TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatika instituut

IDK40LT

Dmitry Fisun 123498IABB

SATELLIIDI TOORANDMETE SIMULATSIOONI TARKVARA ANALÜÜS, PROJEKTEERIMINE JA PROTOTÜÜPIMINE

Bakalaureusetöö

Juhendaja: Inna Švartsman
MSc
Lektor

Kaasjuhendaja: Sander Rikka
MSc
Nooremteadur

Tallinn 2017

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Dmitry Fisun

03.01.2017

Annotatsioon

Antud bakalaureusetöö eesmärk on pakkuda satelliidi toorandmete simulatsiooni lahendust TTÜ Mektory kuupsatelliidi pilditöötluse algoritmi testimiseks. Projekti raames kasutatakse satelliidi liikumise füüsilist modelleerimist monteeringu ja *line scan* kaamera abil. Töö ülesandeks oli simulatsiooni tarkvara analüüs, projekteerimine ja prototüüpimine.

Töö tulemusena on kogutud ja analüüsitud süsteemi nõuded, nende alusel on valitud töövahendid, projekteeritud arhitektuur, realiseeritud monteeringu juhtimise funktsionaalsus ning on alustatud kaamera juhtimise mooduli arendamine.

Tulemuste alusel on pakutud projekti tähtaegade hinnangud ning soovid projekti edasiseks arendamiseks.

Prototüüp on realiseeritud programmeerimiskeeltes C ja Python, kasutatud töövahendid on PIXCI *frame grabberi* programmeerimisteek EPIX XCLIB Lite ning Python seriaalpordi juurdepääsu teek PySerial.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 22 leheküljel, 4 peatükki, 5 joonist, 2 tabelit.

Abstract

Analysis, Design and Prototyping of Satellite Raw Data Simulation Software

The aim of this thesis is to provide a satellite raw data simulation solution for testing the TUT Mektory CubeSat image composition algorithm. CubeSat raw data is simulated using the line scan camera and a computerized mount. Goals of this thesis are analysis, design and prototyping of the simulation software.

As a result, software requirements have been gathered and analyzed, programming tools analyzed and chosen, system architecture developed. The prototype is implemented in Python. The mount controller is fully functional and was implemented using the PySerial serial port access library. The camera controller is under development and relies on the EPIX XCLIB Lite frame grabber programming library for the PIXCI frame grabbers.

Considering results and time spent on the project, deadline estimations were provided along with suggestions for continuing the project.

This paper can be used by CubeSat developers who are interested in applying line scan cameras for Earth observation.

The thesis is in Estonian and contains 22 pages of text, 4 chapters, 5 figures, 2 tables.

Lühendite ja mõistete sõnastik

Line scan kaamera	Kaamera, mille sensor on valgustundlike pikslite riba (laiusega üks piksel)
Area scan kaamera	Kaamera, mille sensor on valgustundlike pikslite maatriks.
Monteering	Teleskoobi alus, mis võimaldab hoida vaateväljas Maa pöörlemise tõttu liikuvate taevaobjekte.
Frame grabber	Seade, mis salvestab üksikuid kaadreid analoog- või digitaal videojadast
Pushbroom skanner	Satelliitkaugseires kasutatav seade, mis seisneb ühel liinil paigaldatud spektroskoopilistest sensoritest. Sensorite liin on asendatud perpendikulaarselt satelliidi liikumisele.
Whiskbroom skanner	Satelliitkaugseires kasutatav seade, mis seisneb liikuvate peeglite süsteemist ja sensoritest. Peeglite süsteem skaneerib maapinna ja suunab projektsiooni sensori sisse.
Camera Link	Seriaal ühendusega arvutiliides, mis on mõeldud masinnägemise seadmetega ühendamiseks
PCI	*Peripheral Component Interconnect,* personaalarvuti siini standard
DLL	*Dynamic Link Library*
telemeetria	Satelliidi kaader koos metaandmetega (nagu positsioon pildistamisel)
ADCS	*Attitude Determination and Control System*
pööre	Suhteline pöörlemine ümber x-telje
kallutus	Suhteline pöörlemine ümber y-telje
lengerdus	Suhteline pöörlemine ümber z-telje
WP3	*Work Package 3 Team Data Systems.* Satelliidiprogrammi Andmetöötluse meeskond
WP6	*Work Package 6 Team On-Board Computer.* Satelliidiprogrammi Pardaarvuti meeskond

Sisukord

Jooniste loetelu

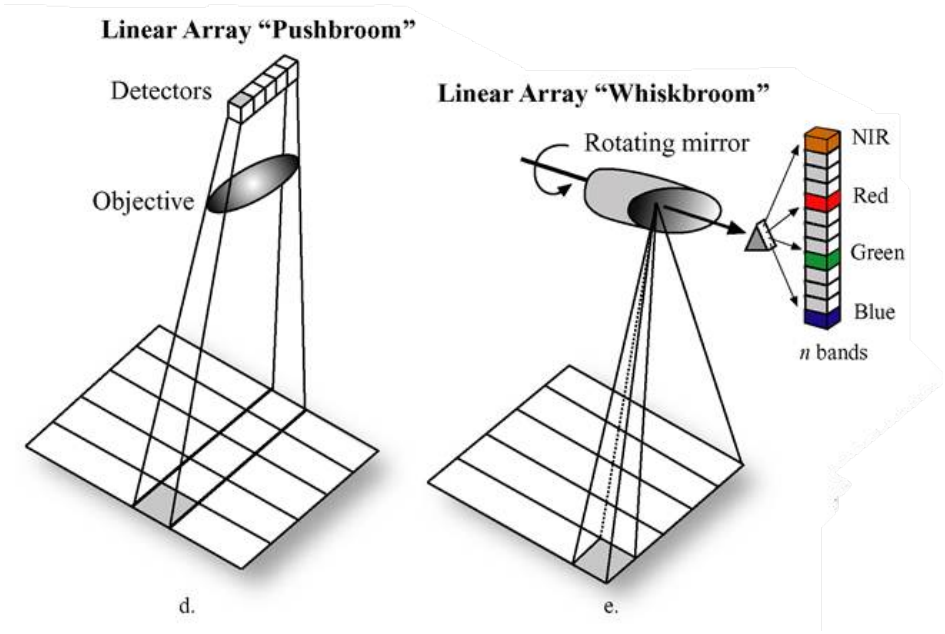
Tabelite loetelu

1 Sissejuhatus

Antud bakalaureusetöö eesmärk on pakkuda satelliitandmete simuleerimise lahendust TTÜ Mektory kuupsatelliidi pilditöötluse algoritmi testimiseks. TTÜ Mektory Satelliidiprogrammi raames plaanitakse esmakordselt kuupsatelliitide ajaloos kasutusele võtta *line scan* kaamerat, mis muudab selle suurte optiliste satelliitide lähedaseks. Satelliidi suurus ja kaal ei võimalda kasutada tööstussatelliitidel kasutatavaid stabiliseerimistehnoloogiaid, seega alternatiivlahendusena on valitud andmete tarkvaraline analüüs ja töötlemine. Saadud testandmed võimaldavad kontrollida pilditöötluse algoritmi korrektsust ning hinnata *line scan* kaamera kasutamise perspektiivsust ja tootlust.

1.1 Taust ja probleem

Kaugseire valdkonnas kasutatakse laialdaselt *pushbroom* skannerid tänu nende madalale kaalule ja hinnale (võrreldes eelnevalt levinud *whiskbroom* skanneritega). *Whiskbroom* skanneri korral toimub pildiandmete saamine vähese sensorite arvuga, kui liikuvate peeglite süsteem skaneerib maapinna ja suunab projektsiooni sensori sisse. *Pushbroom* skanner on *whiskbroom* tehnoloogia edasiareng. Suurest sensorite arvust moodustatakse massiivi, mis skaneerib maapinna perpendikulaarselt satelliidi liikumisele [1].



Joonis 1. *Pushbroom* ja *whiskbroom* skannerite skemaatiline kuju [2].

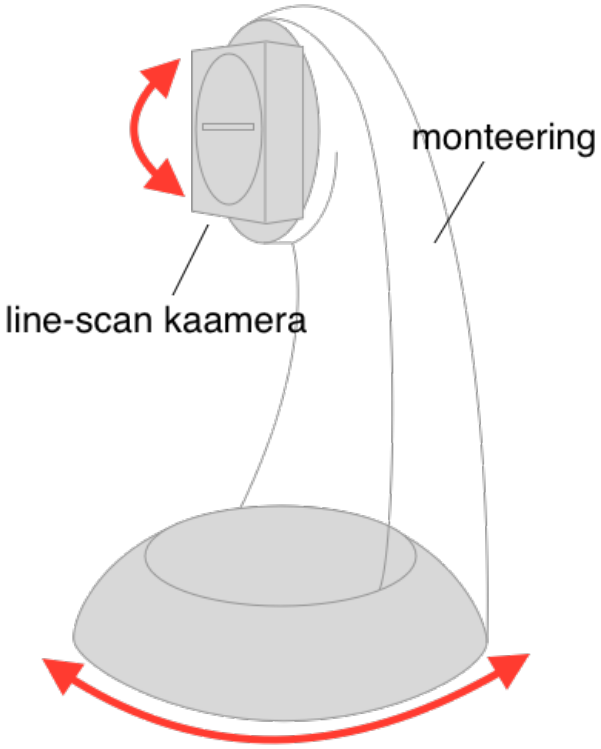
Line scan kaamera põhineb samal printsiibil, mis *pushbroom* skanner, kuid esimene on nõrgem spektraalse informatsiooni kogumise osas. Eritingimus seisneb samuti selles, et TTÜ Mektory kuupsatelliidi korral on võimatu üheaegselt liikumist kontrollida ja positsiooni määrata, kuna satelliidi pöörlemisel tekkiv magnetväli mõjutab asendi saamiseks kasutatavaid magnetomeetreid. Pildistamise režiimis liigub satelliit juhtimiseta, salvestades iga kaadri koos selle positsiooniga (telemeetriat). Telemeetria töötlemine ja tervikpildi konstrueerimine toimub satelliidi pardaarvutis ning lõpptulemus saadetakse TTÜ Mektory Kosmosekeskuse maajaamale.

Pilditöötluse algoritmi valideerimise ja korrigeerimise jaoks on vajalik testandmete kogum, mis oleks lähedane orbiidil saadavatele andmetele. Projekti raames otsustati kasutada satelliidi liikumise füüsilist modelleerimist, et uurimise jooksul ehitatud testpinki saaks edaspidi kasutada satelliidi komponentide testimiseks.

1.2 Ülesande püstitus

Valitud lahendus seisneb satelliidi liikumise simuleerimises monteeringu ja *line scan* kaamera abil. *Line scan* kaamera on arvutiga ühendatud läbi *frame grabberi* ning on paigaldatud monteeringule, mis on juhitav arvuti abil. Monteering võimaldab pöörata kaamerat kahes suunas. Eksperimendi jooksul kaamerat suunatakse testmustrile ning muudetakse kaamera asendit. Kaamera liikumisel salvestatakse pildiandmed koos

suhtelise positsiooniga. Saadud andmekogum võimaldab testida satelliidi pilditöötluse moodulit.



Joonis 2. Monteeringu koos sellele paigaldatud *line scan* kaameraga skemaatiline kuju. Nooltega on näidatud pöörlemissuunad.

Töö ülesanne on analüüsida ja projekteerida satelliidi toorandmete simuleerimise rakendust ning teha selle prototüüpi.

1.3 Metoodika

Valitud metoodika seisneb järgmistest sammudest:

1. nõuete kogumine ja analüüs huvitatud osapooltega suhtlemise ja olemasoleva infrastruktuuri uurimise kaudu,
2. kehtestatud nõuete alusel arendusvahendite analüüs ja valik,
3. simulatsiooni tarkvara projekteerimine ja prototüüpimine,
4. tulemuste analüüs,
5. tähtaegade hinnang.

1.4 Ülevaade tööst

Töö esimeses osas tutvustatakse TTÜ Mektory Satelliidiprogrammi ja satelliitide pildistamistehnoloogiaid, selgitakse töö tausta ja probleemi, kirjeldatakse ülesande püstitust ja metoodikat.

Teises osas vaadeldakse analüüsi käigus kehtestatud nõuded, nende alusel valitud töövahendid ning arendusmetoodika.

Kolmandas osas kirjeldatakse simulatsiooni tarkvara arhitektuuri ja prototüüpe.

Neljandas osas analüüsitakse töö tulemusi, antakse tähtaegade hinnanguid ning pakutakse võimalusi ja soovitusi projekti edasiseks arendamiseks.

2 Analüüs

Järgnevalt esitatakse simulatsiooni seadme ülevaadet, nõuete kogumise ja analüüsi käigus tehtud otsuseid ja nende põhjendusi.

2.1 Simulatsiooni seadme ülevaade

Simulatsiooni seade koosneb Celestron monteeringust, mis on arvutiga ühendatud läbi Celestron NexStar+ juhtarvuti. Suhtlemine juhtarvutiga toimub RS-232 pordi kaudu. Monteeringule on paigaldatud Basler L103K-1k *line scan* kaamera, mis on ühendatud PIXCI CL1 *frame grabberiga* Camera Link liidese kaudu.

Joonis 3. Simulatsiooni seadme prototüüp [3].

2.2 Nõuete kogumine ja analüüs

Analüüsi käigus uuriti nii Satelliidiprogrammi meeskondade nõudeid ja ettepanekuid, kui organisatsioonis olevat infrastruktuuri ja varustust.

2.2.1 Funktsionaalsed nõuded

Analüüsi käigus sai püstitatud järgmised funktsionaalsed nõuded:

- Kasutaja saab määrata kasutatud objektiivi ning simuleeritava orbiidi kõrgust.
- Kasutaja saab määrata monteeringu liikumistrajektoori.
- Kasutaja saab testimistarkvara kasutada nii graafilise kui käsurea kasutajaliidesega.
- Kasutaja saab katsetust peatada.

2.2.2 Mittefunktsionaalsed nõuded

Analüüsi käigus sai püstitatud järgmised mittefunktsionaalsed nõuded:

- Testimistarkvara peab automaatselt teisendama positsiooni andmeid vastavalt simuleeritavale orbiidile.
- Testimistarkvara komponendid peavad olema kergelt C keelde teisaldatavad.
- Iga kaader peab olema seostatud ühe positsiooniga.
- Kaadri maksimaalne pikkus on 16 352 piksli.
- Positsioon koosneb kuuest elemendist – laiuskraad, pikkuskraad, kõrgus, pööre (suhteline pöörlemine ümber x-telje), kallutus (suhteline pöörlemine ümber y-telje) ja lengerdas (suhteline pöörlemine ümber z-telje).
- Positsiooni andmete täpsus on 3 komakohti kraadides ning 5 komakohti radiaanides.
- Positsiooni arvnäitajad ümardatakse allapoole.
- Katsetuse maksimaalne kestus on 5 minutit.
- Maksimaalne ooteaeg andmebaasiga ühendamisel on 5 sekundit.
- Maksimaalne ooteaeg monteeringuga ühendamisel on 10 sekundit.
- Maksimaalne ooteaeg kaameraga ühendamisel on 10 sekundit.
- Vea korral lõpetatakse tööd ning näidatakse veateadet.

2.2.3 Varustus

Projekti realiseerimiseks oli pakutud järgmised seadmed ja tarkvara:

- Celestron monteering kahe pöörlemismootoriga,
- Celestron NexStar+ juhtarvuti seriaal (RS-232) liidesega,
- Basler L103K-1k *line-scan* kaamera Camera Link liidesega,
- EPIX PIXCI CL1 Camera Link *frame grabber* PCI liidesega,
- Fujitsu Siemens Celsius M450 tööjääm (Core 2 Duo E6600, PCI, RS-232, Ubuntu 14.04.4 LTS 64bit),
- HP Compaq dc5800 Microtower tööjääm (Intel Pentium E2220, PCI, RS-232, Microsoft Windows 7 Ultimate SP1 64bit).

2.3 Töövahendite valik

Järgnevalt esitatakse töövahendite valiku protseduuri ja põhjendust.

2.3.1 Valikukriteeriumid

Tööriistade ja arhitektuuri valikul pöörati erilist tähelepanu järgmistele kriteeriumitele, mis lähtusid projekti akadeemilisest iseloomust ning infrastruktuurist:

- Multiplatvormilisus (Windows ja Linux),
- Levinumate programmeerimiskeelte kasutamine,
- Võimalikult madalamad kulud.

Lisatingimused tulid satelliidi tarkvaraarenduse meeskondadest:

- Komponentide võimalikult lihtne C keelde teisaldamine nende käivitamiseks pardaarvutis (WP6),
- Komponentide integreeritavus .NET platvormiga, kuna pildi komponeerimise tarkvara oli realiseeritud C# keeles (WP3).

2.3.2 Monteeringu juhtimine

Monteeringu juhtimine toimub seriaalpordi (RS-232) kaudu. Monteeringu ühendamine arvutiga on võimalik kas juhtarvuti läbi (sel juhul saab kasutada Celestroni ametliku juhtarvuti kommunikatsiooni protokolli) või otseselt (kuid sel juhul saab kasutada ainult mitteametlikku pöördprojekteerimise teel saadud monteeringu sisekäsude protokolli

kirjeldust) [3]. Kuna olemasolev NexStar+ juhtarvuti toetas ametliku kommunikatsiooni protokolli ja pakkus kõike lõputöö probleemi lahendamise jaoks vajalikke operatsioone, valiti selle kasutatavate tööriistade hulka.

Monteeringu juhtimisel on vaja arvestada, et maksimaalne ooteaeg positsiooni või liikumiskäskude vastamisel on 3.5 sekundit [4].

Monteeringu kontrolleri arendamist võimaldavad järgmised teegid:

Nimetus	**RXTX**	**PySerial**	**Serialio Serial Port Standard Windows**	**Termios**
Keel	Java	Python	Java	C
Multiplatvormiline	+	+	– (Windows)	– (Unix)
Tasuta	+	+	– ($49.95)	+
Integreeritav C#-ga	–	+ (IronPythoni kaudu)	–	+

Tabel 1. Seriaalpordi juurdepääsu teegide võrdlus.

Kõikidele valikukriteeriumitele vastav teek on PySerial, mille järgi võeti see kasutusele monteeringu kontrolleri arendamisel.

2.3.3 Kaamera juhtimine

Seadmete spetsifikatsioonide uurimisel selgus, et kaadrite salvestamiseks EPIX PIXCI CL1 *frame grabberi* abil on vajalik EPIXi ametlik C keeles realiseeritud programmeerimisteek XCLIB, mis on saadav mitmel versioonil.

Nimetus	**Toetatud platvorm**	**Hind**
XCLIB for Windows	Windows 10, 8, 7, Vista, XP (32 and 64 bit), Windows 2000, NT (32 bit)	$495.00
XCLIB Lite for Windows	Windows 10, 8, 7, Vista, XP (32 and 64 bit), Windows 2000, NT (32 bit)	$195.00
XCLIB for Linux	Linux on Intel i386 (32 bit), Intel x86-64 (64 bit)	$495.00

Tabel 2. XCLIB teegi versioonide võrdlus [5].

XCLIB Lite on programmeerimisteegi piiratud ja lihtsustatud versioon. Kuna see pakkus piisavat funktsionaalsust käesoleva lõputöö probleemi lahendamiseks ning

otsustamise ajal ei õnnestunud leida odavamaid alternatiive, valiti XCLIB Lite for Windows kaamera juhtimise töövahendina. Sellest lähtuvalt sai sihtplatvormiks Microsoft Windows 7. Kuna XCLIB Lite kasutusel kirjutatud tarkvara saab käivitada kasutades teegi edasijõudnud versiooni, on võimalik lõpplahenduse kasutamine ka Linux platvormil.

2.3.4 Valitud varustus ja tarkvara

Eelnevalt langetud otsuste põhjal valiti järgmised seadmed ja tarkvara testpingi ehitamiseks:

- Celestron monteering kahe pöörlemismootoriga,
- Celestron NexStar+ juhtarvuti seriaal (RS-232) liidesega,
- Basler L103K-1k *line-scan* kaamera Camera Link liidesega,
- EPIX PIXCI CL1 Camera Link *frame grabber* PCI liidesega,
- HP Compaq dc5800 Microtower tööjääm (Intel Pentium E2220, PCI, RS-232),
- Microsoft Windows 7 Ultimate SP1 64bit,
- EPIX XCLIB Lite for Windows,
- PySerial,
- Python 3.5.

2.4 Arendusmetoodika valik

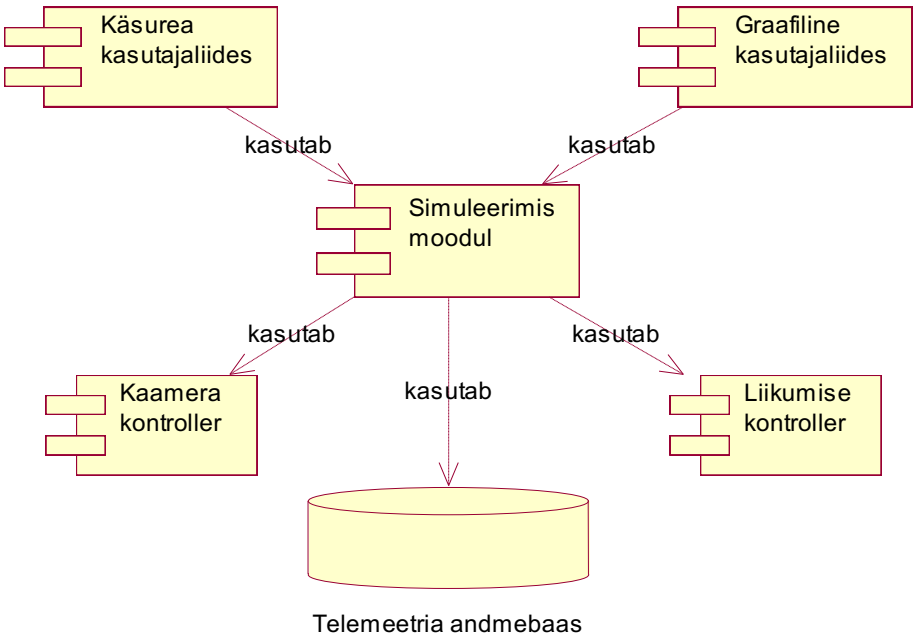
Kuna tarkvara analüüsi käigus oli tegemist suhteliselt ebakindlate nõudmistega; mõned tarkvaraarendajad ei olnud probleemi valdkonnaga tuttavad ning projektiga kaasnesid kõrged riskid, otsustati iteratiivset arendusmetoodikat kasutusele võtta [7]. Valitud arendusmetoodika on evolutsiooniline prototüüpimine.

3 Disain ja prototüüpimine

Järgnevalt esitatakse simulatsiooni tarkvara arhitektuuri ja komponentide prototüüpe kirjeldust.

3.1 Simulatsiooni tarkvara ülevaade

Analüüsi käigus kehtestatud nõuetest lähtudes valiti tarkvara arendusmetoodikaks komponent-orienteeritud tarkvaraarendust. Testimistarkvara funktsionaalsus on jagatud kuue komponendi vahel, millest tähtsaim on simuleerimismoodul.



Joonis 4. Simulatsiooni tarkvara arhitektuur.

Simuleerimismoodul saab kasutajaliidese kaudu sisendit katsetuse korraldamiseks, seejärel kontrollib, kas liikumise ja kaamera kontrollerid on valmisolekus ning vastavad katsetuse seadistusele, ning alustab katsetust. Katsetuse jooksul saadab simuleerimismoodul päringuid kontrolleritele ning koheselt seostab saadud andmeid (kaadrit kaamera kontrollerist ja positsiooni liikumise kontrollerist) ja salvestab neid andmebaasi.

Järgnevalt vaadeldakse detailselt iga testimistarkvara komponendi.

3.2 Kasutajaliides

Käsurea kasutajaliides ei ole interaktiivne. Kõik parameetrid sisestatakse programmi käivitamisel parameetritena. Programmi käivitamisel saab kasutaja määrata järgmisi parameetreid:

- viide tekstifaili liikumistrajektooriga (kohustuslik),
- liikumistelgede seadistus (laiuskraad, pikkuskraad, kõrgus, pööre, kallutus, lengerdus) sõltuvalt kaamera asendist testpingil,
- viide tekstifaili tulemuste salvestamiseks (mittekohustuslik, vaikimisi salvestatakse käivitamise kausta lisades nimele ajatempli).

Graafiline kasutajaliides pakub identset funktsionaalsust. Parameetrite muutmine katsetuse jooksul ei ole võimalik.

3.3 Simuleerimismoodul

Simuleerimismooduli ülesanne on sisendandmete alusel kaamera ja liikumise kontrolleri juhtimine ning kontrolleritest saadud andmete seostamine ja salvestamine andmebaasi.

Simuleerimismoodulil on neli vastutusala:

- sisendandmete valideerimine ja lugemine,
- trajektoori teisendamine monteeringu juhtimiskäskudeks,
- liikumise kontrolleri juhtimine,
- kaamera kontrolleri juhtimine,
- katsetuse tulemuste salvestamine.

3.4 Kaamera kontroller

Kaamera kontroller kasutab EPIX XCLIB Lite DLLi kaamera juhtimiseks. Kuna XCLIB on realiseeritud C keeles, integreerimine Pythoniga toimub *ctypes* kaudu, mis on Pythoni sisseehitatud teek väliste funktsioonide kasutamiseks.

3.5 Liikumise kontroller

Liikumise kontroller vastutab liikumise ja positsiooni määramise eest. Monteeringu liikumine on võimalik kahes režiimis [5]:

1. Kindlas suunas liikumise alustamine või peatamine *("Slewing"*). Saab määrata liikumise kiirust, peatamine toimub koheselt.
2. Liikumine kindlasse asendisse (*"GoTo"*). Monteering liigub maksimaalse kiirusega valitud asendisse ning seejärel aeglasemate pöörlemiste abil korrigeerib oma positsiooni.

Kuna viimane meetod ei rahuldanud determineeritud liikumise nõuet, valiti esimest liikumise režiimi. Sellest lähtuvalt pakuti liikumiskäsu formaadi – kasutaja saab määrata liikumiskäskude jada, kus iga käsk sisaldab ühe liikumiskomponendi liikumissuunda (positiivne või negatiivne), aega (sekundites) ja kiirust (kaaresekundites sekundis).

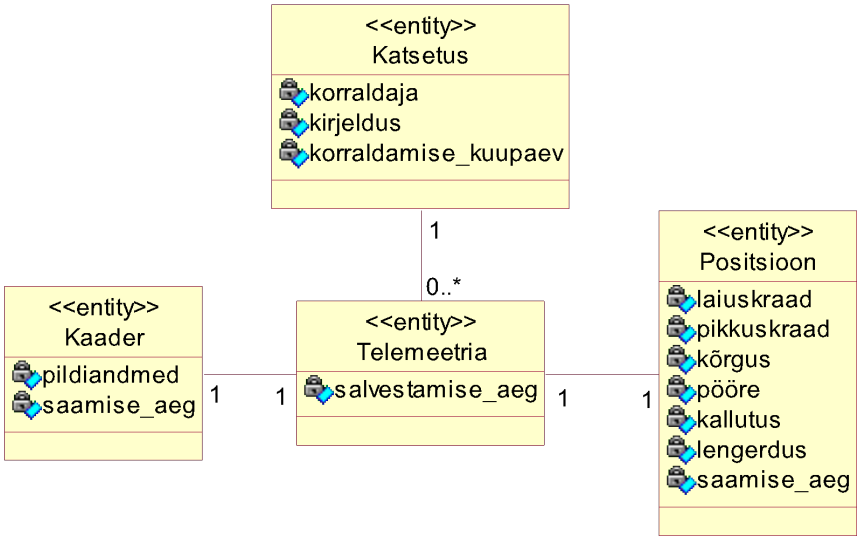
Kui liikumine mingil teljel on võimatu, siis see käsu ignoreeritakse. Kui ei saa nõutud kiirusega liikuda, tagastatakse veateadet.

Arvestades, et kaamera paigaldamine monteeringule on võimalik mitmel viisil, selgus lisanõue liikumise kontrolleri konfigureerimisele. Tarkvara käivitamisel peab olema võimalik testpingi andureid vastandada vajalikele liikumiskomponentidele.

3.6 Telemeetria andmebaas

Telemeetria andmebaasi projekteerimisel lähtuti tingimusest, et peaks olema võimalik korrigeerida valesti seostatud telemeetriat (ehk muuta seost salvestatud kaadrite ja positsioonide vahel).

Kontseptuaalse andmebaasi relatsiooni suhet kirjeldab järgmine lausend: Telemeetria on Katsetuse jooksul registreeritud Positsiooni ja Kaadri seos.



Joonis 5. Telemeetria andmebaasi kontseptuaalne diagramm.

Kuna kasutatud monteering võimaldas ainult kaheteljelist kaamera liikumist ning edaspidi sooviti kasutada enam funktsionaalseid lahendusi, andmebaasi projekteerimisel arvestati kuus liikumiskomponenti (laiuskraad, pikkuskraad, kõrgus, pööre, kallutus, lengerdus).

Et lihtsustada testimistarkvara parandamist ja silumist ning pakkuda rohkem informatsiooni pilditöötluse mooduli testimiseks, otsustati ajatempleid andmetele lisada koheselt tekkimise hetkel ehk kaamera ja liikumise kontrollerite sees. Sel juhul saab tuvastada simuleerimismooduli andmete vastandamisvigu ning tulemusi korrigeerida ilma katsetuse taaskäivitamiseta. Kaadri ja positsiooni seostamise tulemus (telemeetria) samuti saab ajatempli juurde.

4 Kokkuvõte

Käesoleva töö eesmärgid olid simulatsiooni tarkvara nõuete kogumine ja analüüs, töövahendite analüüs ja valik, arhitektuuri projekteerimine ning prototüüpimine. Enamus nendest on edukalt sooritatud: ajalimiidi tõttu sai valmis ainult osa prototüüpi funktsionaalsusest. Liikumise kontrolleri prototüüp on täiesti funktsionaalne, kaamera kontrolleri prototüüp on arendamise faasis, simulatsiooni mooduli prototüüp ja telemeetria andmebaas on realiseerimata.

Tulemuste alusel antakse järgmisi tähtaegade hinnanguid (2016. a. augusti seisuga):

- Simuleerimise tarkvara prototüüpi valmisolek – 2016. a. novembri lõpp,
- Simuleerimisandmete tootmise metoodika väljatöötamine ja tõestamine – 2017. a. juuni algus,
- Simuleerimisandmete tootmine – 2017. a. juuni lõpp.

Hinnangute alusel soovitatakse vaadelda lihtsamaid lahendusi satelliitpiltide saamiseks, kuna kavandatav tehnoloogia nõuab rohkem aja- ja inimressursse, kui on saadav TTÜ Mektory Satelliidiprogrammi raames. Vaatamata sellele, *line scan* kaamerate kasutamine väikestes satelliitides on uudne ja perspektiivne idee, millele tasub tähelepanu pöörata järgmistes satelliidiprogrammides.

Antud lõputöö tuleb kasuks kuupsatelliidi arendajate, kes on huvitatud *line scan* kaamera kasutamisest.

Kasutatud kirjandus

[1] NASA Earth Observatory – Earth Observing 1 (EO-1) : The Earth-Sensing Legacy. [WWW]. http://earthobservatory.nasa.gov/Features/EO1/eo1_2.php. (03.01.2017)

[2] Jensen, J. R. Remote Sensing of the Environment: An Earth Resource Perspective. 2nd ed. NJ: Prentice Hall, 2007.

[3] Kotov, E. *Satelliidi testpingi prototüüp.* [Foto]. WP3, TTÜ Mektory Satelliidiprogramm, 2016.

[4] Swanson, M. Programming for NexStar Controlled Telescopes. [WWW]. http://www.nexstarsite.com/PCControl/ProgrammingNexStar.htm. (03.01.2017)

[5] Celestron, LLC. NexStar Communication Protocol. [WWW]. http://www.celestron.com/media/795779/1154108406_nexstarcommprot.pdf. (03.01.2017)

[6] EPIX, Inc. XCLIB Frame Grabber Programming Library. [WWW]. http://epixinc.com/products/xclib.htm. (03.01.2017)

[7] McConnell, S. Code Complete. 2nd ed. Microsoft Press, 2004, lk. 23-60.

Lisa 1 – Liikumise kontroller

```
import serial
import math

__author__ = "Dmitry Fisun"
__copyright__ = "Copyright 2016 TUT-Mektory Satellite Project"
__credits__ = ["Dmitry Fisun", "Eduard Kotov", "Sander Rikka",
               "Kaan Sadik Karadag"]
__version__ = "0.3"
__maintainer__ = "Dmitry Fisun"
__email__ = "dmitry.fisun@ttu.ee"
__status__ = "Prototype"


connection = serial.Serial()


def connect(port):
    """Connects to the required port if no connection exists.
       Most common values:
       '/dev/ttyS0' for Linux systems
       'COM1' for Windows systems"""
    if not connection.is_open:
        connection.baudrate = 9600
        connection.bytesize = 8
        connection.parity = 'N'
        connection.stopbits = 1
        connection.port = port
        connection.open()


def close():
    """Closes active connection"""
    if connection.is_open:
        connection.close()


def send_request(request, response_length=1):
    """Sends request to the serial port.
       Reads response with the length given (1 byte by default)
       Returns received response or empty string if error occurred"""
    if connection.is_open:
        print("Sending request > " + str(request))
        connection.write(request)
```

```
        response = connection.read(response_length)
        print("Response < " + str(response))
    else:
        print("Error while connecting to the port. Check connection.")
        response = ''
    return response


def start_positive_azimuth_slew(speed):
    start_slew(speed, True, True)


def start_negative_azimuth_slew(speed):
    start_slew(speed, True, False)


def stop_azimuth_slew():
    stop_slew(True)


def start_positive_altitude_slew(speed):
    start_slew(speed, False, True)


def start_negative_altitude_slew(speed):
    start_slew(speed, False, False)


def stop_altitude_slew():
    stop_slew(False)


MAX_SPEED = 9
MIN_SPEED = 1
AZM = 16
ALT = 17
POSITIVE_SLEW = 36
NEGATIVE_SLEW = 37


def start_slew(speed, is_azm, is_positive):
    if MAX_SPEED >= speed >= MIN_SPEED:
        command = bytearray([80, 2,
                             AZM if is_azm else ALT,
                             POSITIVE_SLEW if is_positive else NEGATIVE_SLEW,
                             speed,
                             0, 0, 0])
        send_request(command)
```

```
def stop_slew(is_azm):
    command = bytearray([80, 2,
                         AZM if is_azm else ALT,
                         36, 0, 0, 0, 0])
    send_request(command)


def convert_percents_to_radians(position):
    """Converts position in percent values to radians."""
    position = [component * 2 * math.pi for component in position]
    return position


def convert_percents_to_degrees(position):
    """Converts position in percent values to degrees."""
    position = [component * 360 for component in position]
    return position


REVOLUTION = 65536
POSITION_LENGTH = 10

PRECISE_REVOLUTION = 4294967296
PRECISE_POSITION_LENGTH = 18


def request_position(is_precise=False):
    """Requests position from the mount. Returns array of position components
       with each component describing a percentage of the revolution related
       to the point where the mount was powered on."""
    request = bytearray(b'z') if is_precise else bytearray(b'Z')
    response_length = PRECISE_POSITION_LENGTH if is_precise else
POSITION_LENGTH
    response = send_request(request, response_length)

    current_azm = 0
    current_alt = 0

    if is_precise and len(response) == PRECISE_POSITION_LENGTH:
        current_azm = int(response[0:8], 16) / PRECISE_REVOLUTION
        current_alt = int(response[9:17], 16) / PRECISE_REVOLUTION
    elif not is_precise and len(response) == POSITION_LENGTH:
        current_azm = int(response[0:4], 16) / REVOLUTION
        current_alt = int(response[5:9], 16) / REVOLUTION

    return [current_azm, current_alt]
```

Lisa 2 – Kaamera kontroller

```
from ctypes import *
import time


__author__ = "Dmitry Fisun"
__copyright__ = "Copyright 2016 TUT-Mektory Satellite Project"
__credits__ = ["Dmitry Fisun", "Eduard Kotov"]
__version__ = "0.1"
__maintainer__ = "Dmitry Fisun"
__email__ = "dmitry.fisun@ttu.ee"
__status__ = "Prototype"


xclib = WinDLL("C:\\Program Files\\Epix\\XCLIB\\XCLYBW64.dll")

PIXCI_LIVE = False
last_captured_field = None

FORMAT = "default"
FORMATFILE = "xcvidset.fmt"
UNITMAP = 1
BUF = c_long(1)


def pixci_open():
    xclib.pxd_PIXCIclose()
    i = xclib.pxd_PIXCIopen("", c_char_p(b'default'), None)
    if i < 0:
        print("PIXCI open failed with error code", xclib.pxd_mesgFault(1))
    else:
        print("PIXCI opened successfully with code", i)


def snap():
    if PIXCI_LIVE:
        xclib.pxd_goUnLive(1)
        PIXCI_LIVE = False
    else:
        xclib.pxd_goSnap(1, 1)


def live():
    PIXCI_LIVE = not PIXCI_LIVE
    if PIXCI_LIVE:
        xclib.pxd_goLive(1, 1)
    else:
        xclib.pxd_goUnLive(1)
```

```
def save():
    if PIXCI_LIVE:
        xclib.pxd_goUnLive(1)
        PIXCI_LIVE = False

    if True:
        xclib.pxd_saveBmp(1, "example.bmp", 1, 0, 0, -1, -1, 0, 0)


def timer():
    if PIXCI_LIVE and last_captured_field is not pxd_capturedFieldCount(1):
        last_captured_field = pxd_capturedFieldCount(1)


def test_bmp_saving():
    pixci_open()
    image = create_string_buffer(8)
    result = xclib.pxd_saveBmp(1, c_char_p(b'test.bmp'), 1, 0, 0, -1, -1, 0,
0)


def test_pixci_functions():
    pixci_open()

    print(xclib.pxd_goSnap(1, 1))

    last_captured_buffer = xclib.pxd_capturedBuffer(UNITMAP)
    print(last_captured_buffer, "Last Captured Buffer")

    image_aspect_ratio = xclib.pxd_imageAspectRatio()
    print(image_aspect_ratio, "Image aspect ratio")

    image_width = xclib.pxd_imageXdim()
    print(n_width, "Image x dimension")

    buffer = create_string_buffer(image_width)
    image = xclib.pxd_readuchar(UNITMAP, BUF, 0, 80, n_width, 81, buffer,
n_width, c_char_p(b'GREY'))
    print(image, repr(buffer.raw))

    buffers_field_count = xclib.pxd_buffersFieldCount(UNITMAP, BUF)
    print(buffers_field_count, "Buffers Field Count")

    captured_buffer = xclib.pxd_capturedBuffer(UNITMAP)
    print(captured_buffer, "Captured Buffer")

    xclib.pxd_PIXCIclose()
```