (curpos / parms.transpfreq + TIMEOUTBUF));
    throw eControllerError();
  }

  //return if aborted
  if (aborted)
    throw eAborted();

  if (verbose)
  {
    if (c == 0x05)
      printf("done.\n\n");
    else if (c == 0xF0)
      printf("reached minimum wavelength.\n\n");
    else if (c == 0xF1)
      printf("reached maximum wavelength.\n\n");
  }

  if (c != 0x05 && c != 0xF0 && c != 0xF1)
  {
    fprintf(stderr, "%s: controller returned wrong byte (0x%.2X).\n",
            __PRETTY_FUNCTION__, c);
    throw eControllerError();
  }

  //restore RTS state
  setRTS(state);
}


// *** void controller::privateStepForward(int stepcount) ***
// step forward stepcount steps.
// if stepcount == 0, then step until
// maximum wavelength is reached

void controller::privateStepForward(double nanometers, bool verbose)
{
  unsigned char c;
  int stepcount;

  //get current RTS state
  bool state = getRTS();

  setRTS(true);              //to controller

  //convert from nanometers to steps
  stepcount = (int) rint(nanometers / STEPWL);

  serialWrite(CMD_FORWARD);
  serialWrite(CMD_START);
  serialWrite(high(stepcount));
  serialWrite(low(stepcount));

  if (verbose)
  {
    if (stepcount)
      printf("\nStepping %.0f nanometers forward...", nanometers);
    else
      printf("\nStepping to maximum wavelength...");

    fflush(stdout);
  }

  if (!stepcount)
```

```
    stepcount = (int) ((MAXWL - MINWL) / STEPWL);

  try
  {
    // wait for response
    c = serialRead((int) ((double) stepcount / parms.transpfreq +
                  TIMEOUTBUF));
  }
  catch(eSerialReadTimeout)
  {
    fprintf(stderr, "\n%s: controller did not respond in %d seconds.\n",
            __PRETTY_FUNCTION__,
            (int) ((double) stepcount / parms.transpfreq + TIMEOUTBUF));
    throw eControllerError();
  }

  //return if aborted
  if (aborted)
    throw eAborted();

  if (verbose)
    if (c == 0x01)
      printf("done.\n\n");
    else if (c == 0xF1)
      printf("reached maximum wavelength.\n\n");

  if (c != 0x01 && c != 0xF1)
  {
    fprintf(stderr, "%s: controller returned wrong byte (0x%.2X).\n",
            __PRETTY_FUNCTION__, c);
    throw eControllerError();
  }

  //restore RTS state
  setRTS(state);
}


void controller::stepForward(double nanometers, bool verbose)
{
  //stepping to the end
  if (!nanometers)
    // if manual stepping is disabled then don't allow
    // stepping to the end
    if (!parms.manualstep)
      return;
    else
      privateStepForward(nanometers, verbose);
  else
    goTo(getPosition() + nanometers);
}


// *** void controller::privateStepBackward(int stepcount) ***
// step backward stepcount steps.
// if stepcount == 0, then step until
// minimum wavelength is reached

void controller::privateStepBackward(double nanometers, bool verbose)
{
  unsigned char c;
  int stepcount;

  //get current RTS state
  bool state = getRTS();

  setRTS(true);              //to controller
```

```
  //convert from nanometers to steps
  stepcount = (int) rint(nanometers / STEPWL);

  serialWrite(CMD_BACKWARD);
  serialWrite(CMD_START);
  serialWrite(high(stepcount));
  serialWrite(low(stepcount));

  if (verbose)
  {
    if (stepcount)
      printf("\nStepping %.0f nanometers backward...", nanometers);
    else
      printf("\nStepping to minimum wavelength...");

    fflush(stdout);
  }

  if (!stepcount)
    stepcount = (int) ((MAXWL - MINWL) / STEPWL);

  try
  {
    // wait for response
    c = serialRead((int) ((double) stepcount / parms.transpfreq +
                    TIMEOUTBUF));
  }
  catch(eSerialReadTimeout)
  {
    fprintf(stderr, "\n%s: controller did not respond in %d seconds.\n",
            __PRETTY_FUNCTION__,
            (int) ((double) stepcount / parms.transpfreq + TIMEOUTBUF));
    throw eControllerError();
  }

  //return if aborted
  if (aborted)
    throw eAborted();

  if (verbose)
    if (c == 0x01)
      printf("done.\n\n");
    else if (c == 0xF0)
      printf("reached minimum wavelength.\n\n");

  if (c != 0x01 && c != 0xF0)
  {
    fprintf(stderr, "%s: controller returned wrong byte (0x%.2X).\n",
            __PRETTY_FUNCTION__, c);
    throw eControllerError();
  }

  //restore RTS state
  setRTS(state);
}


void controller::stepBackward(double nanometers, bool verbose)
{
  //stepping to the end
  if (!nanometers)
    // if manual stepping is disabled then don't allow
    // stepping to the end
    if (!parms.manualstep)
      return;
    else
      privateStepBackward(nanometers, verbose);
  else
```

```
    goTo(getPosition() - nanometers);
}

void controller::stop(void)
{
  serialWrite(CMD_STOP);

  usleep(300000);
  emptyInputBuf();
}

double controller::getPosition(void)
{
  unsigned char pos[2];
  int i;

  //get current RTS state
  bool state = getRTS();

  setRTS(true);              //to controller

  serialWrite(CMD_GETPOS);

  try
  {
    i = 0;
    do
      i += serialRead(pos + i, 2 - i);
    while (i < 2);
  }
  catch(eSerialReadTimeout)
  {
    throw eControllerError();
  }

  //restore RTS state
  setRTS(state);

  // return current position in nanometers
  return stepsToWl((pos[0] << 8) + pos[1]);
}

void controller::getConversion(double startwl, double endwl)
{
  int i, j, jmax, pos, endpos;
  double stepcount;
  unsigned char c;
  long rawcount;
  double result, deltawl;
  list < double >volts;
  time_t currenttime;
  struct tm *currenttimestruct;
  char outfilename[28];
  FILE *outfile;
  list < double >::const_iterator volts_iter;
  double sstep, speed, correction, dist, delay;


  if (startwl < MINWL || startwl > MAXWL || endwl < MINWL || endwl > MAXWL)
  {
    printf("\nWavelengths must be in the range [%d..%d].\n\n",
            MINWL, MAXWL);
    return;
  }

  //get current RTS state
  bool state = getRTS();
```

```
setRTS(true);              //to controller

lawson->wake();

lawson->sysCal(false);

deltawl = parms.meassteps * STEPWL;

//scanning forward
if (startwl < endwl)
{
  //step to initial wavelength, approach from the right direction
  if ((getPosition() + 2) > startwl)
  {
    //we don't use goTo(startwl-2) because startwl-2 might be out of range
    goTo(startwl, false);
    privateStepBackward(2, false);
  }
  goTo(startwl, false);
}
//scanning backward
else
{
  //step to initial wavelength, approach from the right direction
  if ((getPosition() - 2) < startwl)
  {
    //we don't use goTo(startwl+2) because startwl+2 might be out of range
    goTo(startwl, false);
    privateStepForward(2, false);
  }
  goTo(startwl, false);
}

//return if aborted
if (aborted)
  throw eAborted();

//convert from nanometers to steps
endpos = wlToSteps(endwl);

//open output file
currenttime = time(NULL);
currenttimestruct = gmtime(&currenttime);

//filename example: 30.09.2004_12:18:19GMT.spec
strftime(outfilename, 28, "%d.%m.%Y_%H:%M:%SGMT.spec",
        currenttimestruct);

if ((outfile = fopen(outfilename, "w")) == NULL)  /* open output file */
{
  printf("Can not open output file '%s' for writing.\n", outfilename);

  //restore RTS state
  setRTS(state);

  return;
}

//start scanning
serialWrite(CMD_SCAN);
serialWrite(high(endpos));
serialWrite(low(endpos));

printf("Scanning from %.1f nm to %.1f nm...    ", startwl, endwl);
fflush(stdout);

stepcount = endwl - startwl / STEPWL;
if (stepcount < 0)
```

```
  stepcount *= -1;

try
{
  // wait for response
  c = serialRead((int) (stepcount / parms.measfreq + TIMEOUTBUF));
}
catch(eSerialReadTimeout)
{
  fprintf(stderr, "\n%s: controller did not respond in %d seconds.\n",
          __PRETTY_FUNCTION__,
          (int) (stepcount / parms.measfreq + TIMEOUTBUF));

  fclose(outfile);
  throw eControllerError();
}
catch(...)
{
  fclose(outfile);
  throw;
}


if (c == 0x09)
{
  jmax = (int) floor(abs((int) (wlToSteps(endwl) -
                          wlToSteps(startwl))) / parms.meassteps) + 1;

  // take readings
  for (j = 0; j < jmax; j++)
  {
    //return if aborted
    if (aborted)
    {
      fclose(outfile);
      unlink(outfilename);
      throw eAborted();

    }

    // in case wordcount == 2
    reading[2] = 0;

    //show progress
    printf("\b\b\b\b%3d%%", 100 * j / jmax);
    fflush(stdout);

    try
    {
      // read wordcount bytes from controller
      i = 0;

      do
        i += serialRead(reading + i, parms.wordcount - i);
      while (i < parms.wordcount);
    }
    // if we can't read anything, then the culprit is
    // probably controller, not serial link
    catch(eSerialReadTimeout)
    {
      fclose(outfile);
      throw eControllerError();
    }
    catch(...)
    {
      fclose(outfile);
      unlink(outfilename);
      throw;
```

```
  }

  rawcount = reading[0] + ((unsigned long) reading[1] << 8) +
          ((unsigned long) reading[2] << 16);

  //beep if overrange
  if (reading[0] == 0xFF && reading[1] == 0xFF &&
          (parms.wordcount == 2 || reading[2] == 0xFF))
  {
    printf("\a");
    fflush(stdout);
  }

  result = rawcount * CONVERSION_FACTOR;

  if (parms.wordcount != 3) // 16 bit mode
    result *= 256;
  if (M201_UNIPOLAR == false)   // if bipolar
    result = result * 2 - 5;

  result /= pow(2, parms.gain);

  //add result to vector
  volts.push_back(result);
}

fprintf(outfile, "# %s\n\n", outfilename);
fprintf(outfile, "### Controller ###\n");
fprintf(outfile, "# PC_baud: %d\n", parms.PC_baud);
fprintf(outfile, "# measfreq0: %d\n", parms.measfreq0);
fprintf(outfile, "# dstepsize: %d\n", parms.dstepsize);
fprintf(outfile, "# measfreq: %d\n", parms.measfreq);
fprintf(outfile, "#\n");
fprintf(outfile, "### M201 ###\n");
fprintf(outfile, "# M201_baud: %d\n", parms.M201_baud);
fprintf(outfile, "# wordcount: %d\n", parms.wordcount);
fprintf(outfile, "# gain: %d\n", parms.gain);
fprintf(outfile, "# filter: %d\n", parms.filter);
fprintf(outfile, "\n");
fprintf(outfile, "# Stepping direction: %s\n\n",
        (startwl < endwl) ? "forward" : "backward");
fprintf(outfile,
        "### Values are in nanometers and millivolts ###\n\n");

//if scanning backwards then reverse list
if (startwl > endwl)
  volts.reverse();

volts_iter = volts.begin();
pos = wlToSteps(min(startwl, endwl));

sstep = (stepfreq_high(parms.measfreq0) << 8) +
        stepfreq_low(parms.measfreq0);

while (volts_iter != volts.end())
{
  //distance from start or end
  dist = min(pos - wlToSteps(min(startwl, endwl)),
          (wlToSteps(max(startwl, endwl)) - pos));

  //stepping speed at the moment of sending measurement command
  if ((sstep - dist * parms.dstepsize) >=
          ((stepfreq_high(parms.measfreq) << 8) +
                  stepfreq_low(parms.measfreq)))
    speed = F_OSC / (64 * (sstep - dist * parms.dstepsize));
  else
    speed = parms.measfreq;
```

```
  //3-byte command + 1-byte reply; each byte = 1 start + 8 data + 1 stop bits
      //plus command processing time (in seconds)
      //minus one sampling interval (due to Lawson's measuring peculiarity)
      delay = 40.0 / BaudRate[parms.M201_baud] +
              M201_ADSTARTDELAY - 1 / parms.samplingrate;

      //first measurement does not need any correction
      if (volts_iter != volts.begin())
      {
        //transfer delay over RS232
        correction = delay * speed * STEPWL;
        //signal propagation delay in analog circuit
        correction -= ADELAY;
      }
      else
        correction = 0;

      if (startwl > endwl)
        correction *= -1;

      // wavelength and millivolts in output file
      fprintf(outfile, "%1.2f\t%1.6f\n", stepsToWl(pos) + correction,
              (*volts_iter) * 1000.0);

      pos += parms.measssteps;
      volts_iter++;
    }

    printf("\b\b\b\bdone.\n");
    printf("Result was saved to file '%s'\n", outfilename);
    printf("\n");

    fclose(outfile);
  }
  else
  {
    fclose(outfile);
    unlink(outfilename);
    fprintf(stderr, "\n\n%s: cotroller returned wrong byte (0x%.2X).\n",
            __PRETTY_FUNCTION__, c);
    throw eControllerError();
  }

  lawson->standby();

  // display graph
  if (parms.showgraph)
    showGraph(&volts, startwl, endwl, deltawl);

  //restore RTS state
  setRTS(state);
}


void controller::showGraph(list < double >*volts, double startwl,
        double endwl, double deltawl)
{
  int i;
  float *graph_x, *graph_y;
  double x_min, x_max, y_min, y_max, tmp;
  float x_interval, y_interval;
  list < double >::const_iterator volts_iter;
  ostringstream titlestream;

  if ((graph_x = (float *) malloc(volts->size() * sizeof(float))) == NULL)
  {
    fprintf(stderr, "Out of memory\n");
    exit(-1);
```

```
}

if ((graph_y = (float *) malloc(volts->size() * sizeof(float))) == NULL)
{
  fprintf(stderr, "Out of memory\n");
  exit(-1);
}

volts_iter = volts->begin();
y_min = 10000;
y_max = 0;
tmp = min(startwl, endwl);
i = 0;

while (volts_iter != volts->end())
{
  graph_x[i] = tmp;
  graph_y[i] = *volts_iter * 1000.0;
  y_max = max(graph_y[i], y_max);
  y_min = min(graph_y[i], y_min);
  tmp += deltawl;
  volts_iter++;
  i++;
}

// scale x-axis
x_min = min(startwl, endwl);
x_max = max(startwl, endwl);
tmp = x_max - x_min;
i = 0;

while (tmp > 5)
{
  tmp /= 5.0;
  i++;
}

tmp = ceil(tmp);
x_interval = tmp * pow(5.0, i - 1);

x_max = floor(x_max / x_interval) * x_interval == x_max ?
        x_max : (floor(x_max / x_interval) + 1.0) * x_interval;

x_min = floor(x_min / x_interval) * x_interval;


// scale y-axis
i = 0;
tmp = y_max - y_min;

while (tmp > 5)
{
  tmp /= 10.0;
  i++;
}

tmp = ceil(tmp);
y_interval = tmp * pow(10.0, i - 1);

y_max = floor(y_max / y_interval) * y_interval == y_max ?
        y_max : (floor(y_max / y_interval) + 1.0) * y_interval;

y_min = floor(y_min / y_interval) * y_interval;

dislin::winsiz(GRAPH_WIDTH, GRAPH_HEIGHT);
dislin::setpag("da4l");    // 2970 x 2100
dislin::scrmod("revers");  //white background, black lines
dislin::clrmod("full");    //256-color mode
```

```
  dislin::x11mod("store");   //graphical output is send to a pixmap
  //that will be copied to the graphics window
  dislin::metafl("xwin");
  dislin::disini();
  dislin::winkey("escape");  //escape closes graph window
  dislin::errmod("all", "off"); //suppress error messages
  dislin::complx();          //complex font
  dislin::axspos(350, 1900); //position of axis system
  dislin::axslen(2250, 1700);   //length of axis system
  dislin::name("Wavelength, nm", "x");
  dislin::name("Voltage, mV", "y");
  dislin::namdis(50, "xy");  //distance between name and labels
  if (x_interval >= 1)
    dislin::labdig(-1, "x"); //x-axis precision
  if (y_interval >= 1)
    dislin::labdig(-1, "y"); //y-axis precision
  dislin::graf(x_min, x_max, x_min, x_interval, y_min, y_max, y_min,
          y_interval);
  dislin::height(50);        // bigger font for title
  titlestream << "gain$ = 2^" << (int) parms.gain << " = ";
  titlestream << (int) pow(2, parms.gain) << "$";
  dislin::texmod("on");      //turn on TeX mode
  dislin::titlin((char *) titlestream.str().c_str(), 4);
  dislin::title();
  dislin::texmod("off");     //turn off TeX mode
  dislin::height(36);        //change font size back to normal

  // secondary y-axis
  dislin::labdig(2, "y");    //precision
  dislin::yaxis(y_min / 5000 * pow(2, parms.gain),
          y_max / 5000 * pow(2, parms.gain),
          y_min / 5000 * pow(2, parms.gain),
          y_interval / 5000 * pow(2, parms.gain),
          1700, "ADC scale", 0, 2600, 1900);
  dislin::dotl();            //long dotted line style
  dislin::grid(1, 1);        //draw grid
  dislin::solid();           //solid line style
  dislin::color("red");
  dislin::curve(graph_x, graph_y, volts->size());

  dislin::disfin();

  free(graph_x);
  free(graph_y);
}


void controller::setM201BaudRate(void)
{
  lawson->setBaudRate(parms.M201_baud);

  //get current RTS state
  bool state = getRTS();

  setRTS(true);              //to controller

  serialWrite(CMD_U1BAUDRATE);
  serialWrite(ubrr_high(BaudRate[parms.M201_baud]));
  serialWrite(ubrr_low(BaudRate[parms.M201_baud]));

  //restore RTS state
  setRTS(state);
}


void controller::setGain(unsigned char power)
{
  parms.gain = power;
```

```
  lawson->buildModeWords();
  lawson->setMode();
}


void controller::wakeM201(void)
{
  lawson->wake();
}


void controller::standbyM201(void)
{
  lawson->standby();
}


//return current reading in millivolts
double controller::getSingleConversion(bool standby)
{
  int i;
  unsigned char c;
  long rawcount;
  double volts;

  //get current RTS state
  bool state = getRTS();

  setRTS(false);             //to M201

  if (standby)
    lawson->wake();

  //Read conversion command
  serialWrite(0x81);
  serialWrite(0x00);
  serialWrite(0x81);

  try
  {
    c = serialRead();
  }
  catch(eSerialReadTimeout)
  {
    fprintf(stderr, "\n%s: M201 did not respond in %d seconds.\n",
            __PRETTY_FUNCTION__, READTIMEOUT);
    throw eM201Error();
  }

  if (c != 0x81)
  {
    fprintf(stderr, "\n%s: M201 returned wrong byte (0x%.2X).\n",
            __PRETTY_FUNCTION__, c);
    throw eM201Error();
  }

  // in case wordcount == 2
  reading[2] = 0;

  i = 0;
  do
    i += serialRead(reading + i, parms.wordcount - i);
  while (i < parms.wordcount);

  rawcount = reading[0] + ((unsigned long) reading[1] << 8) +
          ((unsigned long) reading[2] << 16);
```

```
  volts = rawcount * CONVERSION_FACTOR;

  if (parms.wordcount != 3)  // 16 bit mode
    volts *= 256;
  if (M201_UNIPOLAR == false)   // if bipolar
    volts = volts * 2 - 5;

  volts /= pow(2, parms.gain);

  // if Lawson was in standby before then put it to standby again
  if (standby)
    lawson->standby();

  //restore RTS state
  setRTS(state);

  return volts * 1000;
}


//park scanner to longer wavelength end
void controller::park(void)
{
  privateStepForward(0, true);
}


controller::~controller(void)
{
  try
  {
    setRTS(true);            //to controller
    serialWrite(CMD_RESET);  //reset controller

    closePort();
  }
  catch(eSerialError)
  {
  }
}
```

D.4 spectrometer.cpp

```
/* spectrometer.cpp                  (C) Joel Kuusk, 2003-2005
 *
 * FIRS (Field InfraRed Scanner) user interface, main program file
 *
 * Compiling:
 * g++ -lpthread -ldislin -o spectrometer spectrometer.cpp
 * controller.cpp m201.cpp linuxserial.cpp
 */

#include "spectrometer.h"

struct parameters parms;
bool monitoring;
bool aborted;
extern int errno;
struct termios ttyoldtio;
int tty;

void printHelp(void)
{
  printf("\n\nUsage:\n");
  printf("h - print this help to the screen\n");
  printf("e - read error code from controller\n");
  printf("w - step to wavelength\n");
```

```
  printf("f - step forward\n");
  printf("b - step backward\n");
  printf("m - measure spectrum (result to file)\n");
  printf("r - reread configuration file and reset controller\n");
  printf("s - take single measurement (result to display)\n");
  printf("c - continuously monitor Lawson's reading (press any key to interrupt)\n");
  printf("g - change gain\n");
  printf("p - park scanner to longer wavelength end and quit program\n");
  printf("q - exit program and reset controller and Lawson (if possible)\n");
  printf("\n");
  return;
}

void readConf(void)
{
  int i, j;
  FILE *confi;
  char line[MAXLINE], portname[MAXLINE];

  if ((confi = fopen(CONFIFILE, "r")) == NULL)  /* open configuration file */
  {
    printf("Can not open configuration file '%s'.\n", CONFIFILE);
    printf("Default settings:\n");
    printf("port = %s\n", parms.port.c_str());
    printf("PC_baud = %d\n", parms.PC_baud);
    printf("measfreq0 = %d\n", parms.measfreq0);
    printf("dstepsize = %d\n", parms.dstepsize);
    printf("measfreq = %d\n", parms.measfreq);
    printf("M201_baud = %d\n", parms.M201_baud);
    printf("wordcount = %d\n", parms.wordcount);
    printf("gain = %d\n", parms.gain);
    printf("filter = %d\n", parms.filter);
    printf("channel = %d\n", parms.channel);
    printf("\n");
    return;
  }

  while (fgets(line, MAXLINE, confi) != NULL)
  {
    if (strncmp(line, "PC_baud", 7) == 0)
    {
      j = 0;

      // find beginning of number
      for (i = 7; !isdigit(line[i]); i++)
        if (i >= strlen(line))
        {
          printf("Error in configuration file: PC_baud value is missing.\n");
          printf("Default value: %d (%dbps).\n\n", parms.PC_baud,
                 BaudRate[parms.PC_baud]);
          j = 1;
          break;
        }

      if (j)
        continue;

      // read number
      if (!isdigit(line[i]) || atoi(&(line[i])) < 0
            || atoi(&(line[i])) > 5)
      {
        printf("Error in configuration file: PC_baud value must be in the ");
        printf("range [0..5].\n");
        printf("Default value: %d (%dbps).\n\n", parms.PC_baud,
                BaudRate[parms.PC_baud]);
      }
      else
        parms.PC_baud = atoi(&(line[i]));
```

```
}
else if (strncmp(line, "measfreq0", 9) == 0)
{
  j = 0;

  // find beginning of number
  for (i = 9; !isdigit(line[i]); i++)
    if (i >= strlen(line))
    {
      printf("Error in configuration file: measfreq0 value is missing.\n");
      printf("Default value: %dHz.\n\n", parms.measfreq0);
      j = 1;
      break;
    }

  if (j)
    continue;

  // read number
  j = atoi(&(line[i]));

  if (j && (int) (F_OSC / (64 * j)) > 0xFFFF)
  {
    printf("Error in configuration file: measfreq0 must be greater or ");
    printf("equal to %d.\n", (int) ceil(F_OSC / (64.0 * 0xFFFF)));
    printf("Default value: %dHz.\n\n", parms.measfreq0);
  }
  else
    parms.measfreq0 = j;
}
else if (strncmp(line, "transpfreq0", 11) == 0)
{
  j = 0;

  // find beginning of number
  for (i = 11; !isdigit(line[i]); i++)
    if (i >= strlen(line))
    {
      printf("Error in configuration file: transpfreq0 value is missing.\n");
      printf("Default value: %dHz.\n\n", parms.transpfreq0);
      j = 1;
      break;
    }

  if (j)
    continue;

  // read number
  j = atoi(&(line[i]));

  if (j && (int) (F_OSC / (64 * j)) > 0xFFFF)
  {
    printf("Error in configuration file: transpfreq0 must be greater or ");
    printf("equal to %d.\n", (int) ceil(F_OSC / (64.0 * 0xFFFF)));
    printf("Default value: %dHz.\n\n", parms.transpfreq0);
  }
  else
    parms.transpfreq0 = j;
}
else if (strncmp(line, "transpfreq", 10) == 0)
{
  j = 0;

  // find beginning of number
  for (i = 10; !isdigit(line[i]); i++)
    if (i >= strlen(line))
    {
      printf("Error in configuration file: transpfreq value is missing.\n");
```

```
      printf("Default value: %dHz.\n\n", parms.transpfreq);
      j = 1;
      break;
    }

  if (j)
    continue;

  // read number
  j = atoi(&(line[i]));

  if (j && (int) (F_OSC / (64 * j)) > 0xFFFF)
  {
    printf("Error in configuration file: transpfreq must be greater or ");
    printf("equal to %d.\n", (int) ceil(F_OSC / (64.0 * 0xFFFF)));
    printf("Default value: %dHz.\n\n", parms.transpfreq);
  }
  else
    parms.transpfreq = j;
}
else if (strncmp(line, "dstepsize", 9) == 0)
{
  j = 0;

  // find beginning of number
  for (i = 9; !isdigit(line[i]); i++)
    if (i >= strlen(line))
    {
      printf("Error in configuration file: dstepsize value is missing.\n");
      printf("Default value: %d.\n\n", parms.dstepsize);
      j = 1;
      break;
    }

  if (j)
    continue;

  // read number
  j = atoi(&(line[i]));

  if ((j < 0) || (j > 0xFF))
  {
    printf("Error in configuration file: dstepsize value must be in the ");
    printf("range [0..255].\n");
    printf("Default value: %d. (%d)\n\n", parms.dstepsize, j);
  }
  else
    parms.dstepsize = j;
}
else if (strncmp(line, "measfreq", 8) == 0)
{
  j = 0;

  // find beginning of number
  for (i = 8; !isdigit(line[i]); i++)
    if (i >= strlen(line))
    {
      printf("Error in configuration file: measfreq value is missing.\n");
      printf("Default value: %d Hz.\n\n", parms.measfreq);
      j = 1;
      break;
    }

  if (j)
    continue;

  // read number
  j = atoi(&(line[i]));
```

```
  if ((int) (F_OSC / (64 * j)) > 0xFFFF)
  {
    printf("Error in configuration file: measfreq value must be greater ");
    printf("or equal to %d.\n", (int) ceil(F_OSC / (64.0 * 0xFFFF)));
    printf("Default value: %d Hz.\n\n", parms.measfreq);
  }
  else
    parms.measfreq = j;
}
else if (strncmp(line, "meassteps", 9) == 0)
{
  j = 0;

  // find beginning of number
  for (i = 9; !isdigit(line[i]); i++)
    if (i >= strlen(line))
    {
      printf("Error in configuration file: meassteps value is missing.\n");
      printf("Default value: %d steps.\n\n", parms.meassteps);
      j = 1;
      break;
    }

  if (j)
    continue;

  // read number
  j = atoi(&(line[i]));

  if (j < 1 || j > 255)
  {
    printf("Error in configuration file: meassteps value must be in the ");
    printf("range [1..255]\n");
    printf("Default value: %d steps.\n\n", parms.meassteps);
  }
  else
    parms.meassteps = j;
}
else if (strncmp(line, "port", 4) == 0)
{
  j = 0;
  // find beginning of port name
  for (i = 4; line[i] != '/'; i++)
    if (i >= strlen(line))
    {
      printf("Error in configuration file: port name is missing.\n");
      printf("Default value: %s.\n\n", parms.port.c_str());
      j = 1;
      break;
    }

  if (j)
    continue;

  // read port name
  for (; line[i] != '\t' && line[i] != ' ' && line[i] != '\n'
         && line[i] != 0 && j < (MAXLINE - 1); i++, j++)
    portname[j] = line[i];

  portname[j] = 0;

  parms.port = portname;
}
else if (strncmp(line, "M201_baud", 9) == 0)
{
  j = 0;
```

```
  // find beginning of number
  for (i = 9; !isdigit(line[i]); i++)
    if (i >= strlen(line))
    {
      printf("Error in configuration file: M201_baud value is missing.\n");
      printf("Default value: %d (%dbps).\n\n", parms.M201_baud,
             BaudRate[parms.M201_baud]);
      j = 1;
      break;
    }

  if (j)
    continue;

  // read number
  if (!isdigit(line[i]) || atoi(&(line[i])) < 0
        || atoi(&(line[i])) > 5)
  {
    printf("Error in configuration file: M201_baud value must be in the ");
    printf("range [0..5].\n");
    printf("Default value: %d (%dbps).\n\n", parms.M201_baud,
           BaudRate[parms.M201_baud]);
  }
  else
    parms.M201_baud = atoi(&(line[i]));
}
else if (strncmp(line, "channel", 7) == 0)
{
  j = 0;

  // find beginning of number
  for (i = 7; !isdigit(line[i]); i++)
    if (i >= strlen(line))
    {
      printf("Error in configuration file: channel number is missing.\n");
      printf("Default value: %d.\n\n", parms.channel);
      j = 1;
      break;
    }

  if (j)
    continue;

  // read number
  if (!isdigit(line[i]) || atoi(&(line[i])) < 0
        || atoi(&(line[i])) > 7)
  {
    printf("Error in configuration file: channel value must be in the ");
    printf("range [0..7].\n");
    printf("Default value: %d\n\n", parms.channel);
  }
  else
    parms.channel = atoi(&(line[i]));
}
else if (strncmp(line, "gain", 4) == 0)
{
  j = 0;

  for (i = 4; !isdigit(line[i]); i++)
    if (i >= strlen(line))
    {
      printf("Error in configuration file: gain value is missing.\n");
      printf("Default value: %d.\n\n", parms.gain);
      j = 1;
      break;
    }

  if (j)
```

```
    continue;

  // read number
  if (!isdigit(line[i]) || atoi(&(line[i])) < 0
        || atoi(&(line[i])) > 7)
  {
    printf("Error in configuration file: gain value must be in the ");
    printf("range [0..7].\n");
    printf("Default value: %d.\n\n", parms.gain);
  }
  else
    parms.gain = line[i] - '0';
}
else if (strncmp(line, "filter", 6) == 0)
{
  j = 0;

  for (i = 6; !isdigit(line[i]); i++)
    if (i >= strlen(line))
    {
      printf("Error in configuration file: filter value is missing.\n");
      printf("Default value: %d.\n\n", parms.filter);
      j = 1;
      break;
    }

  if (j)
    continue;

  if (!isdigit(line[i]) || atoi(&(line[i])) < 0
        || atoi(&(line[i])) > 2)
  {
    printf("Error in configuration file: filter value must be in the ");
    printf("range [0..2]\n");
    printf("Default value: %d.\n\n", parms.filter);
  }
  else
    parms.filter = atoi(&(line[i]));
}
else if (strncmp(line, "wordcount", 9) == 0)
{
  j = 0;

  for (i = 9; !isdigit(line[i]); i++)
    if (i >= strlen(line))
    {
      printf("Error in configuration file: wordcount value is missing.\n");
      printf("Default value: %d.\n\n", parms.wordcount);
      j = 1;
      break;
    }

  if (j)
    continue;

  if (!isdigit(line[i]) || atoi(&(line[i])) < 2
        || atoi(&(line[i])) > 3)
  {
    printf("Error in configuration file: wordcount value must be 2 or 3.\n");
    printf("Default value: %d.\n\n", parms.wordcount);
  }
  else
    parms.wordcount = atoi(&(line[i]));
}
else if (strncmp(line, "showgraph", 9) == 0)
{
  j = 0;
```

```
      for (i = 9; !isdigit(line[i]); i++)
        if (i >= strlen(line))
        {
          printf("Error in configuration file: showgraph value is missing.\n");
          printf("Default value: %s.\n\n",
                  parms.showgraph ? "Show" : "Don't show");
          j = 1;
          break;
        }

      if (j)
        continue;

      if (!isdigit(line[i]) || atoi(&(line[i])) < 0
              || atoi(&(line[i])) > 1)
      {
        printf("Error in configuration file: showgraph value must be 0 or 1.\n");
        printf("Default value: %s.\n\n",
                parms.showgraph ? "Show" : "Don't show");
      }
      else
        parms.showgraph = atoi(&(line[i])) ? true : false;
    }
    else if (strncmp(line, "manualstep", 10) == 0)
    {
      j = 0;

      for (i = 10; !isdigit(line[i]); i++)
        if (i >= strlen(line))
        {
          printf("Error in configuration file: manualstep value is missing.\n");
          printf("Default value: %s.\n\n",
                  parms.manualstep ? "Enabled" : "Disabled");
          j = 1;
          break;
        }

      if (j)
        continue;

      if (!isdigit(line[i]) || atoi(&(line[i])) < 0
              || atoi(&(line[i])) > 1)
      {
        printf("Error in configuration file: manualstep value must be 0 or 1.\n");
        printf("Default value: %s.\n\n",
                parms.manualstep ? "Enabled" : "Disabled");
      }
      else
        parms.manualstep = atoi(&(line[i])) ? true : false;
    }
  }                          //while

  fclose(confi);

  if (parms.transpfreq0 > parms.transpfreq)
  {
    parms.transpfreq0 = 250;
    parms.transpfreq = 600;

    printf("\ntranspfreq0 > transpfreq. Applying default settings:\n");
    printf("transpfreq0 = %d\n", parms.transpfreq0);
    printf("transpfreq = %d\n\n", parms.transpfreq);
  }
}

//get keypress
unsigned char getkey(bool verbose)
{
```

```
  unsigned char c;
  struct termios newtio;

  tty = open("/dev/tty", 0);
  tcgetattr(tty, &ttyoldtio);
  newtio = ttyoldtio;
  newtio.c_lflag &= ~ICANON; //clear ICANON flag
  if (!verbose)
    newtio.c_lflag &= ~ECHO; //clear ECHO flag
  tcsetattr(tty, TCSANOW, &newtio);

  //read command
  read(tty, &c, sizeof(char));

  tcsetattr(tty, TCSANOW, &ttyoldtio);
  close(tty);

  return c;
}

void *waitForEscape(void *s)
{
  controller *smeter = (controller *) s;

  //enable asynchronous cancelling
  pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, NULL);
  pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, NULL);

  //wait for escape (0x1b)
  while (getkey() != 0x1b);

  aborted = true;

  //stop spectrometer
  smeter->stop();
  //smeter->standbyM201();

  pthread_exit(NULL);
}

//continuously monitors Lawson's reading
void *monitorLawson(void *s)
{
  printf("\n");

  controller *smeter = (controller *) s;

  try
  {
    //wake up Lawson
    smeter->wakeM201();

    while (monitoring)
    {
      printf("%.1f nm\t%.6f mV      \r", smeter->getPosition(),
             smeter->getSingleConversion(false));
      fflush(stdout);
    }

    //put Lawson back to standby
    smeter->standbyM201();
  }
  catch(eM201Error)
  {
    fprintf(stderr, "\n%s: Communication error with Lawson.\n",
            __PRETTY_FUNCTION__);
    fprintf(stderr,
            "You may try doing something but don't be surprised\n");
```

```
    fprintf(stderr, "if it fails.\n");
  }
  catch(eControllerError)
  {
    fprintf(stderr, "\n%s: Communication error with controller.\n",
            __PRETTY_FUNCTION__);
    fprintf(stderr,
            "You may try doing something but don't be surprised\n");
    fprintf(stderr, "if it fails.\n");
  }
  catch(eSerialError)
  {
    fprintf(stderr, "\n%s: Serial communication error.\n",
            __PRETTY_FUNCTION__);
    fprintf(stderr, "Check cables, power, etc.\n");
    fprintf(stderr,
            "You may try doing something but don't be surprised\n");
    fprintf(stderr, "if it fails.\n");
  }

  pthread_exit(NULL);
}

void killMonitorThread(pthread_t * monitorthread)
{
  //kill thread waiting for keypress
  if (errno = pthread_cancel(*monitorthread))
  {
    perror("pthread_cancel");
    exit(-1);
  }

  //we must call pthread_join to avoid memory leaks
  if (errno = pthread_join(*monitorthread, NULL))
  {
    perror("pthread_join");
    exit(-1);
  }
}

// main program
int main(int argc, char **argv)
{
  char line[MAXLINE];
  double start, end, defaultstart, defaultend;
  int gain, count;
  unsigned char c;
  pthread_t monitorthread, m;

  // default values
  parms.showgraph = false;   //don't show graph after scan
  parms.PC_baud = 5;         //300bps
  parms.measfreq0 = 250;     // Hz
  parms.dstepsize = 0;
  parms.measfreq = 250;      // Hz
  parms.meassteps = 50;      // 50 steps = 10 nanometers resolution
  parms.M201_baud = 5;       // 300bps
  parms.wordcount = 2;       // 16 bit
  parms.gain = 0;            // 1
  parms.filter = 2;          // 400 Hz
  parms.channel = 0;         // channel
  parms.manualstep = false;  // disable manual stepping

  parms.port = PORTNAME;

  aborted = false;

  // read configuration file
```

```
readConf();

controller *smeter = new controller();

try
{
  //create monitoring thread
  if (errno = pthread_create(&monitorthread, NULL, waitForEscape, NULL))
  {
    perror("pthread_create");
    return -1;
  }

  try
  {
    smeter->signOn();
  }
  catch(eAborted)
  {
    printf("aborted\n\n");
    printf("Spectrometer initialization was not completed!!!\n\n");
  }

  if (!aborted)
  {
    //if not aborted then kill thread waiting for keypress
    killMonitorThread(&monitorthread);

    //reset tty
    tcsetattr(tty, TCSANOW, &ttyoldtio);
    close(tty);
  }
  else
    aborted = false;
}
catch(eM201Error)
{
  fprintf(stderr, "\n%s: Communication error with Lawson\n",
          __PRETTY_FUNCTION__);
  fprintf(stderr, "during spectrometer initialization. Exiting...\n");
  fprintf(stderr, "Check power, serial port settings, etc.\n");
  delete smeter;

  return -1;
}
catch(eControllerError)
{
  fprintf(stderr, "\n%s: Communication error with controller\n",
          __PRETTY_FUNCTION__);
  fprintf(stderr, "during spectrometer initialization. Exiting...\n");
  delete smeter;

  return -1;
}
catch(eSerialError)
{
  fprintf(stderr, "\n%s: Serial communication error\n",
          __PRETTY_FUNCTION__);
  fprintf(stderr, "during spectrometer initialization. Exiting...\n");
  fprintf(stderr, "Check cables, power, serial port settings, etc.\n");
  delete smeter;

  return -1;
}

printf("\nEnter command (h for help): ");
fflush(stdout);
```

```
while (1)
{
  try
  {
    // main menu loop
    while (1)
    {
      switch (getkey(true))
      {
        case 'h':
          printHelp();
          break;
        case 'm':
          start = end = count = 0;

          // find appropriate default values
          if ((smeter->getPosition() - MINWL) <= ((MAXWL - MINWL) / 2))
          {
            defaultstart = MINWL;
            defaultend = MAXWL;
          }
          else
          {
            defaultstart = MAXWL;
            defaultend = MINWL;
          }

          printf("\nEnter initial wavelength [%.0f]: ", defaultstart);
          fflush(stdout);

          do
          {
            c = getkey();
            if (c >= '0' && c <= '9')
            {
              start = 10 * start + c - '0';
              printf("%c", c);
              fflush(stdout);
              count++;
            }
            //0x7f - backspace
            else if (start && c == 0x7f)
            {
              start = (int) floor(start / 10.0);
              printf("\b \b");
              fflush(stdout);
              count--;
            }
            //0x1b - escape
          }
          while (c != '\n' && c != 0x1b);

          printf("\n");

          //0x1b - escape
          if (c == 0x1b)
            break;

          if (!count)
            start = defaultstart;

          count = 0;

          printf("Enter final wavelength [%.0f]: ", defaultend);
          fflush(stdout);

          do
          {
```

```
    c = getkey();
    if (c >= '0' && c <= '9')
    {
      end = 10 * end + c - '0';
      printf("%c", c);
      fflush(stdout);
      count++;
    }
    else if (end && c == 0x7f)
    {
      end = (int) floor(end / 10.0);
      printf("\b \b");
      fflush(stdout);
      count--;
    }
  }
  while (c != '\n' && c != 0x1b);

  printf("\n");

  //0x1b - escape
  if (c == 0x1b)
    break;

  if (!count)
    end = defaultend;

  //create monitoring thread
  if (errno =
          pthread_create(&monitorthread, NULL, waitForEscape,
                  NULL))
  {
    perror("pthread_create");
    return -1;
  }

  try
  {
    smeter->getConversion(start, end);
  }
  catch(eAborted)
  {
    printf("\b\b\b\baborted\n\n");
  }
  catch(...)
  {
    //if not aborted then kill thread waiting for keypress
    killMonitorThread(&monitorthread);

    //rethrow exception;
    throw;
  }

  if (!aborted)
  {
    //if not aborted then kill thread waiting for keypress
    killMonitorThread(&monitorthread);

    //reset tty
    tcsetattr(tty, TCSANOW, &ttyoldtio);
    close(tty);
  }
  else
    aborted = false;

  break;
case 'g':         //change gain
  gain = count = 0;
```

```
  printf("\nEnter log_2 gain [%d]: ", parms.gain);
  fflush(stdout);

  do
  {
    c = getkey();
    if (c >= '0' && c <= '9')
    {
      gain = 10 * gain + c - '0';
      printf("%c", c);
      fflush(stdout);
      count++;
    }
    else if (gain && c == 0x7f)
    {
      gain = (int) floor(gain / 10.0);
      printf("\b \b");
      fflush(stdout);
      count--;
    }
  }
  while (c != '\n' && c != 0x1b);

  printf("\n");

  //0x1b - escape
  if (c == 0x1b)
    break;

  if (!count)
    gain = parms.gain;

  if (gain < 0 || gain > 7)
    printf("Gain value must be in the range [0..7].\n");
  else
    smeter->setGain(gain);

  break;
case 'w':         //go to wavelength
  start = count = 0;

  printf("\nEnter wavelength [%d]: ", MINWL);
  fflush(stdout);

  do
  {
    c = getkey();
    if (c >= '0' && c <= '9')
    {
      start = 10 * start + c - '0';
      printf("%c", c);
      fflush(stdout);
      count++;
    }
    else if (start && c == 0x7f)
    {
      start = (int) floor(start / 10.0);
      printf("\b \b");
      fflush(stdout);
      count--;
    }
  }
  while (c != '\n' && c != 0x1b);

  printf("\n");

  //0x1b - escape
```

```
  if (c == 0x1b)
    break;

  if (!count)
    start = MINWL;

  //create monitoring thread
  if (errno =
          pthread_create(&monitorthread, NULL, waitForEscape,
                  NULL))
  {
    perror("pthread_create");
    return -1;
  }

  try
  {
    smeter->goTo(start);
  }
  catch(eAborted)
  {
    printf("aborted\n\n");
  }
  catch(...)
  {
    //if not aborted then kill thread waiting for keypress
    killMonitorThread(&monitorthread);

    //rethrow exception;
    throw;
  }

  if (!aborted)
  {
    //if not aborted then kill thread waiting for keypress
    killMonitorThread(&monitorthread);

    //reset tty
    tcsetattr(tty, TCSANOW, &ttyoldtio);
    close(tty);
  }
  else
    aborted = false;

  break;
case 'f':
  end = 0;

  printf("\nHow many nanometers");
  if (parms.manualstep)
    printf(" (0 - to the end)");
  printf("? [0]: ");
  fflush(stdout);

  do
  {
    c = getkey();
    if (c >= '0' && c <= '9')
    {
      end = 10 * end + c - '0';
      printf("%c", c);
      fflush(stdout);
    }
    else if (end && c == 0x7f)
    {
      end = (int) floor(end / 10.0);
      printf("\b \b");
      fflush(stdout);
```

```
    }
  }
  while (c != '\n' && c != 0x1b);

  printf("\n");

  //0x1b - escape
  if (c == 0x1b)
    break;

  //create monitoring thread
  if (errno =
          pthread_create(&monitorthread, NULL, waitForEscape,
                  NULL))
  {
    perror("pthread_create");
    return -1;
  }

  try
  {
    smeter->stepForward(end);
  }
  catch(eAborted)
  {
    printf("aborted\n\n");
  }
  catch(...)
  {
    //if not aborted then kill thread waiting for keypress
    killMonitorThread(&monitorthread);

    //rethrow exception;
    throw;
  }

  if (!aborted)
  {
    //if not aborted then kill thread waiting for keypress
    killMonitorThread(&monitorthread);

    //reset tty
    tcsetattr(tty, TCSANOW, &ttyoldtio);
    close(tty);
  }
  else
    aborted = false;

  break;
case 'b':
  end = 0;

  printf("\nHow many nanometers");
  if (parms.manualstep)
    printf(" (0 - to the end)");
  printf("? [0]: ");
  fflush(stdout);

  do
  {
    c = getkey();
    if (c >= '0' && c <= '9')
    {
      end = 10 * end + c - '0';
      printf("%c", c);
      fflush(stdout);
    }
    else if (end && c == 0x7f)
```

```
    {
      end = (int) floor(end / 10.0);
      printf("\b \b");
      fflush(stdout);
    }
  }
  while (c != '\n' && c != 0x1b);

  printf("\n");

  //0x1b - escape
  if (c == 0x1b)
    break;

  //create monitoring thread
  if (errno =
          pthread_create(&monitorthread, NULL, waitForEscape,
                  NULL))
  {
    perror("pthread_create");
    return -1;
  }

  try
  {
    smeter->stepBackward(end);
  }
  catch(eAborted)
  {
    printf("aborted\n\n");
  }
  catch(...)
  {
    //if not aborted then kill thread waiting for keypress
    killMonitorThread(&monitorthread);

    //rethrow exception;
    throw;
  }

  if (!aborted)
  {
    //if not aborted then kill thread waiting for keypress
    killMonitorThread(&monitorthread);

    //reset tty
    tcsetattr(tty, TCSANOW, &ttyoldtio);
    close(tty);
  }
  else
    aborted = false;

  break;
case 'p':
  smeter->park();
  delete smeter;

  return 0;
case 'q':
  delete smeter;
  printf("\n");
  return 0;
case 'r':
  printf("\n");
  delete smeter;

  readConf();
  sleep(1);
```

```
  smeter = new controller();
  //create monitoring thread
  if (errno =
          pthread_create(&monitorthread, NULL, waitForEscape,
                  NULL))
  {
    perror("pthread_create");
    return -1;
  }

  try
  {
    smeter->signOn();
  }
  catch(eAborted)
  {
    printf("aborted\n\n");
    printf("Spectrometer initialization was not completed!!!\n\n");
  }
  catch(...)
  {
    //if not aborted then kill thread waiting for keypress
    killMonitorThread(&monitorthread);

    //rethrow exception;
    throw;
  }

  if (!aborted)
  {
    //if not aborted then kill thread waiting for keypress
    killMonitorThread(&monitorthread);

    //reset tty
    tcsetattr(tty, TCSANOW, &ttyoldtio);
    close(tty);
  }
  else
    aborted = false;

  break;
case 'e':
  printf("\n\n");
  closePort();
  openPort(300, parms.port.c_str());
  c = serialRead();
  fprintf(stderr, "\n\nError code 0x%.2X\n", c);
  switch (c)
  {
    case ERR_U0BUF_FULL:
      fprintf(stderr, "USART0 (PC) transmit buffer overflow.\n");
      fprintf(stderr,
              "Increase PC baud rate, decrease stepping frequency, ");
      fprintf(stderr,
              "or increase number of steps between measurements.\n\n");
      break;
    case ERR_LAWSON_SLOW:
      fprintf(stderr,
              "Lawson is too slow for this scanning rate.\n");
      fprintf(stderr,
              "Increase Lawson's baud rate, increase number of steps \n");
      fprintf(stderr,
              "between measurements, or decrease word count.\n\n");
      break;
    default:
      fprintf(stderr, "Unknown error\n\n");
      break;
  }
```

```
        fprintf(stderr,
                "To continue work you must now reset controller \n");
        fprintf(stderr,
                "and reread configuration file (press r).\n\n");

        break;
      case 's':
        printf("\n%.1f nm\t%.6f mV\n", smeter->getPosition(),
                smeter->getSingleConversion());
        break;
      case 'c':
        monitoring = true;

        //create monitoring thread
        if (errno =
                pthread_create(&monitorthread, NULL, monitorLawson,
                        (void *) smeter))
        {
          perror("pthread_create");
          return -1;
        }

        //wait for keypress
        getkey();

        monitoring = false;

        //we must call pthread_join to avoid memory leaks
        if (errno = pthread_join(monitorthread, NULL))
        {
          perror("pthread_join");
          return -1;
        }
        break;
      default:
        printf("\n");
        break;
    }                   //switch

    printf("Enter command (h for help): ");
    fflush(stdout);
  }                     //while (1)
}                       //try
catch(eM201Error)
{
  fprintf(stderr, "\n%s: Communication error with Lawson.\n",
          __PRETTY_FUNCTION__);
  fprintf(stderr,
          "You may try doing something but don't be surprised\n");
  fprintf(stderr, "if it fails.\n");
  printf("\nEnter command (h for help): ");
  fflush(stdout);
}
catch(eControllerError)
{
  fprintf(stderr, "\n%s: Communication error with controller.\n",
          __PRETTY_FUNCTION__);
  fprintf(stderr,
          "You may try doing something but don't be surprised\n");
  fprintf(stderr, "if it fails.\n");
  printf("\nEnter command (h for help): ");
  fflush(stdout);
}
catch(eSerialError)
{
  fprintf(stderr, "\n%s: Serial communication error.\n",
          __PRETTY_FUNCTION__);
```

```
      fprintf(stderr, "Check cables, power, etc.\n");
      fprintf(stderr,
              "You may try doing something but don't be surprised\n");
      fprintf(stderr, "if it fails.\n");
      printf("\nEnter command (h for help): ");
      fflush(stdout);
    }
  }                          // while(1)

  //we should never reach this place
  return -1;
}
```

D.5 linuxserial.h

```
/* linuxserial.h                  (C) Joel Kuusk, 2003-2005 */

#ifndef __LINUXSERIAL_H
#define __LINUXSERIAL_H

#include <stdlib.h>
#include <string>            //string container, bzero()
#include <termios.h>
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>       // select()
#include <sys/time.h>        // select()
#include <sys/ioctl.h>

#define READTIMEOUT 5
#define WRITETIMEOUT 5

// Exceptions
class eSerialError
{
};
class eSerialWriteTimeout:public eSerialError
{
};
class eSerialReadTimeout:public eSerialError
{
};

void closePort(void);
void openPort(int, const char *);
int serialRead(unsigned char *, int, int = READTIMEOUT);
unsigned char serialRead(int = READTIMEOUT);
void emptyInputBuf(void);
void serialWrite(unsigned char *, int, int = WRITETIMEOUT);
void serialWrite(unsigned char, int = WRITETIMEOUT);
bool getRTS(void);
void setRTS(bool);

#endif        /* __LINUXSERIAL_H */
```

D.6 m201.h

```
/* m201.h                  (C) Joel Kuusk, 2003-2005 */

#ifndef  __M201_H
#define __M201_H

#include <math.h>
#include <iostream>
```

```
#include "linuxserial.h"

//controller's system clock frequency
#define F_OSC 14745600

#define ubrr_high(baud) (unsigned char)(((int)(F_OSC / (16 * (baud)) - 1) >> 8) & 0xFF)
#define ubrr_low(baud) (unsigned char)((int)(F_OSC / (16 * (baud)) - 1) & 0xFF)
#define high(c) (unsigned char)(((c) >> 8) & 0xFF)
#define low(c) (unsigned char)((c) & 0xFF)

#define min(a,b) ((a) < (b)) ? (a) : (b)
#define max(a,b) ((a) > (b)) ? (a) : (b)

#define ADCLOCK 19531.25     //10000000/512 for 10 MHz crystal
#define PLACEHOLDER 0x00
#define CONVERSION_FACTOR 0.0000002980232

// number of measurements to average = 2^average, average = 0..15
#define M201_AVERAGE 0

// sampling rate coefficient of ADC, percentage of measssteps
// e.g. 0.2 means that one measurement lasts for 20% of sampling interval.
#define M201_SAMPLINGCOEF 0.2

// false - bipolar, true - unipolar
#define M201_UNIPOLAR true

// AD read request processing time in seconds (excl. data transfer over RS232)
// As Lawsons processor creates the signals for RS232 communication with
// controller itself, it does not have to wait for full bit time before doing
// something else so the delay is formed of some constant delay
// plus some delay that depends on baud rate. In main program we
// use whole bytes (10 bits) and here we subtract appropriate part of
// last bit.
#define M201_ADSTARTDELAY 180e-6 + 244e-6 - (0.754 + 0.03) / BaudRate[parms.M201_baud]


// Exceptions
class eM201Error
{
};

struct parameters
{
//PC
  string port;
  bool showgraph;
//controller
  int PC_baud;
  int measfreq0;
  int measfreq;
  int transpfreq0;
  int transpfreq;
  int dstepsize;
  int measssteps;
  bool manualstep;
//M201
  unsigned char M201_baud;
  int wordcount;
  int samplingrate;
  unsigned char gain;
  unsigned char filter;
  unsigned char channel;
};

class M201
{
public:
```

```
  M201(void);
  ~M201(void);
  void buildModeWords(void);
  void signOn(void);
  void setBaudRate(unsigned char);
  void echoTest(void);
  void initMode(void);
  void sysCal(bool = true);
  void setMode(void);
  void setChannel(unsigned char, bool = true);
  void zeroDigitalOutput(void);
  void standby(void);
  void wake(void);
  int getVersion(void);

private:
  void sendPacket(unsigned char param1, unsigned char param2);

  unsigned char modereghi, moderegmid, modereglo;
};

extern struct parameters parms;
extern int BaudRate[];

#endif        /* __M201_H */
```

D.7 controller.h

```
/* controller.h                  (C) Joel Kuusk, 2003-2005 */

#ifndef  __CONTROLLER_H
#define __CONTROLLER_H

#include <sstream>           //ostringstream, graph title
#include <time.h>
#include <math.h>            //rint()
#include <list.h>            //list container
#include "m201.h"

#ifndef PI
#define PI 3.141592654
#endif

// grating parameter (1/m)
#define GRATING (1e-3 / 600)

// scale shift (nm)
#define SHIFT -75.5

// difference form Littrow mount in radians
#define EPSILON (2.4907 / 180 * PI)

// length of lever a (m)
#define LEVERA 0.04167

// error of length of lever a (m)
#define DELTAA -0.0003

#define stepfreq_high(freq) (unsigned char)(((int)(F_OSC / (64.0 * freq)) >> 8) & 0xFF)
#define stepfreq_low(freq) (unsigned char)((int)(F_OSC / (64.0 * freq)) & 0xFF)
#define MINWL 800            // wavelength when pos == MINWLPOS
#define MAXWL 2535           // max wavelength
#define STEPWL 0.2           // delta wavelength with one step
#define MINWLPOS 50          // controller's step count when wavelength == MINWL
#define TIMEOUTBUF 3         // buffer for calculated timeouts in seconds

//signal propagation delay in seconds in analog circuit
```

```
#define ADELAY 0.0058

#define GRAPH_WIDTH 640
#define GRAPH_HEIGHT 480

// Command tokens
#define CMD_U0ECHOTEST 0x00
#define CMD_START 0x01
#define CMD_STOP 0x02
#define CMD_FORWARD 0x03
#define CMD_BACKWARD 0x04
#define CMD_GOTO 0x05
#define CMD_U0BAUDRATE 0x06
#define CMD_U1BAUDRATE 0x07
#define CMD_CHANGESTEP 0x08
#define CMD_SCAN 0x09
#define CMD_INIT 0x0A
#define CMD_SETWORDCOUNT 0x0B
#define CMD_GETPOS 0x0C
#define CMD_M201SPEED 0x0D
#define CMD_RESET 0xA0

// Error codes
#define ERR_U0BUF_FULL 0x10  //USART0 buffer overflow
#define ERR_LAWSON_SLOW 0x20 //Lawson is too slow for this scanning rate

// Exceptions
class eControllerError
{
};
class eAborted
{
};

double stepsToWl(int);
int wlToSteps(double);

class controller
{
public:
  controller(void);
  ~controller(void);
  void signOn(void);
  void setM201BaudRate(void);
  void initPosition(void);
  void stepForward(double, bool = true);
  void stepBackward(double, bool = true);
  void goTo(double, bool = true);
  void stop(void);
  void getConversion(double, double);
  double getSingleConversion(bool = true);
  void setGain(unsigned char);
  void wakeM201(void);
  void standbyM201(void);
  void park(void);
  double getPosition(void);

private:
  void showGraph(list < double >*, double, double, double);
  void privateStepForward(double, bool = true);
  void privateStepBackward(double, bool = true);

  M201 *lawson;
  unsigned char reading[3];
};

extern bool aborted;
```

```
#endif        /* __CONTROLLER_H */
```

D.8 spectrometer.h

```
/* spectrometer.h                  (C) Joel Kuusk, 2003-2005 */

#ifndef  __SPECTROMETER_H
#define __SPECTROMETER_H

#include <pthread.h>
#include "controller.h"

#define CONFIFILE "spectrometer.conf"
#define PORTNAME "/dev/ttyS1"
#define MAXLINE 256

unsigned char getkey(bool = false);

#endif        /* __SPECTROMETER_H */
```

Lisa E PUBLIKATSIOON

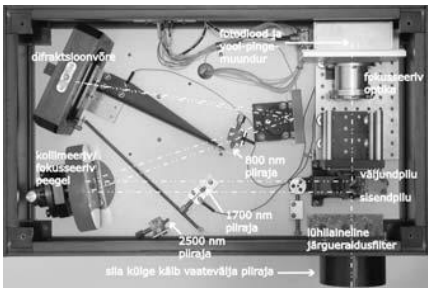
Joel Kuusk, Matti Pehk *”Välispektromeetri juhtimine ja andmehõive”* (stendiettekanne XXXV Eesti füüsikapäevadel 22.-23. märtsil 2005 Tartus).

Välispektromeetri juhtimine ja andmehõive

Joel Kuusk (Tartu Ülikool), Matti Pehk (Tartu Observatoorium)

Taimkattelt peegeldunud optiline (nähtav ja lähis- ning keskmine infrapuna, 400-2500 nm) kiirgus on peamine taimkatte seisundi vahendaja optilises kaugseires. Seni on peamiselt tehnilistel põhjustel vähe mõõtmistulemusi Päikese spektri lähis- ja kesk-infrapunases osas (800-2500 nm), ehkki just selles spektripiirkonnas kujundavad mitmed biokeemilised komponendid (vesi, proteiin, ligniin, tselluloos, pooltselluloos, suhkrud, tärklis) lehtede ja okaste optilised omadused. Siit tuleneb vajadus laiendada nii laboratoorsete kui välimõõtmiste spektripiirkonda sellesse spektrialasse, mis võib teha võimalikuks kiirguslevi mudelite jaoks vajalike komponentide optilise määramise senise väga töömahuka keemilise analüüsi asemel.

Käesolevas ettekandes antakse ülevaade Tartu Observatooriumi taimkatte kaugseire töörühmas valmistatavast 800-2500 nm piirkonnas töötavast spektromeetrist. Spektromeeter on ette nähtud taimkattelt tagasi peegelduva päikesekiirguse spektraalse jaotuse registreerimiseks maapealsete vahenditega kättesaadaval kõrgusel. Sisendoptika muutmisega on võimalik teda kohandada ka taimede üksikute osade mõõtmisteks sobivaks.

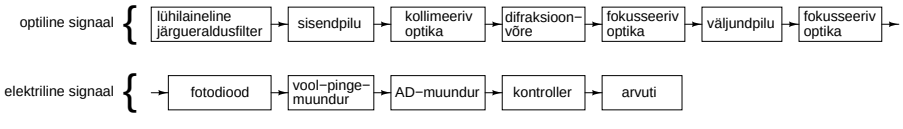


Joonis 1: Elementide paigutus ja kiirte käik spektromeetris

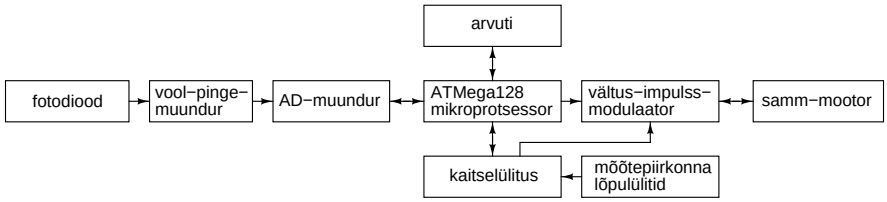
Spektromeetri põhilised tehnilised parameetrid on järgmised.

- Optilise osa moodustab klassikaline autokollimatsioonisüsteem valgusjõuga 1:5,7; difraktsioonvõrele langeva ja sealt peegelduva kiire vaheline nurk on 2°.
- Dispergeeriva elemendina kasutatakse difraktsioonvõret 600 joont/mm, λ_{max}=1300 nm.
- Kiirgusvastuvõtjateks on firma Hamamatsu kaks InGaAs fotodioodi.
- Spektri skaneerimine on realiseeritud siinusmehhanismiga sammu väärtusega 0,2 nm.
- Vaateväli (10°) on tagatud 5-astmelise väljadiafragmaga.
- Spektraalne lahutusvõime on 10 nm.
- Kogu spektri minimaalne skaneerimise aeg on 6 s.
- Spektromeetris kasutatakse analoog-digitaalmuundurit (AD-muundurit) M201 firmalt Lawson Labs, kontroller on tehtud Atmeli mikroprotsessori ATMega128 baasil.
- Arvutiga on spektromeetri kontroller ühendatud jadaliidese abil.
- Välimõõtmiste ajal on spektromeetri kontroller, sülearvuti ning autonoomne toide paigutatud eraldi kohvrisse.

Praeguseks on spektromeetri ehitus jõudnud sellisesse staadiumi, et temaga on võimalik alustada proovimõõtmisi. Kiirte käik ja komponentide paigutus spektromeetris on kujutatud joonisel 1, mõõdetava signaali käiku läbi spektromeetri illustreerib plokkskeem joonisel 2. Esimesed proovimõõtmised on läbi viidud ühe vastuvõtjaga lainealas 800-1700 nm. Hetkel puuduvad veel pikalaineline järgueraldusfilter ja



Joonis 2: Mõõdetava signaali liikumise plokkskeem

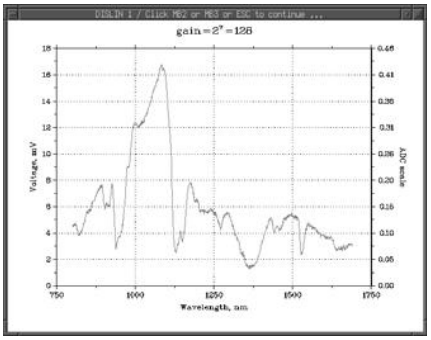


Joonis 3: Spektromeetri ja kontrolleri plokkskeem

Välimõõtmistel tuleb skaneeriva spektromeetriga muutuvate valgustustingimuste tõttu sooritada mõõtmine võimalikult kiiresti. Seetõttu on nii spektromeetri riistvara kui tarkvara välja töötatud, pidades silmas maksimaalset kiirust ja minimaalset energiakulu. Näiteks samm-mootori juhtimiseks kasutatakse energiasäästlikku vältus-impulssmodulatsiooni (pulse-width modulation). Spektromeetri ja tema kontrolleri plokkskeem on kujutatud joonisel 3.

Spektromeetri juhtprogramm on kirjutatud keeles C++, mõõtmistulemuste visualiseerimiseks on kasutusel joonestusteek Dislin. Töötada on võimalik nii graafilises kui tekstirežiimis. Programm võimaldab mõõta kogu spektri või ainult osa sellest, sammuda soovitud lainepikkusele ja võtta ühekordselt või korduvalt lugemeid ühel lainepikkusel. Spektri mõõtmisel on graafilises keskkonnas töötamise korral võimalik näidata tulemust ekraanil graafikuna, lisaks salvestatakse see tabuleeritud kujul kõvakettale. Programmi konfiguratsioonifailis on võimalik muuta diskreetimisintervalli ja skaneerimise kiirust. Sammude vahelejätmise vältimiseks toimub sammumise alustamine ja peatumine sujuvalt ning võimalik on eraldi määrata nii algkiirust kui kiirendust mõõtmise ja ühelt lainepikkuselt teisele sammumise jaoks. Ka AD-muunduri tööparameetreid (võimendus, bittide arv) saab konfiguratsioonifailis seada. Spektromeetri juhtprogrammi ekraanitõmmised on kujutatud joonistel 4 ja 5.

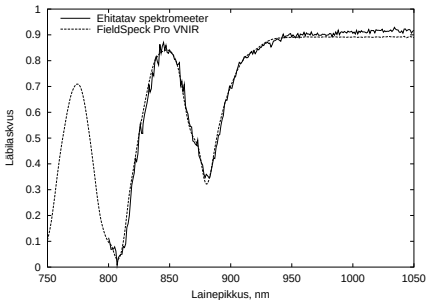
Joonis 4: Spektromeetri juhtprogrammi ekraanitõmmis



Kontrolleri programm on kirjutatud assemblerkeeles. Kogu programmi töö on üles ehitatud riistvaraliste ja tarkvaraliste katkestuste töötlemisele.

Kontrolleril on kaks jadaväratit AD-muunduriga ja arvutiga suhtlemiseks. Kogu AD-muunduri andmevoog läbib kontrollerit. Initsialiseerimisel ja üksikute lugemite võtmisel toimib kontroller AD-muunduri ja arvuti vahel läbipaistva lüüsina, spektri mõõtmise ajal kontrollib seadme tööd täielikult kontrolleri programm. Maksimaalse kiiruse saavutamiseks toimub lugemite võtmine sammumisega samaaegselt. Kontroller suhtleb AD-muunduriga kiirusel 300-9600 boodi sekundis. Enne lugemi võtmist saadab kontroller üle jadaliidese kolm baiti AD-muundurile ja saab ühe baidi tagasi. Seetõttu on vajalik saata AD-muundurile käske sõltuvalt skaneerimise ja jadavärati kiirusest mõned sammud enne vastavale lainepikkusele jõudmist.

Mikrokontrolleri programmis on ka mõned lihtsamad veaolukordade kontrollid (andmepuhvri ületäitumise kontroll, AD-muunduriga andmeside kiiruse kontroll). Vea ilmnedes antakse sellest märku valgusdioodi vilgutamisega ja spektromeeter viiakse avariirežiimi - arvutiga suhtlemise kiirus seatakse 300 boodi peale, arvutile saadetakse pidevalt veakoodi ja oodatakse lähtestamiskäsku.



Joonis 6: Klaasfiltri läbipaistvus mõõdetuna kahe erineva spektromeetriga

Spektromeetriga on läbi viidud esimesed proovimõõtmised laboritingimustes. Testobjektiks on valitud klaasfilter, millel vahemikus 800-1050 nm on keeruka kujuga läbilaskvuskõver. Joonisel 6 on mõõdetud klaasfiltri läbilaskvus ehitatava spektromeetriga ja võrdluseks sama filtri läbilaskvus mõõdetuna spektromeetriga FiledSpec Pro VNIR firmalt Analytical Spectral Devices, Inc. Lainepikkuste vahemikuks on valitud kahe spektromeetri mõõtepiirkondade ühisosa. Ehitatava spektromeetriga mõõdetud läbilaskvuskõvera sakilisus on põhjustatud difraktsioonvõre vibreerimisest mõõtmise ajal. Selle mõju peaks oluliselt aitama vähendada võre pöörava mehhanismi