

Hyperspektrikameran ohjelmiston sekä käyttöliittymän tuotekehitys

Joeli Pikkarainen

Opinnäytetyö

Toukokuu 2019

Tekniikan ja liikenteen ala

Insinööri (AMK), tieto- ja viestintätekniikka koulutusohjelma

Ohjelmistotekniikka

Tekijä(t) Pikkarainen, Joeli	Julkaisun laji Opinnäytetyö, AMK	Päivämäärä Toukokuu 2019
	Sivumäärä 34	Julkaisun kieli Suomi
		Verkojulkaisulupa myönnetty: x
Työn nimi Hyperspektrikameran ohjelmiston sekä käyttöliittymän tuotekehitys		
Tutkinto-ohjelma Tieto- ja viestintätekniikka		
Työn ohjaaja(t) Rantonen, Mika; Saarisilta, Juha		
Toimeksiantaja(t) Senop Oy		
<p>Tiivistelmä</p> <p>Opinnäytetyön tarkoituksena oli toimia osana Senop Oy:n tuotekehitystiimiä, kehittää ja dokumentoida hyperspektrikameran PC-ohjelmistoa ja hyperspektrikuvantamisessa hyödynnettäviä fysikaalisia ilmiöitä. Sovelluskehityksellä pyrittiin luomaan PC-ohjelmiston käyttöliittymästä ja käyttölogiikasta käyttäjäystävällisempää. Fysikaalisiin ilmiöihin perehtymällä pyrittiin keräämään tietoa, jonka avulla on helppo perehdyttää uusi henkilö hyperspektrikuvantamisen aihepiiriin.</p> <p>Työ toteutettiin tapaustutkimuksen näkökulmasta, koska työn tarkoitus oli tuottaa haluttuja lopputuloksia ohjelmistokehityksen kannalta. Työn alkaessa todettiin ensin kehitettävän ohjelmiston tila ja listattiin toivottuja kehityskohteita ja ominaisuuksia. Omaisuuksien kehitys priorisoitiin suhteuttamalla kehityskohteen tarpeellisuus ja toteutukseen arvioitu aika.</p> <p>Tuotekehitysprosessin aikana resursseja jaettiin PC-ohjelmiston kehittämisen lisäksi muille prosessin ohjelmistojen tutkimukselle ja kehittämiselle sekä yhteistyökumppanien kanssa toteutettavalle yhteistyölle.</p> <p>Opinnäytetyön tuloksena saatiin aikaan paranneltu PC-ohjelmisto, ohjelmistokomponentteja kuvaavia dokumentteja, yhteistyön tuottamia tuloksia, jotka mahdollistavat uudenlaisia hankkeita tulevaisuudessa, lisää tietämystä fysikaalisista ilmiöistä, sekä laitteen toiminnasta.</p>		
Avainsanat (asiasanat) Hyperspektrikamera, Hyperspektrikuvantaminen, Ohjelmistokehitys		
Muut tiedot (salassa pidettävät liitteet)		

Author(s) Pikkarainen Joeli	Type of publication Bachelor's thesis	Date May 2019
		Language of publication: Finnish
	Number of pages 34	Permission for web publication: x
Title of publication Development of software and user interface for hyperspectral camera		
Degree programme Information and communication technology		
Supervisor(s) Rantonen, Mika; Saarisilta Juha		
Assigned by Senop Oy		
<p>Abstract</p> <p>The purpose of the thesis was to be part of Senop Oy's product development; develop and document the client and the physical phenomena regarding hyperspectral imaging and the camera PC software. The software development aimed to make the PC client and its operation logic more user friendly. The physical phenomena were researched to produce information by which a new person can familiarize themselves to this subject matter.</p> <p>The thesis was carried out as a case study, because the aim of the study was to develop results for the software development of an existing application. At the beginning of the development process, the state of existing application was studied and the most important features were agreed upon. The priority of the developed features was decided based on the necessity and the time estimated for the development.</p> <p>During the development the resources were shared between the PC client, the research and development of other production applications and affiliate collaboration.</p> <p>The thesis resulted in an improved PC application, documentation of some parts of the software, possibilities for new projects with cooperation affiliates regarding new software implementations, increased knowledge of the physical phenomena and the functionality of the hyperspectral camera.</p>		
Keywords/tags (subjects) Hyperspectral camera, Hyperspectral imaging, Software development		
Miscellaneous (Confidential information)		

Sisältö

1	Opinnäytetyön lähtökohdat.....	4
1.1	Toimeksiantaja	4
1.2	Työn tavoitteet	4
1.3	Tutkimusmenetelmät	5
2	Hyperspektrikuvaus.....	5
2.1	Teoria.....	5
2.2	Optiikka.....	6
2.3	Fabry-Perot interferometri.....	7
2.4	CMOS-kuv sensori	8
2.5	FPGA	9
3	Hyperspektrikuvantamisen sovellukset.....	10
3.1	Yleistä	10
3.2	Spektri.....	10
3.3	Agrikulttuuri	11
3.4	Lääketiede	12
3.5	Rikostutkimus	12
4	ENVI	12
4.1	Datakuutio	13
4.2	Header	13
5	Kehitysympäristö.....	16
6	Ohjelmisto.....	17
6.1	Ohjelmiston yleiskuvaus.....	17
6.2	PC-ohjelmisto	18

7	PC-käyttöliittymän muutokset	21
7.1	Snapshot ja kuvantoisto	21
7.2	Sessioiden tallennus	21
7.3	RGB-kuvan korjaus	23
7.4	Referenssianalyysi	24
7.5	Saturoituneen spektrin taseus	26
7.6	Muut muutokset.....	28
8	Testaus.....	30
9	Kehitysideat	30
9.1	Luokittelijan integrointi	30
9.2	Suoratoiston konfigurointi	31
10	Yhteenveto.....	32
	Lähteet	33
	Liitteet	35

Kuviot

Kuvio 1. Värikuvan ja datakuution vertailu	6
Kuvio 2. FPI-Illustraatio	7
Kuvio 3. CMOSIS lohkokaavio.....	9
Kuvio 4. Sähkömagneettinen spektri (Electromagnetic radiation, n.d).....	11
Kuvio 5. HSI-kehitysympäristö.....	16
Kuvio 6. Ylätason ohjelmistodiagrammi.....	18
Kuvio 7. Käyttöliittymän Device-välilehti	19
Kuvio 8. Datavisualization-välilehti ja luokkadiagrammi	20
Kuvio 9. Värikuvauksen korjaus.....	23
Kuvio. 10 Referenssin laskentaan käytetty funktio.....	25

Kuvio 11. Erbium-referenssin kuvaajat	26
Kuvio 12. Korjaamaton ja korjattu saturoitunut spektri	27
Kuvio 13. Saturaation tarkistus	27
Kuvio 14. Datapisteen osoitin	29
Kuvio 15. Osoittimen piirtäminen	29
Kuvio 16. HSC ja luokittelija tuotantolinjastolla	31

Taulukot

Taulukko 1. Header-tiedoston kentät ja niiden selitykset (ENVI Header Files, n.d.) ...	15
--	----

Sanasto

FPGA	Field Programmable Gate Array
FPI	Fabry–Pérot Interferometri
CMOSIS	Complementary Metal Oxide Semiconductor Image Sensor
HSC	Hyperspectral Camera
SENOP HSI	Senop Hyper Spectral Imager

1 Opinnäytetyön lähtökohdat

1.1 Toimeksiantaja

Opinnäytetyön toimeksiantaja oli Senop Oy. Senop Oy on Pohjoismaiden suurimman pimeänäkölaitteiden valmistajan Millog Oy:n perustama tytäryhtiö, joka keskittyy modernien sensoriteknologioiden kehittämiseen ja vaatimaan järjestelmäintegraatioon. Senop Oy:n toiminta alkoi 1.1.2016, jolloin Millog Oy:n optroniikkaliiketoiminta ja Oricopa Oy:n integrointi liiketoiminta keskitettiin toimeksiantajan yhtiöön. (Senop Oy keskittyy sensorijärjestelmien kehittämiseen puolustus- ja turvallisuusviranomaisien tarpeisiin 2016.)

Senop Oy:n tuotteet koostuvat suureksi osaksi viranomaisille tarkoitetuista turvallisuusjärjestelmistä ja -ratkaisuista, kuten valonvahvistimista, pimeätähtäimistä ja lämpökamera-, valvonta- ja paikannusjärjestelmistä. Yhtiö tarjoaa myös taktisten mobiili- ja taistelunjohtojärjestelmien kehittämistä ja valmistamista sekä modernisointia ja järjestelmäintegroitihankkeiden toteuttamista. Senop Oy ja Millog Oy ovat osa Patria-konsernia. (Senop Oy keskittyy sensorijärjestelmien kehittämiseen puolustus- ja turvallisuusviranomaisien tarpeisiin 2016.)

1.2 Työn tavoitteet

Opinnäytetyön tavoitteena oli kehittää toimeksiantajan SENOP-HSI PC-ohjelmistoa, joka on Windows 7 ja 10-käyttöjärjestelmillä toimiva ohjelma hyperspektrikuvien analysointiin ja hyperspektrikameran ohjaamiseen. Työn alussa tutkittiin PC-ohjelmiston sen hetkisiä ominaisuuksia ja käyttöliittymän toimintaa, ohjelmistossa käytettäviä rajapintoja sekä hyperspektrikameran toimintaa. PC-ohjelmalla pystyi alkuvaiheessa seuraamaan suoratoistoa kamerasta, kaappaamaan datakuutioita, lataamaan kuvausdataa kamerasta, analysoimaan kyseistä dataa, päivittämään kameran laiteohjelmiston ja asettamaan kameraan uusia kuvaussekvenssejä.

PC-ohjelmistolle oli tarkoitus kehittää uusia toimintoja, jotka helpottavat sen käyttöä ja mahdollistavat erilaisia data-analyyssejä. Nämä toiminnot tuovat valmiille kameralle lisäarvoa asiakkaan kannalta.

Tarkoituksena oli myös dokumentoida SENOP-HSI ohjelmisto mahdollista jatkokehitystä varten. Kyseisen projektin kehittämiseen liittyvän henkilön on tarkoitus saada kattava kuva projektin teknologioista tämän opinnäytetyön avulla.

1.3 Tutkimusmenetelmät

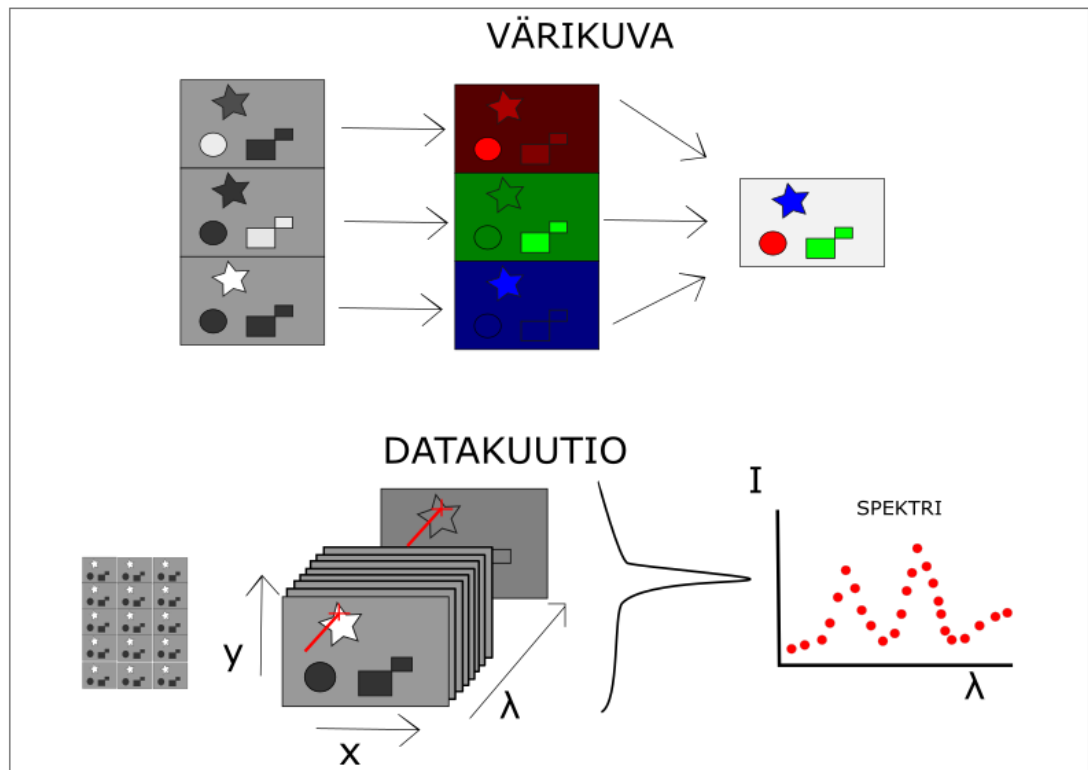
Opinnäytetyö suoritettiin tapaustutkimuksena, koska kehityksen kohteena oli jo valmiiksi tunnettu ilmiö ja kehitetty ohjelmisto. Teoriatiedon perusta on tarkoitus kerätä dokumenteista, joita Senop käyttänyt tuotteen kehityksessä. Kokeelliset aineistot, kuten ohjelmisto-ominaisuuksien demonstrointi ja testaus tullaan tuottamaan itse käyttämällä itse kerättyä mittausdataa ja uusia ohjelmistokäännöksiä. Ohjelmistokehitykseen perustuvat tutkimukset tullaan perustelemaan kyseisten teknologioiden ja protokollien virallisilla dokumentaatioilla. Perehdytystä aiheeseen tullaan myös samaan yhteistyökumppanien järjestämistä koulutustapahtumista ja seminaareista.

2 Hyperspektrikuvaus

2.1 Teoria

Hyperspektrikuvantaminen on spektrofotometrian sovellus. Sen avulla voidaan tunnistaa eri materiaaleja analysoimalla niiden uniikkia elektromagneettista spektriä.

Toisin kuin moderni digitaalivalokuvaus, jossa kamera kaappaa kolme aallonpituuskaistaa punaiselta, vihreältä ja siniseltä näkyvän valon alueelta, joista muodostetaan värikuva, hyperspektrikamera kaappaa jopa satoja kaistoja, joista muodostetaan kolmiulotteinen datakuutio. Datakuution avulla päästään käsiksi jokaisen kaapatun pikselin spektriin. Kuviossa 1 nähdään ero normaalin valokuvauksen ja hyperspektrikuvauksen lopputuloksista.



Kuvio 1. Värikuvan ja datakuution vertailu

Hyperspektrikuvaksen toiminta perustuu siis siihen, että pystytään kaappaamaan monokromaattista valoa kohteesta usealla aallonpituudella. Mitä tarkemmalla välillä aallonpituuksia voidaan kuvata suurelta alueelta, sitä tarkempia analyysyjä kaapattusta kuvasta voidaan tehdä.

SENOP HSC-kameran kuvankaappaus perustuu valoa tarkentavaan optiikkaan, valon suodattimena toimivaan Fabry-Pérot interferometriin (FPI), kahteen CMOSIS -kennoon, ohjelmoitavaan FPGA-mikropiiriin ja integroituun laiteohjelmistoon.

2.2 Optiikka

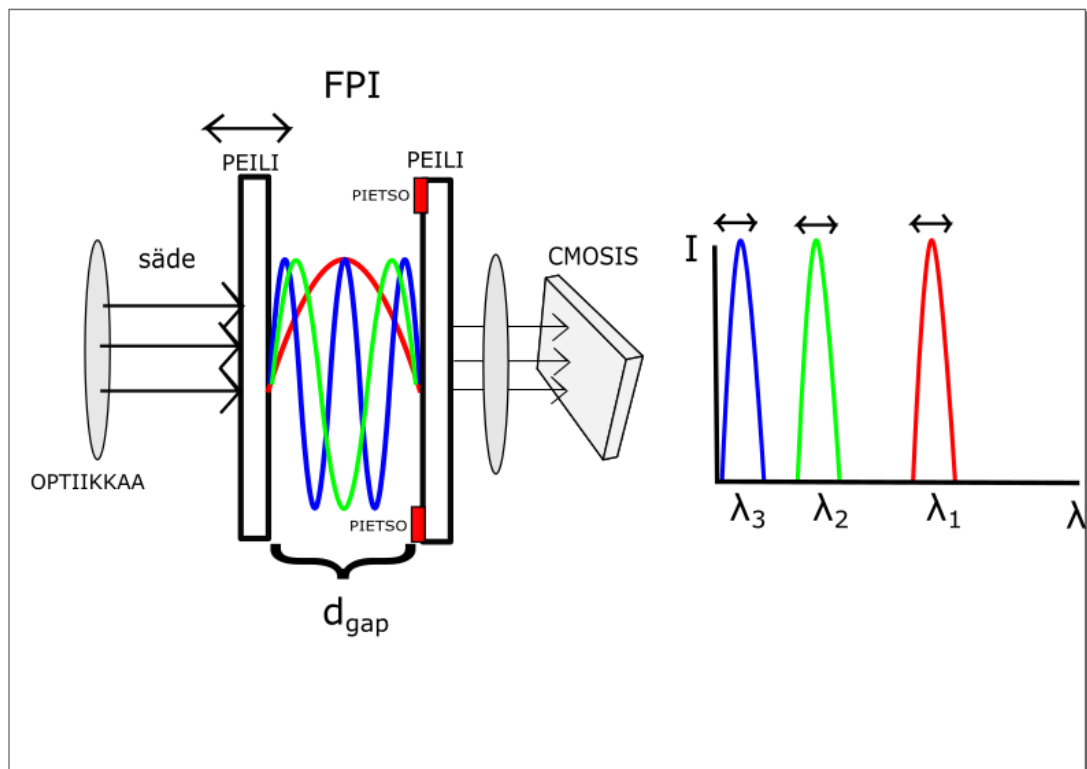
Optiikan tarkoitus kameroissa on kerätä, ohjata ja kohdistaa valoa optiselle sensorille. (Optics n.d).

SENOP-HSC:ssä objektiivin keräämä valo ohjataan ensin optiikan avulla FPI:lle, joka suodattaa valosta halutun aallonpituuden. FPI:n läpäisevät säteet ohjataan optiikan avulla optisille sensoreille.

2.3 Fabry-Perot interferometri

Fabry-Perot interferometri on kaistanpäästösuoatim, jonka toiminta perustuu kah-
teen osittain heijastavaan kohtisuorassa olevaan lasipintaan. FPI:hin ohjattu valo hei-
jastuu lasipintojen välillä, mikä aiheuttaa interferenssin. Samassa vaiheessa olevat sä-
teet aiheuttavat rakentavan interferenssin, minkä seurauksena kaikki samassa vai-
heessa olevat säteet siirtyvät FPI:stä läpi. Eri vaiheessa olevat säteet taas aiheuttavat
dekonstruktivisen interferenssin ja suodattuvat pois. (Saari 2018, 3.).

Yksinkertaistettu illustraatio ilmiöstä nähdään kuviossa 2.



Kuvio 2. FPI-Illustraatio

FPI:tä voidaan käyttää säädettävänä filtterinä ja halutun säteen aallonpituus valitaan muuttamalla FPI:n heijastavien pintojen ilmaväliä kaavalla 1.

$$2d_{gap} \cos(\theta) = (m + \frac{\varepsilon(\lambda)}{\pi})\lambda \quad (1)$$

missä d_{gap} = heijastavienpintojen ilmaväli

θ = valon tulokulma

m = kertaluku

$\varepsilon(\lambda)$ = säteen vaihesiirtymä.

(Saari 2018.)

Ideaalitulanteessa, jossa oletetaan tulokulman θ olevan nolla astetta $\cos(0^\circ) = 1$ ja vaihesiirtymän $\varepsilon(\lambda)$ olevan π voidaan kaava yksinkertaistaa muotoon

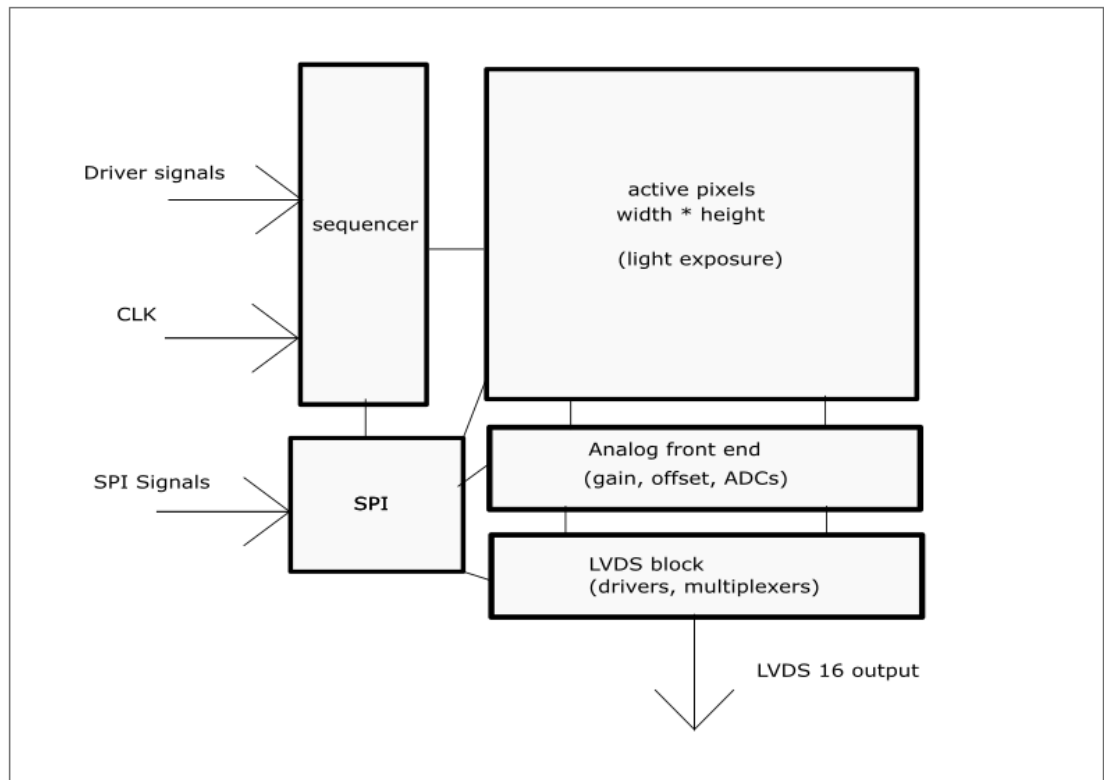
$$d_{gap} = m \left(\frac{\lambda}{2}\right) \quad (2)$$

josta nähdään, että ilmavälille d säädetyt pinnat päästävät läpi aallonpituuden $\frac{\lambda}{2}$ monikerrat m .

FPI:n peilien etäisyyden muuttaminen vaati suurta tarkkuutta. Tämän lisäksi etäisyyksiä tulee pystyä muuttamaan todella nopeasti, jotta kehyksien kaappaamiseen kuluva aika ei tule sietämättömän pitkäksi. Tähän ratkaisuna FPI:hin on integroitu kolme pietsoelektronista komponenttia.

2.4 CMOS-kuvasensori

Complementary Metal Oxide Semiconductor Image Sensor (CMOSIS) on useista metallioksidisista puolijohteista rakennettu sensori. Sensorin altistaminen fotoneille saa aikaan varauksen puolijohteissa, joka kuvastaa kyseiselle pikselille päästetyn valon määrää. CMOSIS toimii global shutter periaatteella, jossa kaikkien pikselien valotus luetaan samanaikaisesti. Puolijohteiden varaus muutetaan 10- tai 12-bittiseksi digitaaliksi signaaliksi riippuen kuvasmoodista ja siirretään vahvistimen ja multiplekserin läpi ja lopulta LVDS-väylää pitkin FPGA:lle. FPGA pystyy ohjaamaan kuvasensoreita SPI-väylän kautta. (ks Kuvio 3).



Kuvio 3. CMOSIS lohkokaavio

2.5 FPGA

Field Programmable Gate Array (FPGA) on puolijohteista rakentuva ohjelmoitava lo-
giikkapiiri. FPGA:n uudelleenohjelmoitavuus mahdollistaa helpomman suunnittelun
ja käyttöönoton sekä pienentää riskejä suunnittelussa tapahtuneissa virheissä. Uu-
delleenohjelmointi mahdollistaa myös bugikorjausten ja jatkuvan kehityksen tuote-
kehitysprosessiin. FPGA:n suunnittelu on myös halvempaa, eikä FPGA:n korkeampi
hintaa Application Specific Integrated Circuits (ASIC)-piireihin verrattuna ole kyseisen
tuotteen tuotantomäärään nähden liiallinen, varsinkaan jatkuvasti kehittyvässä tuot-
teessa. (What is an FPGA? n.d.)

SENOP HSC:n FPGA:lla kuvasensoreilta tulleelle kuvalle tehdään mustan tason poisto,
kuolleen pikselin poisto ja kuvan rajausta. FPGA:lta kuvadata voidaan siirtää joko Disp-
lay Porttia pitkin näytölle suoratoistona, tallentaa laitteen SSD:lle tai ladata tietoko-
neelle PC-ohjelmiston avulla, jossa se muunnetaan datakuutioksi.

FPGA kommunikoi SPI-väylän kautta signaaleja FPI:ltä CMOSIS:lle, koska FPI:n siirtyessä suuria etäisyyksiä joudutaan kennolla odottamaan ennen kuvan ottamista, jotta FPI ehtii asettua oikeaan asentoon.

3 Hyperspektrikuvantamisen sovellukset

3.1 Yleistä

Hyperspektrikuvantamisen avulla saatavan datakuution ansiosta, sillä on käyttötarkeituuksia monella eri tutkimuksen ja teollisuuden alalla. Tavassa tarkastella kappaleita niiden spektrin avulla on paljon hyviä puolia, mikä mahdollistaa tarkempia, helpompia ja turvallisempia analyysejä. Hyperspektrikuvaamisen on monia etuja.

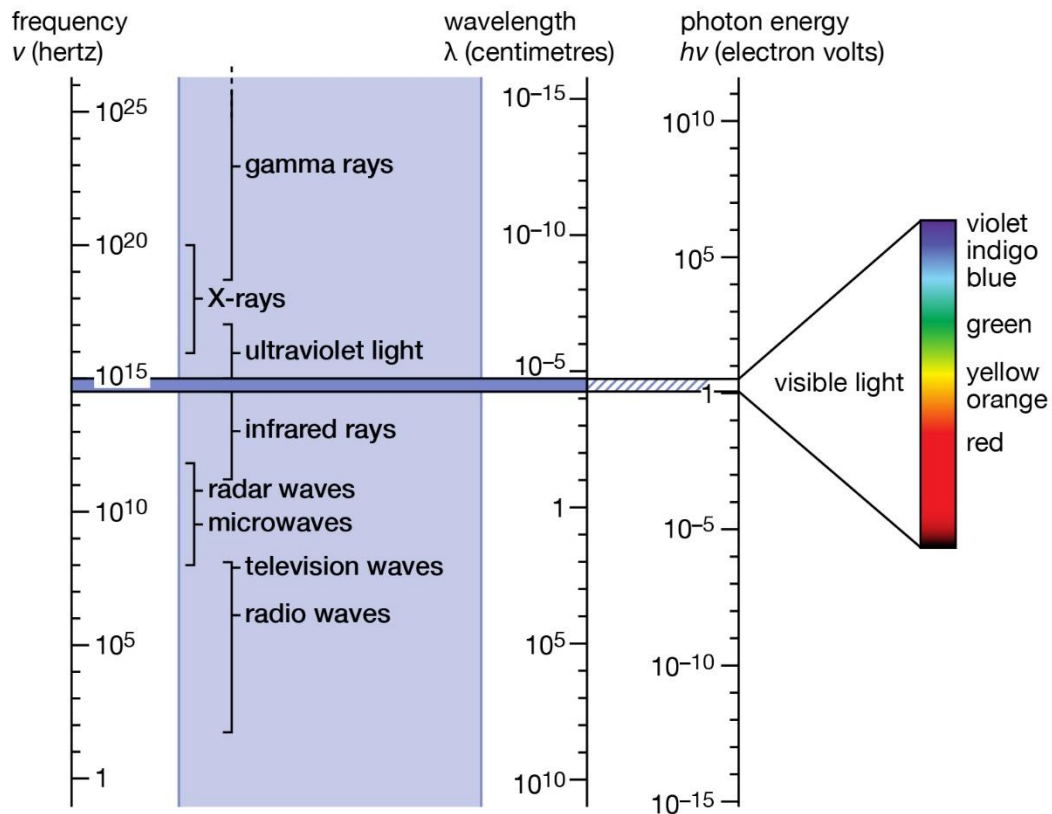
Näytekappaletta ei tarvitse valmistella, eikä sen kanssa tarvitse olla fyysisessä kosketuksessa. Tämän ansiosta hyperspektrikuvantamisella voidaan tutkia kappaleita, jotka ovat kosketusherkkiä tai vaarallisia. Kemiallisessa analyysissä näytteestä on pakko ottaa koekappale ja valmistella mahdolliset laboratorioympäristöt. Vapaus koekappaleen ottamisesta takaa näytteen eheyden ja mahdollistaa ihmiselle vaarallisen näytteen ottamisen turvallisesti, sekä mahdollistaa näytteen ottamisen uudelleen, mikäli alkuperäinen koekappale tuhoutuu tai saastuu. (Da-Wen Sun 2010, 8)

Hyperspektrikuvaus mahdollistaa datan keräämisen suuresta näyteosiesta. Esimerkiksi droneen kiinnitetty hyperspektrikamera voi kuvata kokonaisen pellon, jonka avulla voidaan analysoida sadon laatua. Satelliittiin integroitu hyperspektrikamera mahdollistaa suuren maa-alan mineraalikartoituksen.

3.2 Spektri

Sähkömagneettisella spektrillä tarkoitetaan joukkoa elektromagneettisia partikkeleita, jotka on järjestetty jonkin yhteisen fyysisen ominaisuuden mukaan, kuten aallonpituus tai taajuus. (Spectrum n.d)

Sähkömagneettinen spektri järjestetään fysiikassa usein taajuuden tai aallon pituuden mukaan, kuten kuviossa 4. SENOP HSC:n tapauksessa tarkasteltavat aallonpituudet ovat näkyvävalon (SWIR) 400 – 700 nanometriä ja lähi-infrapun (NIR) 700 – 1000 nanometriä alueella. Tämä on kuitenkin vain kapea osa koko sähkömagneettisesta spektristä.



© Encyclopædia Britannica, Inc.

Kuvio 4. Sähkömagneettinen spektri (Electromagnetic radiation, n.d)

3.3 Agrikulttuuri

Agrikulttuurissa hyperspektrikuvantamisen avulla voidaan selvittää sadon määrää ja laatua. Hyperspektrikuvaaminen miehittämättömästä ilma-aluksesta (UAV) mahdollistaa mosaiikkikuvan muodostamisen useasta datakuutiosta, josta voidaan analysoida maa-alueella olevan biomassan määrää. Ilma-aluksesta kuvatessa tulee ottaa huomioon monia virheen aiheuttavia tekijöitä, koska ulkoilmassa kuvatessa altistutaan usein epäoptimaalisille valaistusolosuhteille. Mittausvirhettä voidaan ilmakuvauksessa korjata esimerkiksi maahan asetetulla tunnetulla referenssikohdalla ja

ilma-alukseen asennetulla irradianssisensorilla. Kuvauksesta tuotettujen analyysien avulla maanviljelijät voivat optimoida tarvittavan lannoitteen määrää, sekä valvoa sadon laatua. (Honkavaara 2013.)

3.4 Lääketiede

Lääketieteessä hyperspektrikuvantamista voidaan käyttää muun muassa ihotautien analysointiin. Hyperspektrikameralla on onnistuttu tunnistamaan melanooman levinneisyyttä ihossa. Hyperspektrikameran avulla on onnistuttu arvioimaan melanooman rajaa tarkemmin, kuin yleisesti käytettävillä kliinisillä tutkimustavoilla. Hyperspektrikuvantaminen on myös tutkimustapana nopeampi ja vaatii vähemmän kokemusta ja valmisteluja. (Neittaanmäki-Perttu 2015.)

Hyperspektrikameralla on mahdollisesti myös muita lääketieteellisiä tutkimuskohteita kuten erilaiset kasvaimet, ihottumat, tulehdukset, kehon eritteiden kemikaalipitoisuudet ja lääkkeiden tunnistus.

3.5 Rikostutkimus

Rikostutkimuksessa hyperspektrikuvantamisen hyödyt perustuvat laajalti mahdollisuuden analysoida eri kohteiden uniikkia spektriä ja tehdä analyysjä ilman kemiallisia metodeja ja laboratorio-olosuhteita. Hyperspektrikuvantamisen hyötynä rikostutkimuksen tapauksessa on, että data voidaan kerätä näytteestä mahdollisimman nopeasti, mikä takaa analysoitavan näytedatan eheyden. Myös mahdollisesti vaarallisia näytteitä voidaan analysoida ilman suoraa fyysistä kontaktia, mikä takaa myös tutkijoiden turvallisuuden. Vaikka rikostutkimuksen sovellukset ovat varmasti yksi haastavimmista tutkimuksista käytännössä, on sillä tulevaisuudessa varmasti hyödyllisiä käyttötarkoituksia tutkimusten ja analysointityökalujen ja luokittelijoiden kehityksessä. (Puupponen 2014.)

4 ENVI

ENVI (Environment for Visualizing Images) on Harris Geospatial Solutions yhtiön luoma ohjelmistoympäristö, joka on tarkoitettu hyperspektrikuvien analysointiin ja

hyödyntämiseen. SENOP HSI sovelluksessa käytetään soveltavasti ENVI:n tiedostomuotoa datakuutioiden tallentamiseen ja käsittelyyn. ENVI tiedostot muodostuvat datakuutiosta ja header-tiedostosta ja sitä käytetään yleisesti mallina eri hyperspekt-rikuvantamisen ratkaisuihin etenkin kaukokartoituksessa, jossa hyperspektri kamera on liitetty ilma-alukseen tai satelliittiin. (CIRES, 1967-2002. Pioneering a Succesfull Partnership 2002.)

4.1 Datakuutio

SENOP HSI:n Datakuutioon tallennetaan kuvauksesta saatu raakadata. SENOP HSI:n datakuutio on flätti .dat päätteinen binääritiedosto, jonka metatiedot ovat erillisessä header-tiedostossa.

SENOP HSI käyttää datakuutiossaan BSQ (Band Sequential) formaattia. BSQ-formaatissa pikselien arvot on järjestetty alkamaan vasemmasta yläreunasta rivi kerrallaan jokaiselle kuvatulle aallonpituudelle. Tämä on optimaalinen formaatti yksittäisen pikselin datan käsittelyyn xy-akselilla mitattua aallonpituutta kohden. (ENVI Image Files n.d.)

4.2 Header

Header on datakuutiota täydentävä hdr-päätteinen tekstitiedosto, joka sisältää metadatan kuvauksesta. Header-tiedosto muodostetaan PC-ohjelmistossa datan latauksen yhteydessä, kun jokaisen erillisen kehyksen header jäsennetään yhteen header-tiedostoon.

SENOP HSI:n datakuution header sisältää seuraavia tietoja: Sanalla "senop" alkavat rivit ovat SENOP HSI:n omia parametrejä, eivätkä periydy ENVI:n formaatista. Header-tiedoston kentät selitetään taulukossa 1. Tiedosto alkaa merkkijonolla **ENVI**.

description	Tiedostoa kuvaava teksti.
wavelength units	Aallonpituuden yksikkö, nm eli nanometri.

samples	Datapisteiden määrä x-akselilla
lines	Datapisteiden määrä y-akselilla
file type	Oletuksena ENVI
interleave	Datan järjestys kuutiassa. Datan järjestys SENOP HSI:ssä on BSQ eli Band Sequential. BSQ:ssa data on järjestetty siten, että datapisteet alkavat ensimmäisen kehyksen ylimmältä riviltä vasemmasta reunasta ja etenevät rivi kerrallaan kehyksen loppuun, jonka jälkeen alkaa seuraava kehyks.
byte order	Eli tavujärjestys. Oletuksena 1 eli Most Significant Byte first (MSF) eli eniten merkitsevä tavu luetaan ensin.
data type	Tiedoston tavujen muoto. Oletuksena 12 , eli unsigned integer 16-bit (uint16). Kokonaislukujen lukujoukon avaruus siis $0_{10} - 65535_{10}$ tai $0x00_{16} - 0xFFFF_{16}$.
senop sequence name	Sekvenssin nimi, jota kuvauksessa käytettiin.
bands	Kuvattujen kehysten määrä.
acquisition time	Kuvauksen ajanhetki formaatissa YYYY-MM-DDTHH:MM:SS.DZ , jossa YYYY on vuosi MM on kuukasi DD on päivämäärä T on päivämäärän ja tunnin erotin, kirjaimellisesti "T" HH on kellonajan tunnit MM on kellonajan minuutit SS on kellonajan sekunnit D on sekuntien desimaalit Z on aikavyöhyke formaatissa UTC
wavelength	Lista kuvatuista aallonpituuksista.

fwhm	Full Width Half Maximum eli kuvatun aallonpituuden puoliarvoväleveys nanometreinä (nm). Haetaan laitteen kalibrointiedostoista.
data gain values	Lista vahvistuskertoimista, jonka avulla saadaan datakuvauksen pisteet yksikköön $\frac{W}{m^2 * \mu m * sr}$
senop sequence order	Lista, jonka mukaan kaikki muut aallonpituutta vastaavat listat järjestetään.
senop order	Aallonpituuden kuvauksessa käytetty kertaluku, eli FPI:n asento.
senop timestamp	Kuvauksen aloitus hetkestä kulunut aika kyseisen aallonpituuden kuvaukseen nanosekunteina.
senop sensor	Kehyksen kuvaukseen käytetty sensori VIS tai NIR.
senop frame counter	Lista joka kertoo järjestyksen, jossa aallonpituudet kaapattiin koko kuvausskriptin ajan aikana.
senop user counter	Vanhentunut, ei käytössä, tullaan poistamaan käytöstä.
senop repeat	Vanhentunut, ei käytössä, tullaan poistamaan käytöstä.
senop integration time	Lista kehysten kaappaamisessa käytetyistä valotusajoista millisekunteina, ms .
senop gps	Lista GPS-moduulin tallentamista geolokaatioista.
senop acceleration	Lista kiihtyvyyssanturin tallentamista x,y,z-kiihtyvyyssarvoista.
senop gyroscope	Lista gyroskoopin tallentamista x,y,z-asentoarvoista.

Taulukko 1. Header-tiedoston kentät ja niiden selitykset (ENVI Header Files, n.d.)

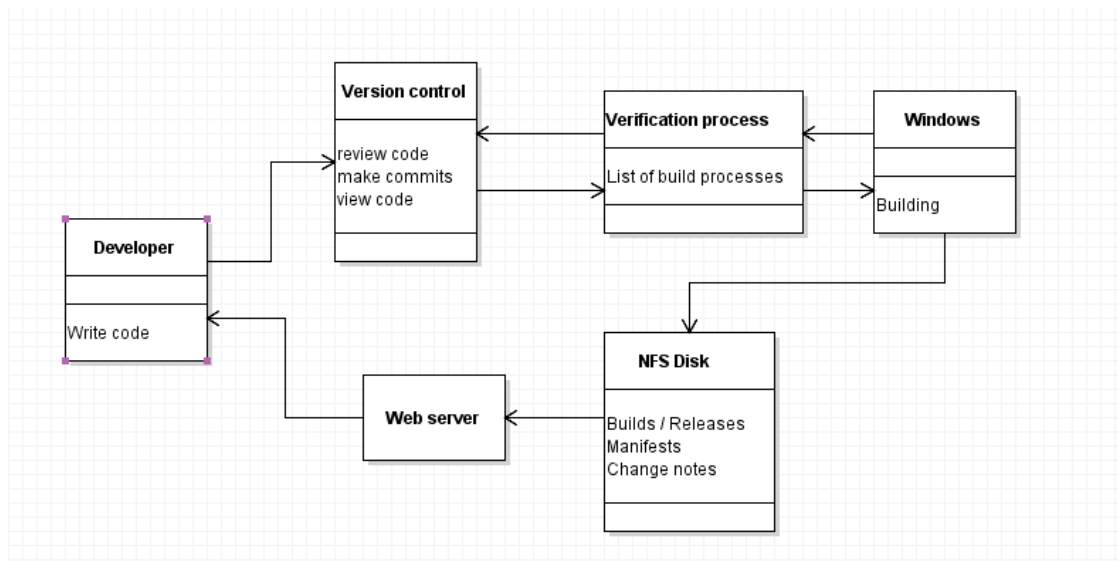
SENOP HSI:n kuvausdatasta ja sitä käsittelevistä ohjelmistokomponenteista on laadittu dokumentaatio, jonka on tarkoitus olla avuksi tuotekehitystiimin jäsenille, ohjelmistokehittäjille sekä yhteistyökumppaneille ja asiakkaille, jotka haluavat kehittää omaa PC-ohjelmistoaan SENOP HSC:lle. Tarkempaa tietoa SENOP HSI:n datasta ja sitä jäsentelevistä ohjelmistokomponenteista on liitteessä 1.

5 Kehitysympäristö

Senop HSI:n PC-ohjelmiston ohjelmointiympäristössä käytetään Qt:n ohjelmistokehitystä. Qt:n ohjelmistokirjasto tarjoaa laajan kehyksen ja projektikonfiguroinnin, sekä PC- että sulatettujen ohjelmistojen kehittämiseen. Julkaistavan PC- ohjelmiston lähdekoodi on avoin.

Senop HSI:n virallinen julkaisukäännösympäristö koostuu useista virtuaalikoneista, joissa tehdään koodikatselmointia, versionhallintaa, verifiointeja ja virallisia käännöksiä. Virtuaalikoneet toimivat Windows 10 PC:n Virtual Box-virtualisointiympäristössä.

Kuviossa 5 nähdään kehitysympäristö suhteessa yksittäiseen ohjelmistokehittäjään. Ideaalisesti yksittäisen kehittäjän tarvitsee käyttää vain versionhallintarajapintaa organisaatiossa sovitun toimintamallin mukaan. Muu toiminta käännösympäristössä on automatisoitua.



Kuvio 5. HSI-kehitysympäristö

Versionhallintatyökalu on työkalu, joka on kehitetty parantamaan ohjelmistokehityksen versionhallintaa. Versionhallinta helpottaa monen kehittäjän yhtäaikaisen työskentelyn samassa ohjelmointiprojektissa. Versionhallinta vaatii yhden tai useamman ohjelmoijan katselmoimaan koodiin tehdyt muutokset. Tehtyjä muutoksia ei päästetä master-haaraan ennen kuin, tarvittavat katselmoinnit on hyväksytty. Tämän projektin kehitysketjussa vaaditaan myös automatisoitujen verifiointien, testien sekä käännöksen onnistuminen. Hyväksytty katselmointi aloittaa automatisoidut testit ja käännökset automatisoidulla virtuaalikoneella toimivalla palvelimella.

Käännöspalvelimella on automatisoitu virtuaalikone, johon on listattu kaikki projektiin kuuluvat verifiointi- ja käännösprosessit. Tämän ratkaisun on tarkoitus helpottaa jatkuvaa kehitystä projektissa ja se mahdollistaa testien, koodikäännösten ja julkaisujen luonnin automatisoinnin. Prosesseja voidaan seurata ja ohjata selaimessa toimivasta käyttöliittymästä. Prosessit alkavat, kun versionhallintaan tuodaan uusia muutoksia.

SENOH HSI PC-ohjelmistot käännetään erillisellä Windows 10-käännös-PC:llä. Tällä tavalla varmistetaan aidosti natiivi Windows-ympäristö julkaisukäännökselle.

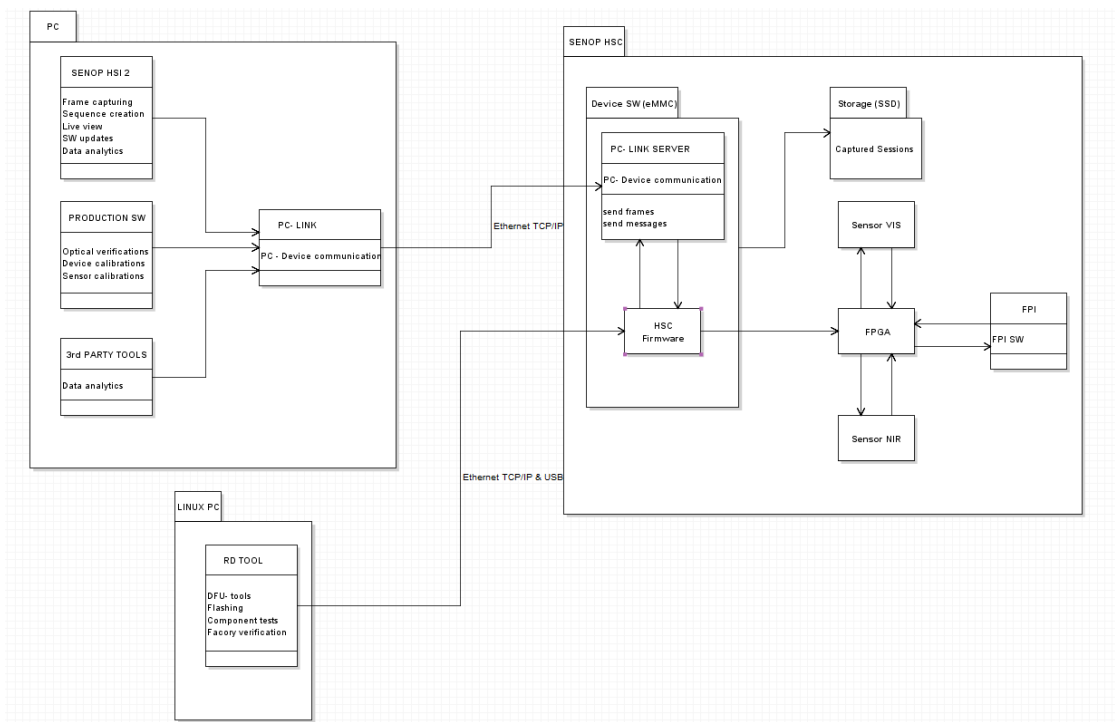
Ohjelmistokehityksen lopulliset käännökset ja asennusohjelmat tallennetaan onnistuneen käännösprosessin jälkeen Web-palvelimelle, joka käyttää NFS-verkkolevyjärjestelmää. Tältä palvelimelta voidaan noutaa asiakkaille toimitettavat laite- ja PC-ohjelmistot.

6 Ohjelmisto

6.1 Ohjelmiston yleiskuvaus

SENOH hyperspektrikameran ohjelmisto muodostuu kolmesta pääkomponentista, joihin kuuluu PC-ohjelmistot, hyperspektrikameran laiteohjelmisto ja laitteen piirikortille räätälöity laiteohjelmistojulkaisu.

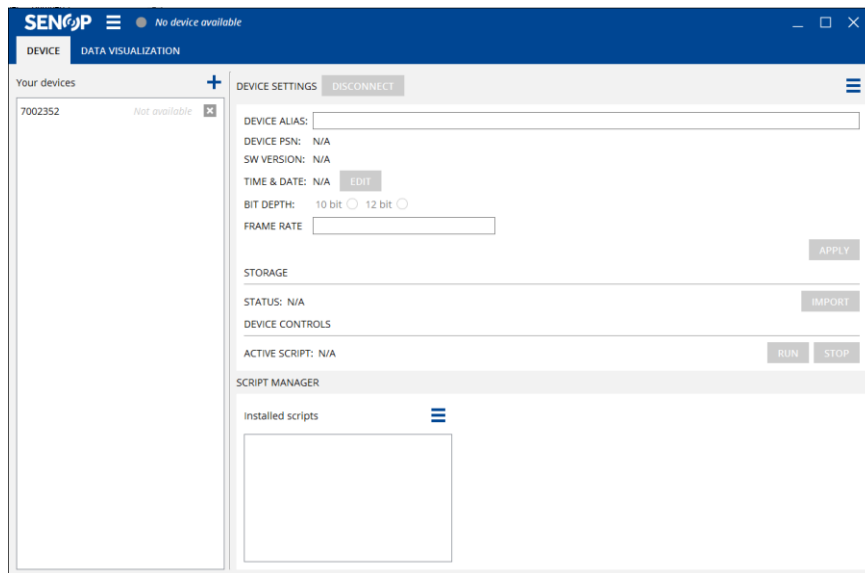
Kuviossa 6 näytetään ohjelmiston pääkomponentit ja niiden väliset yhteydet. SENOP HSI 2 on asiakkaalle tarkoitettu PC-ohjelmisto. Production SW on tuotannossa käytetty kalibrointiohjelmisto. RD TOOL on Linux PC:llä toimiva testausohjelmisto, jolla voidaan ladata uusi laiteohjelmisto piirikortin flash-muistiin ja FPI:lle, sekä ajaa tehtaalla suoritettavia testejä ja verifiointeja.



Kuvio 6. Ylätason ohjelmistodiagrammi

6.2 PC-ohjelmisto

Senop HSI 2 on loppukäyttäjälle tarkoitettu graafisella käyttöliittymällä toteutettu ohjelmisto. Kuviossa 7 on PC-ohjelmiston Device-välilehti. Tässä näkymässä käyttäjä pystyy yhdistymään laitteeseen ethernet- kaapelin kautta, katsomaan laitteen tietoja, lataamaan laitteelle kuvausskriptejä, lataamaan sessioita laitteelta, säätämään kuvauksen asetuksia ja päivittämään laitteen ohjelmiston.



Kuvio 7. Käyttöliittymän Device-välilehti

Kuviossa 8 on SENOP HSI 2 Datavisualization-välilehti ja ohjelmistokomponenttien yhteydet. Tällä välilehdellä käyttäjä voi katsoa suoratoistoa laitteelta, tutkia kuvattuja datakuutioita ja niiden spektrejä ja luoda uusia kuvasskriptejä. Skriptien toiminta on selitetty tarkemmin liitteen 1 kappaleessa Capturing Sessions (Scripts and Sequences).



Kuvio 8. Datavisualization-välilehti ja luokkadiagrammi

7 PC-käyttöliittymän muutokset

7.1 Snapshot ja kuvantoisto

PC-ohjelmiston kuvantoiston ja kuvankaappauksen käyttölogiikkaa muutettiin. Sen sijaan, että ohjelmisto tarjosi vain yhden painikkeen, joka hoiti sekä suoratoiston aloittamisen pysäyttämisen ja snapshotin kaappaamisen, tehtiin snapshotin kaappauselle oma painike ja muutettiin entinen painike ohjaamaan vain suoratoiston aloittamisen ja pysäyttämisen. Ominaisuuden muutos parantaa käyttäjän kokemusta, koska se mahdollistaa kuvan pysäyttämisen ilman että ohjelmisto kaappaa sekvenssin joka kerta kun suoratoisto halutaan pysäyttää.

Toivottu ominaisuus pystyttiin toteuttamaan olemassa olevia funktioita muuttamalla siten, että toisto-pysäytys painike ei kutsu enää snapshotin aloittavaan funktiota vaan tästä vastaa nykyään toivottu snapshot-painike. Painikkeet aktivoidaan PC-ohjelmistossa olevan tilakoneen avulla. Käyttäjä pystyy siis selvästi näkemään, milloin laite on tilassa, joka mahdollistaa suoratoiston tai snapshotin kuvaamisen.

Lisäominaisuutena snapshotin ottamiseen tulisi kirjoittajan mielestä myös implementoida kuvien latauksen käytettävä funktio, joka muodostaa .png-kuvatiedostot jokaisesta tallennetusta kehyksestä.

7.2 Sessioiden tallennus

PC-ohjelmiston kaappaamien snapshottien tallentaminen oli toteutettu siten, että snapshotit tallennettiin tietokoneen tiedostojärjestelmään ennalta määritettyyn tilapäiseen kansioon. Tämä oli käyttäjän kannalta keho ominaisuus sillä kokematon käyttäjä ei välttämättä ollut tietoinen mihin tiedosto tallennettiin. Nyt käyttäjällä on mahdollisuus painaa tallennuspainiketta, kun tämä tahtoo tallentaa kuvatun datakuution. Tallennus painike avaa käyttäjälle geneerisen tallennusdialogin, jossa tämä voi määrittää halutun kansion nimen ja sijainnin tallennettaville tiedostolle. Tallennuksen toiminnallisuus toteutettiin säilyttämällä avatun session sijainti tiedosto jär-

jestelmässä ja kopioimalla tiedostot sieltä käyttäjän määrittämään kansioon. Tallennuksen aikana tarkistetaan, ettei kopioitavassa hakemistossa ole tiedostoja, joiden pääte ei vastaa snapshotissa tallennettavia tai laitteelta ladattaessa tulevien tiedostojen päätteitä. Vain oikeanlaiset tiedostot tallennetaan; .png jokaisesta framesta muodostettu kuvatiedosto, .hdr datakuution header-tiedosto ja .dat binäärimuotoinen datakuutio.

7.3 RGB-kuvan korjaus

Laitteen RGB-kuva ei ollut hyvä, koska ohjelmistoon oli ennalta määrätty RGB kanavien aallonpituudet. Aallonpituudet oli määritetty koodissa, joten käyttäjällä ei ollut mahdollisuutta korjata itse RGB-kuva. Ohjelmistoa korjattiin niin, että pystytään konfiguroimaan aallonpituudet, jotka toimivat sovelluksessa paremmin. Tällä ratkaisulla saatiin sovelluksessa näkymään parempi RGB-kuva.

Kuviossa 9 nähdään vanhan ohjelmiston värikuva, korjattu värikuva ja referenssinä käytetty väritaulu. Korjattu kuva on edelleen hieman kellertävä, joten parametrejä kannattaa konfiguroida lisää vielä jatkossa.



Kuvio 9. Värikuvauksen korjaus

Värikuvaukselle olisi hyödyllistä jatkokehittää lisää toiminnallisuutta ja antaa käyttäjälle mahdollisuus määrittää omat parametrit värikuvan muodostukseen. Käyttäjän

tulisi itse pystyä määrittämään värikuvassa käytettävät aallonpituudet ja värikuvanmuodostuksessa käytettävät värit. Tämä on toteutettavissa antamalla käyttäjälle mahdollisuus kirjoittaa toivotut aallonpituudet sekvenssin muodostus valikossa ja implementoida konenäkö ohjelmisto kuten Open CV (Open Computer Vision). Tämä mahdollistaisi myös helposti erotettavien spektrien nopean analysoinnin suoratoiston aikana.

7.4 Referenssianalyysi

Yleinen metodi spektrin analysoinnissa on valita tunnettu spektri referenssiksi, johon voidaan verrata toista spektriä. Tavoitteena olisi siis pystyä hakemaan toivottu referenssispektri datakuutiosta ja verrata sitä seuraavaksi valittuun näytteeseen. Ominaisuutta toteuttaessa tuli myös ottaa huomioon, että ohjelmiston liukusäädin näyttää käyttäjälle oikeat arvot referenssispektristä. Referenssispektri lasketaan kaavalla;

$$\frac{\text{Sample} - \text{Reference}}{\text{Reference}} \quad (3)$$

, missä *sample* on tutkittava spektri ja *reference on* referenssinä käytetty spektri. Referenssin laskenta ja mallien muodostaminen tehdään suurimmaksi osaksi kuvion 10 funktiolla.

```

void SpectrumWidget::calculateReference()
{
    //Remove the series from chart to get ownership and modify
    auto rmSeries = m_refMapper->series();
    if (rmSeries != nullptr) {
        chart()->removeSeries(rmSeries);
    }
    //Can't make reference without having a ref- point. Shouldn't be possible to get here without
    but still check
    if(m_referencePoints.isEmpty()){
        return;
    }

    //New series to add data to mapper
    auto *series = new QScatterSeries();
    m_calculatedRef = qobject_cast<QXYSeries*>(series);

    m_calcRefModel.clear();//Clear this model, it will be made again.
    m_calcRefModel.setShowIntensities(true);

    QMap<qreal,qreal> intensities;//This will be set to the refModel
    //Radiometric mode to get sample spectrum, Cant use Relative values
    //Compare this point and reference point.
    QList<QPointF> sampledata = m_activeModel->getIntensities(SpectrumModel::Mode::Radiometric);
    qreal referenced_p;//Value of the calculated ref intensity
    qreal wl; //Wavelength
    //Save min and max intensity values
    cs.maxY = -DBL_MAX;
    cs.minY = DBL_MAX;
    int wlindex = 0;
    //Calculation of the referenced point is done here. [ (sample - ref) / ref ]

    foreach (QPointF dataPoint, sampledata) {
        if(m_referencePoints.at(wlindex).y() == 0){
            referenced_p = 0;
        }
        else{
            referenced_p = (dataPoint.y() - m_referencePoints.at(wlindex).y());
            referenced_p = referenced_p / m_referencePoints.at(wlindex).y();
        }
        QPointF calucaltedPoint(dataPoint.x(),referenced_p);
        m_calculatedRef->append(calucaltedPoint);

        wl = dataPoint.x();
        intensities[wl] = referenced_p;

        //Catch values range
        if(referenced_p != 0){
            if(cs.maxY < referenced_p){
                cs.maxY = referenced_p;
            }
            if(cs.minY > referenced_p){
                cs.minY = referenced_p;
            }
        }
        m_calcRefModel.add(wl, referenced_p);
        wlindex++;
    }
    //Set the calculated intensities to the model and give it to the mapper
    m_calcRefModel.setIntensities(intensities);
    m_refMapper->setModel(&m_calcRefModel);
    m_refMapper->setSeries(qobject_cast<QXYSeries*>(m_calculatedRef));
    //Make series //Add it to chart //reset chart
    resetIntensitySeries(m_refMapper);
}

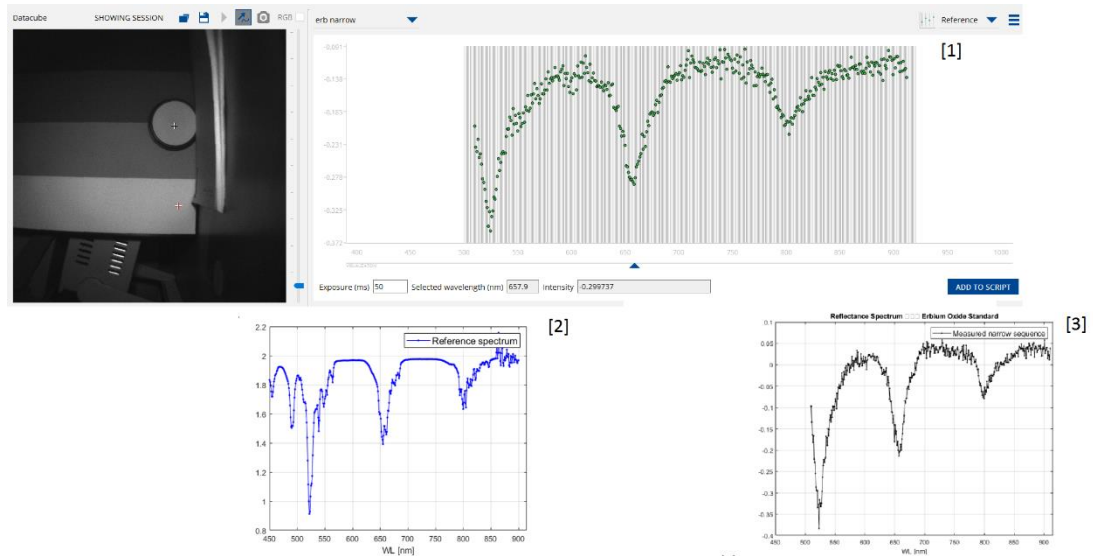
```

Kuvio. 10 Referenssin laskentaan käytetty funktio

Referenssi analyysillä voidaan tutkia toisiaan muistuttavia spektrejä tarkemmin, kuin absoluuttisesta kuvaajasta, sekä sillä voidaan tehdä optisia verifiointeja laitteen kalibrointiin liittyen, kuten erbiumin spektrin vertaaminen aallonpituustauluun.

Kuviossa 11 on vertailussa SENOP HSI:n referenssianalyysillä tehty erbium-spektri, erbium-spektrin kirjallisuusarvo ja vanha MATLAB-ohjelmistolla tehty kuvaaja erbium

referenssin mittauksesta, jollaisia käytetään HSC:n optisessa verifiointissa. Uudella referenssianalyysillä pystytään selvästi löytämään kaksi erbiumin absorptiopiikkiä.



Kuvio 11. Erbium-referenssin kuvaajat

Ohjelmistoa kehittäessä kannatti minimoida tai mahdollisesti eliminoida olioiden toiminnan muuttuminen, vaikka näiden jäsenolijoille tehtäisiinkin muutoksia. Kuvaajan aktiivisen spektrin olio pidetään omassa muuttujassaan, joka palautetaan liukusäädinoliolle, jotta voidaan säilyttää samaan aikaan useita spektrejä.

7.5 Saturoituneen spektrin tasaus

Aikaisemmassa ohjelmistoversiossa ei kerrottu käyttäjälle, milloin valittu spektri on mahdollisesti saturoitunut. Käyttäjälle olisi kriittistä kertoa milloin spektri on saturoitunut tai tasoittaa saturoituneet arvot suurimpaan tunnettuun saturoimattomaan arvoon. Kuviossa 11 on kaksi spektriä, joista ylemmälle ei ole tehty saturaation korjausta ja alemmalle on. Aallonpituuksilla 520 – 640 sensori on saturoitunut, mutta ylemmää kuvaajaa voidaan luulla datan olevan validia. Alemmassa kuvaajassa saturoituneet arvot on muutettu suurimpaan tunnettuun validiin arvoon. Käyttäjä näkee alemman spektrin kuvaajasta, että data on saturoitunutta, eikä sitä kannata käyttää.



Kuvio 12. Korjaamaton ja korjattu saturoitunut spektri

Kuviossa 13 on osa spektrin intensiteetin laskemiseen käytettävästä funktiosta, johon on lisätty tarkistus, jonka avulla tunnistetaan saturoituneet arvot.

```

for (; lines > 0; line++, lines--) {
    const quint16 *ptr = toUInt16Ptr(data.constData() + (line * imageLines + sample) * static_cast<int>(sizeof(quint16)));
    for (int count = samples; count > 0; ptr++, count--) {
        #if 1
        //
        qDebug() << "quint16 from cube : " << *ptr;
        rawsum += *ptr;
        sum += hsiRadiance(*ptr, gain);
        #else
        // Alternative solution to calculate from original 12 bit value (12.4) format
        qreal val = (*ptr) >> 4;
        val += (static_cast<qreal>(*ptr) & 0xF) / static_cast<qreal>(0xF);
        sum += hsiRadiance(val, gain);
        #endif
    }
    if (useCorrection) {
        if ((rawsum / pixels) >= s_saturatedPixel && gain > 0) {
            qDebug() << "this is saturated : " << rawsum / pixels << gain;
            return s_saturatedPixel;
        }
        else {
            qDebug() << "this is good " << rawsum / pixels;
            return (sum / pixels);
        }
    }
    else {
        return (sum / pixels);
    }
}

```

Kuvio 13. Saturaation tarkistus

7.6 Muut muutokset

Käyttöliittymään tehtiin myös seuraavia käyttökokemusta parantavia muutoksia. Kenttä jossa näytetään intensiteetin tarkka arvo. Ennen intensiteetin arvoa pystyi vain arvioimaan kuvaajasta tai tallentamalla tiedoston csv:nä ja avaamalla sen toisessa ohjelmistossa.

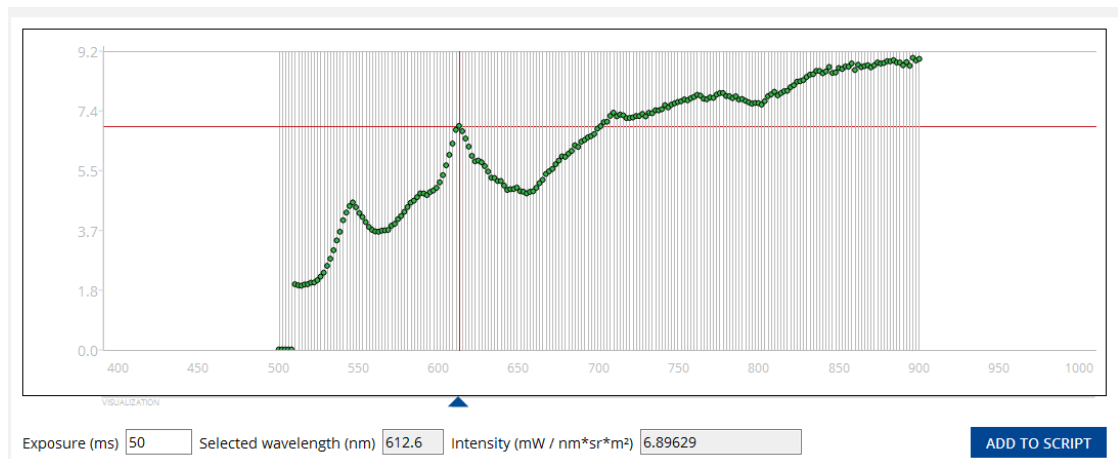
Sessioiden avaaminen oli toteutettu siten, että käyttäjän tuli navigoida geneerisessä tiedostodialogissa hakemistoon, jossa datakuutio ja header-tiedosto ovat. Tämä aiheutti tilanteita, jossa käyttäjä saattoi yrittää avata tiedostoa väärästä hakemistosta, jonka seurauksena mikään sessio ei avautunut. Ominaisuutta muutettiin siten, että käyttäjä näkee nykyään hdr-tiedostot tiedostodialogissa ja avaa session valitsemalla sen.

Histogrammiin lisättiin pykäliä ja akseleille minimi ja maksimi arvot, jotka muuttuvat punaiseksi riippuen siitä epäilläanko kuvan olevan yli- tai alivalottunut.

Käyttöliittymän tekstikentät muutettiin sopivan kokoisiksi ja niille lisättiin niitä kuvaavat yksiköt. Kenttien taustat muutettiin harmaiksi, joihin käyttäjä ei voi antaa syötettä.

Suoratoistonäkymän yläpuolelle lisättiin kenttä, joka kertoo käyttäjälle missä tilassa ohjelmisto ja kamera ovat. Tilat ovat Idle, Live, OpeningSession, ShowingSession ja TakingSnapshot.

Spektrin kuvaajaan lisättiin visualisointia helpottava osoitin (ks. Kuvio 14). Osoittimen avulla on helpompi seurata oikeaa aallonpituutta kapeiden piikkien kohdilla.



Kuvio 14. Datapisteen osoitin

Osoittimen punaiset viivat piirretään olion yliajetussa piirtofunktiossa (ks. Kuvio 15). Funktiossa tunnistetaan kohta, jossa on valittu aallonpituus (x- koordinaatti) ja sitä vastaava intensiteetti (y-koordinaatti).

```
qreal pointerPos = chart()->mapToPosition(QPointF(m_activeWL,m_activeIntensity)).y();
for (int i = 0; i < rows; i++) {
    QModelIndex wix = model->index(i, SpectrumModel::sm_wavelengthColumn);
    wlpos = (model->data(wix).toDouble() - minX)/(maxX - minX);
    int x = x0 + static_cast<int>((x1 - x0) * wlpos);
    painter->drawLine(x, y0, x, y1);

    if( model->data(wix).toDouble() == m_activeWL){
        painter->setPen(QPen(QColor(s_colorRed), 1, Qt::SolidLine));
        //Vertical marker
        painter->drawLine(x, y0, x, y1);
        //Horizontal marker
        painter->drawLine(x0, pointerPos, x1, pointerPos);
        painter->setPen(QPen(QColor(s_colorGreyHover), 1, Qt::SolidLine));
    }
}
```

Kuvio 15. Osoittimen piirtäminen

8 Testaus

Tuotekehitysprojektissa suoritetaan kahta erilaista testiä; Factory Acceptance Test (FAT) ja julkaisutesti.

FAT suoritetaan aina tehtaalla kootulle kameralle. FAT:in tarkoitus on tehdä laitteelle optinen verifiointi, jolla varmistetaan optiikan ja FPI:n toimivuus, että kalibroinnissa saadut tiedot ovat sallituissa rajoissa ja laitteen ja sen liitännärajapintojen yleinen toiminta. Jokaiselle laitteelle tehdään omat FAT, josta laaditaan mittauspöytäkirja vaikkeksi laitteen toiminnasta.

Julkaisutesti on tarkoitettu uusien ohjelmistoversioiden toiminnan testaamiseen. Julkaisutesti suoritetaan aina kun uusi laite- tai PC-ohjelmisto halutaan julkaista asiakkaalle. Testiin kuuluu ohjelmistojen yksittäisten ominaisuuksien testausta, yhteensopivuus testausta ja liittymärajapintojen testausta. Julkaisutestissä tulee siis listata kaikkien moduulien ohjelmistoversiot, eli laiteohjelmiston, FPI:n ja PC-ohjelmiston versio.

9 Kehitysideat

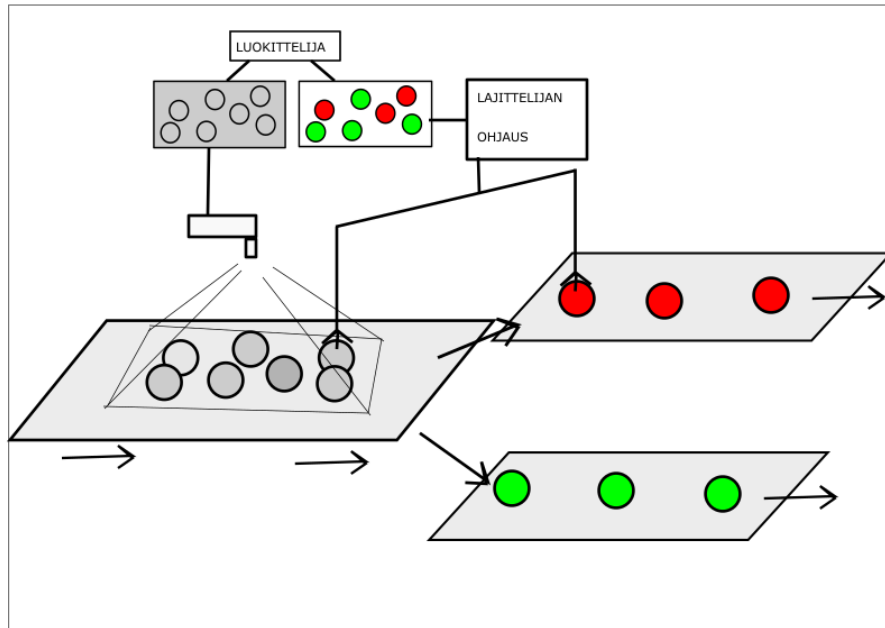
9.1 Luokittelijan integrointi

Opinnäytetyön aikana tehtiin yhteistyötä kolmannen osapuolen ohjelmistokehittäjän kanssa, joka oli kehittänyt tekoälyyn perustuvan sovelluksen hyperspektrikuvien luokitteluun. Ohjelmistokehittäjän ohjelmistossa on määritelty projektityyppejä erilaisille hyperspektridatoille. Yhteistyön tuloksena ohjelmistokehittäjä pystyi integroimaan ohjelmistonsa projektityypin, joka pystyy lukemaan SENOP HSI-datakuutioita. Luokittelijaohjelmiston yleisin käyttötarkoitus on käyttöliittymään perustuva sovellus, jossa käyttäjä määrittää datakuutiosta tunnetut materiaalit, joiden avulla luokittelija luo mallin, jota voidaan käyttää uusien datojen luokitteluun.

Luokittelijaohjelmistoon kuuluu myös ajonaikainen (runtime) ohjelmistolaajennus, jonka avulla voitaisiin mahdollisesti tehdä datan keräystä ja luokittelua samanaikaisesti. Käytännössä integroitu luokittelija toimisi siten, että käyttäjä luo luokittelijalle

mallin tunnetuista materiaaleista, ja käynnistää kameran, joka datakuution kuvattuun lataisi datakuution PC-ohjelmistolle, jossa se ohjattaisiin luokittelijan läpi.

Tällaista sovellusta voitaisiin käyttää esimerkiksi tehtaiden ja tuotantolaitosten linastoilla automaattisen laadunvalvonnan välineenä tai syötteenä linjastolla toimivalle mekaaniselle laitteelle, joka lajittelee erilaiset materiaalit toisistaan, joiden eroja ei kuitenkaan voi huomata ihmissilmällä (ks. kuvio 16).



Kuvio 16. HSC ja luokittelija tuotantolinjastolla

Kuvausolosuhteet on kuitenkin pidettävä hallittuina ja uusien materiaalien tunnistukseen on koulutettava luokittelijaa uudestaan. Luokittelijaohjelmiston rajapintaa voidaan ohjelmoida C-kielellä ja kääntää osana SENOP HSI:tä. Ajonaikaista ohjelmistolaajennusta testattiin SENOP HSI PC-ohjelmistossa onnistuneesti valmiiksi generoidulla datalla ja luokittelijalla.

9.2 Suoratoiston konfigurointi

SENOP HSI:n suoratoistoon tulisi lisätä enemmän konfigurointi mahdollisuuksia, kuten väärävärivärien muodostavien kehysten aallonpituudet ja niihin lisättävien värien

muuttaminen. Suoratoistoon tulisi voida valita lisäksi myös yksittäisiä kehyksiä halutulta aallonpituudelta. Oikeat aallonpituudet löytämällä ja parannellun suoratoiston konfiguroinnilla käyttäjä voisi luoda spesifisesti räätälöityjä suoratoistoja eri käyttötarkoituksiin. Suoratoiston konfigurointia on jo demonstroitu muokkaamalla RGB-kuvan parametrejä.

Tällaisella ominaisuudella uskotaan olevan käyttötarkoituksia esimerkiksi lääketieteellisissä toimenpiteissä. Oikein konfiguroitu suoratoisto voisi helpottaa tietyn laisten kudosten löytämisen, jota on silmällä vaikea havaita. Ominaisuus voisi myös vähentää tarvetta näytekappaleiden ottamiselle ja mahdollistaa tutkittavan alueen kartoituksen olematta suorassa kontaktissa kohteeseen. Konfiguroitavaa suoratoistoa voitaisiin käyttää myös mahdollisesti erilaisten naamiointivälineiden erottamisen ihmisilmälle näyttävälle saman värisestä ympäristöstä. Esimerkiksi geneerisellä taisteijan luminaamion ja lumen sekä kesänaamion ja kasvuston spektreillä voidaan olettaa olevan huomattavia eroja.

10 Yhteenveto

Voidaan todeta, että työn aikana tehdyt kenttätutkimukset ja fysikaaliseen ilmiöön kohdistuvat tutkimukset jäivät vähäisiksi, ohjelmistokehityksen viedessä suurimman osan resursseista. Toisaalta tutkimuksia eri käyttötapauksista tullaan jatkossakin saamaan yhteistyökumppanien, sekä asiakkaiden toimesta. PC-ohjelmiston kehittäminen toivotulle tasolle mahdollistaa tulevaisuudessa resurssien siirtämistä myös hankkeissa suunniteltaviin laajempiin ohjelmistokokonaisuuksiin.

Työn aikana onnistuttiin tekemään yhteistyötä kolmannen osapuolen kanssa, joka tuo luokittelijan integrointimahdollisuuden kautta paljon uusia mahdollisuuksia ja käyttösovelluksia HSC:lle.

SENOP HSI-ohjelmiston kehittämisen lisäksi onnistuttiin hankkimaan laajalti tietoa myös RD TOOLin ja Production SW:n osalta, joka helpottaa mahdollisia tuotannon ja kalibroinnin aiheuttamia ongelmia tulevaisuudessa.

Opinnäytetyön alussa asetetut tavoitteet onnistuttiin saavuttamaan tyydyttävästi. PC-ohjelmistolle saatiin toteutettua suurin osa toivotuista ominaisuuksista ja parannuksista ja näistä onnistuttiin tekemään toimivia julkaisuja, jotka parantavat ohjelmiston käytettävyyttä ja toimivuutta. PC-ohjelmiston kehitys on tilassa, jossa uutta ominaisuutta voidaan heti alkaa suunnittelemaan työnaikana hankitun tiedon avulla. SENOP HSC:tä voidaan myös alkaa lähitulevaisuudessa viemään uusiin hankkeisiin myös PC-ohjelmiston osalta, mikäli resursseja ei kulu paljoa tuotannon tai tuotekehityksen muihin osa-alueisiin.

Lähteet

Da-Wen Sun. 2010. Hyperspectral Imaging for Food Quality Analysis and Control. Elsevier Inc.

Electromagnetic radiation. N.d. Encyclopedia Britannica. Tieteellinen verkkosanakirja. Viitattu 14.5.2019. <https://www.britannica.com/science/electromagnetic-radiation>.

ENVI Header Files. N.d. Harris Geospatial Solutions. Verkkodokumentaatiokokoelma. Viitattu 14.5.2019. <https://www.harrisgeospatial.com/docs/ENVIHeaderFiles.html>.

ENVI Image Files. N.d Harris Geospatial Solutions. Verkkodokumentaatiokokoelma. Viitattu 14.5.2019. <https://www.harrisgeospatial.com/docs/enviimagefiles.html>.

Honkavaara, E., Saari, H., Kaivosoja, J., Pölönen, I., Hakala, T., Litkey, P., Mäkynen, J. & Pesonen, L. 2013. Processing and Assessment of Spectrometric, Stereoscopic Imagery Collected Using a Lightweight UAV Spectral Camera for Precision Agriculture. Artikkel. Jyväskylän Yliopisto. Viitattu 14.5.2019. <https://jyx.jyu.fi/handle/123456789/42524>.

Kisslinger, C. 2002. CIRES, 1967-2002. Pioneering a Successful Partnership. University of Colorado at Boulder. Viitattu 14.5.2019. <https://web.archive.org/web/20110927184848/http://cires.colorado.edu/about/history/06a.html>

Neittaanmäki-Perttu, N., Grönroos, M., Jeskainen, L., Pölönen, I., Ranki, A., Saksela, O. & Senllman, E. 2015. Delineating Margins of Lentigo Maligna Using a Hyperspectral Imaging System. Artikkel. Jyväskylän Yliopisto. Viitattu 14.5.2019. <https://jyx.jyu.fi/handle/123456789/48498?locale-attribute=en>.

Optics. N.d Koulutusmateriaali Opto Engineeringin verkkosivuilla. Viitattu 14.5.2019. https://www.opto-e.com/media/downloads/docs/Basics_Optics_EN.pdf.

Puupponen, H. 2014. Unmixing Methods in Novel Applications of Spectral Imaging.

Väitöskirja. Jyväskylän Yliopisto. Viitattu 14.5.2019. <https://jyx.jyu.fi/handle/123456789/44940>.

Saari, H. 2018. Introduction to Piezo-actuated Fabry-Perot Interferometer tunable filter technology. VTT Microspectrometers.

Senop Oy keskittyy sensorijärjestelmien kehittämiseen puolustus- ja turvallisuusviranomaisten tarpeisiin. Patria Oyj. 2016. Tiedote Patria Oyj:n kotisivulla. Viitattu 2.2.2019. <https://www.patria.fi/fi/media/tiedotteet/senop-oy-keskittyy-sensorijarjestelmien-kehittamiseen-puolustus-ja>.

Spectrum. N.d. Dictionary.com Verkkosanakirja. Viitattu 14.5.2019. <https://www.dictionary.com/browse/spectrum>.

What is an FPGA?. N.d. Xilinx, Inc. Verkkoartikkeli. Viitattu 14.5.2019. <https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html>.

Liitteet

Liite 1. ENVI Data in SENOP HSI Application

ENVI Data in SENOP HSI Application

Date: 13.5.2019 Joeli P

ENVI

Environment for Visualizing Images. Interface/Standard/Software for special geospatial image processing and analyzing. SENOP HSI uses guidelines from ENVI data format that is used in spectral imaging. This document is meant to give a description of the SENOP HSI's data format and how it is handled programmatically.

Official website of ENVI Harris Geospatial Solutions <https://www.harrisgeospatial.com/Software-Technology/ENVI>

Contents

What data can we get from SENOP HSC	9
Capturing Sessions (Scripts and Sequences)	9
Data difference in Hyperspectral camera and imported data	11
Imported/ Translated data	13
dat- file	13
hdr- file	13
Fields of the header.....	14
Description	14
Wavelength units	14
Samples	14
Lines	14
File type	14

Byte order.....	15
Datatype	15
Senop sequence name	16
Bands 16	
Acquisition time	16
Wavelength	16
FWHM 16	
data gain values.....	17
Senop sequence order	17
Senop order.....	17
Senop timestamp	17
Senop sensor	17
Senop frame counter	17
Senop user counter	17
Senop repeat	17
Senop integration time	18
Senop gps	18
Senop acceleration.....	18
Senop gyroscope	18
Class and file description of SENOP HSI.....	18
class EnviHeader::Private : public QSharedData (<i>enviheader.cpp</i>)	18
EnviHeader& EnviHeader::operator=(const EnviHeader &other).....	18
bool EnviHeader::parse(const QString &headerFile).....	19
const QString &EnviHeader::getDescription() const	19
const QString &EnviHeader::getSequenceName() const	19

const QString &EnviHeader::getWavelengthUnits() const	19
const EnviHeader::Time &EnviHeader::getAcquisitionTime() const.....	19
int EnviHeader::getSamples() const	19
int EnviHeader::getLines() const	19
const QString &EnviHeader::getFileType() const	19
const EnviHeader::InterleaveType &EnviHeader::getInterleave() const	20
int EnviHeader::getByteOrder() const	20
int EnviHeader::getDataType() const	20
int EnviHeader::getBands() const	20
const QList<Frame::ESensor> &EnviHeader::getSensor() const	20
const QList<int> &EnviHeader::getFrameCounter() const.....	20
const QList<qreal> &EnviHeader::getFWHM() const	20
const QList<int> &EnviHeader::getSequenceOrder() const	20
int EnviHeader::getSequenceOrder(int index) const	21
const QList<int> &EnviHeader::getOrder() const	21
int EnviHeader::getOrder(int index) const.....	21
QString EnviHeader::getOrderAsString(int index).....	21
const QList<int> &EnviHeader::getUserCounter() const.....	21
const QList<int> &EnviHeader::getRepeat() const	21
const QList<qint64> &EnviHeader::getTimeStamp() const.....	21
const QList<qreal> &EnviHeader::getIntegrationTime() const.....	22
const QList<qreal> &EnviHeader::getDataGainValue() const.....	22
const QList<Frame::GPSFormat> &EnviHeader::getGPSFormat() const.....	22
const QList<QString> &EnviHeader::getGPS() const	22
QGeoPositionInfo EnviHeader::getGeoPosition(int index) const	22
const QList<qreal> &EnviHeader::getGeoPoint(bool recalculate) const	22

const QList<QString> &EnviHeader:: getAcceleration () const	23
Frame::Vector3D EnviHeader:: getAcceleration (int index)	23
const QList<QString> &EnviHeader:: getGyroscope () const	23
Frame::Vector3D EnviHeader:: getGyroscope (int index)	23
void EnviHeader:: reset ()	23
bool parse (const QString &headerFile)	23
void parseDescription (const QString &value)	23
void parseSequenceName (const QString &value)	24
void parseWavelengthUnits (const QString &value)	24
void parseAcquisitionTime (const QString &value)	24
void parseWavelength (const QString &value)	24
void parseSamples (const QString &value)	24
void parseLines (const QString &value)	24
void parseFileType (const QString &value)	24
void parseInterleave (const QString &value)	25
void parseBands (const QString &value)	25
void parseSequenceOrder (const QString &value)	25
void parseOrder (const QString &value)	25
void parseSensor (const QString &value)	25
void parseByteOrder (const QString &value)	25
void parseDataType (const QString &value)	26
void parseFrameCounter (const QString &value)	26
void parseFWHM (const QString &value)	26
void parseUserCounter (const QString &value)	26
void parseRepeat (const QString &value)	26
bool parseDoubleList (QList<qreal> &list, const QString &value)	26

void parseIntList (QList<int> &list, const QString &value)	26
void parseStringList (QList<QString> &list, const QString &value)	27
void parseStringListWithQuotes (QList<QString> &list, const QString &value)	27
void parseTimestamp (const QString &value).....	27
void parseISO8601 (EnviHeader::Time &time, const QString &str)	27
void parseValue (const QString &key, const QString &value).....	27
void parseIntegrationTime (const QString &value)	27
void parseDataGainValue (const QString &value)	28
void parseGPS (const QString &value).....	28
void parseGeoPoint (const QString &value).....	28
void parseAcceleration (const QString &value)	28
void parseGyroscope (const QString &value).....	28
void recalculateGeoPoints()	28
void reset ()	28
class Frame::Private : public QsharedData (<i>frame.cpp</i>)	29
void Frame::setOrientation(const Frame::Vector3D &orientation)	29
void Frame::setOrientation(const QString &orientation)	29
class EnviReader::Private : public QsharedData (<i>envireader.cpp</i>)	29
EnviReader& EnviReader::operator=(const EnviReader &other).....	30
bool EnviReader::setWavelength(qreal wl).....	30
QString getHeaderFileName() const	30
bool setWavelength(qreal wl)	30
void setWavelengthIndex(int wavelengthIndex)	30
int setDefaultWavelengthIndex()	31
HistogramData EnviReader::parseIntensityHistogram(HistogramBins bins) .	31

qreal getSpectrum (int sample, int line, int samples, int lines)	31
const EnviHeader EnviReader:: getHeader () const.....	32
void EnviReader:: setImageFile (const QString &imageFile, bool parse)	32
bool EnviReader:: isValid () const	32
void EnviReader:: swap (EnviReader &other)	33
void EnviReader:: reset ().....	33
EnviVisualizer::EnviVisualizer(QObject *parent)	33
EnviReader EnviVisualizer:: getReader (const QUrl &directory)	33
void EnviVisualizer:: activateWavelength ()	33
HistogramData EnviVisualizer:: getHistogram () const	34
QSize EnviVisualizer:: imageSize () const.....	34
QMap<qreal, qreal> EnviVisualizer:: getIntensities (int sample, int line, int samples, int lines) 34	
QList<qreal> EnviVisualizer:: getWavelengths ()	34
QList<int> EnviVisualizer:: getSequenceOrder ().....	34
QList<int> EnviVisualizer:: getOrder ()	34
QString EnviVisualizer:: getSequenceName ().....	34
void EnviVisualizer:: setActiveWavelength (qreal wl).....	34
void EnviVisualizer:: rgbImage (const QMap<qreal, QGSequence::FalseRgbMark> &map) ..	35
void EnviVisualizer:: setDirectory (const QUrl &directory)	35
static bool convertToGrayscale8 (uchar *ch, int row, int columns, const QByteArray &data) 35	
EnviWriter::EnviWriter(EnviStatusHandler *statusHandler)	35
EnviStatus EnviWriter:: open (const QString &baseName, const QDir &outputFolder, const QString description)	35
EnviStatus EnviWriter:: addFrame (const FrameSource &source).....	36

QString EnviWriter:: getHeaderFileName () const	36
QString EnviWriter:: getImageFileName () const	36
EnviStatus EnviWriter:: close (bool keepFiles)	36
class EnviWriter::Private	37
QString parentPath (QString path)	37
EnviStatus checkStorageAvailability (QString path)	37
EnviStatus openOutputImage (const QString &baseName, const QDir &outputFolder)	37
DefaultEnviStatusHandler::DefaultEnviStatusHandler(): EnviStatusHandler()	37
EnviStatus DefaultEnviStatusHandler:: raiseStatus (EnviStatus error, const QString &message) 38	
void DefaultEnviStatusHandler:: clearError ()	38
bool DefaultEnviStatusHandler:: error () const	38
QString DefaultEnviStatusHandler:: getErrorMessage ()	38
Ghost PC Interface API	39
MessageClient::MessageClient(int port, QObject *parent) : QObject(parent), m_connected(false), m_responsePending(false), k_magic("m4rugrmkcuiqotez")	40
bool MessageClient:: doConnect (QHostAddress ip)	40
bool MessageClient:: doPoll (QHostAddress ip)	40
bool MessageClient:: isConnected ()	40
void MessageClient:: setLogging (bool logging)	40
bool MessageClient:: writeData (QByteArray data)	41
void MessageClient:: disconnect ()	41
void MessageClient:: tick ()	41
void MessageClient:: processReceived ()	41
void MessageClient:: readyRead ()	41
QByteArray MessageClient:: sendMessageSync (QByteArray data, int timeout)	41

bool MessageClient::sendMessageAsync(QByteArray data)	41
FrameClient::FrameClient(QObject *parent) : QObject(parent), m_connected(false), m_size(0), m_type(0), k_magic("m4rugrmkcuiqotez")	42
bool FrameClient::doConnect(QHostAddress ip)	42
bool FrameClient::isConnected()	42
void FrameClient::disconnect()	42
void FrameClient::readyRead()	42

What data can we get from SENOP HSC

Intensity for every pixel 1024*1024 per captured wavelength.

Histogram for every wavelength.

Regular RGB images made by capturing 3 wavelengths.

SENOH HSC can capture whole VIS(400-765nm) and lower NIR(765-1000nm).

We can get the whole spectrum per pixel and can use this to identify and analyze the materials captured.

Capturing Sessions (Scripts and Sequences)

Hype captures sessions defined by scripts and sequences. Scripts are .JSON files that contain sequence objects. Running a **script** records one **session**. Script is a collection of sequences.

Script determines

- Measure Interval for sequences
- Measure occurrence

- Triggering/ Delay

Sequence determines

- Width/order of the capture
 - normal
 - narrow
 - wide
- Wavelengths to be captured
- Exposure time

```

{
  "Name": "test-script",
  "ScriptSegments": [
    {
      "MeasureInterval": "immediate",
      "MeasureOccurrences": 1,
      "RunTriggering": "delay",
      "RunTriggeringTime": 0,
      "SequenceGroup": [
        {
          "DefaultExposure": 1,
          "DefaultOrder": "normal",
          "Name": "500and600nm",
          "Steps": [
            {
              "VF": "X",
              "Wavelength": 500
            },
            {
              "Wavelength": 600
            }
          ]
        },
        {
          "DefaultExposure": 1,
          "DefaultOrder": "narrow",
          "Name": "700and800-narrow",
          "Steps": [
            {
              "VF": "X",
              "Wavelength": 700
            },
            {
              "Wavelength": 800
            }
          ]
        }
      ]
    }
  ]
}

```

Script

Script metadata

First sequence of the script

Second sequence of the script

Image 1 Generic Senop HSI Script

Image 1 shows a generic script in text editor.

Data difference in Hyperspectral camera and imported data

Data in Senop hyperspectral camera's SSD is saved as follows

- /media
 - frames
 - RecordingInfo.txt (metadata about recording)
 - <scriptname_timestamp_references>
 - reference.img
 - reference.png
 - reference.hdr
 - <scriptname_timestamp>
 - <sequence_name_00000>
 - measurement-wl000.img
 - measurement-wl000.png
 - measurement- wl000.hdr
 - measurement-wl001.img
 - measurement-wl001.png
 - measurement-wl001.hdr
 -
 - <sequence_name_00001>
 - measurement-wl000.img
 - measurement-wl000.png
 - measurement- wl000.hdr
 - measurement-wl001.img
 - measurement-wl001.png
 - measurement-wl001.hdr

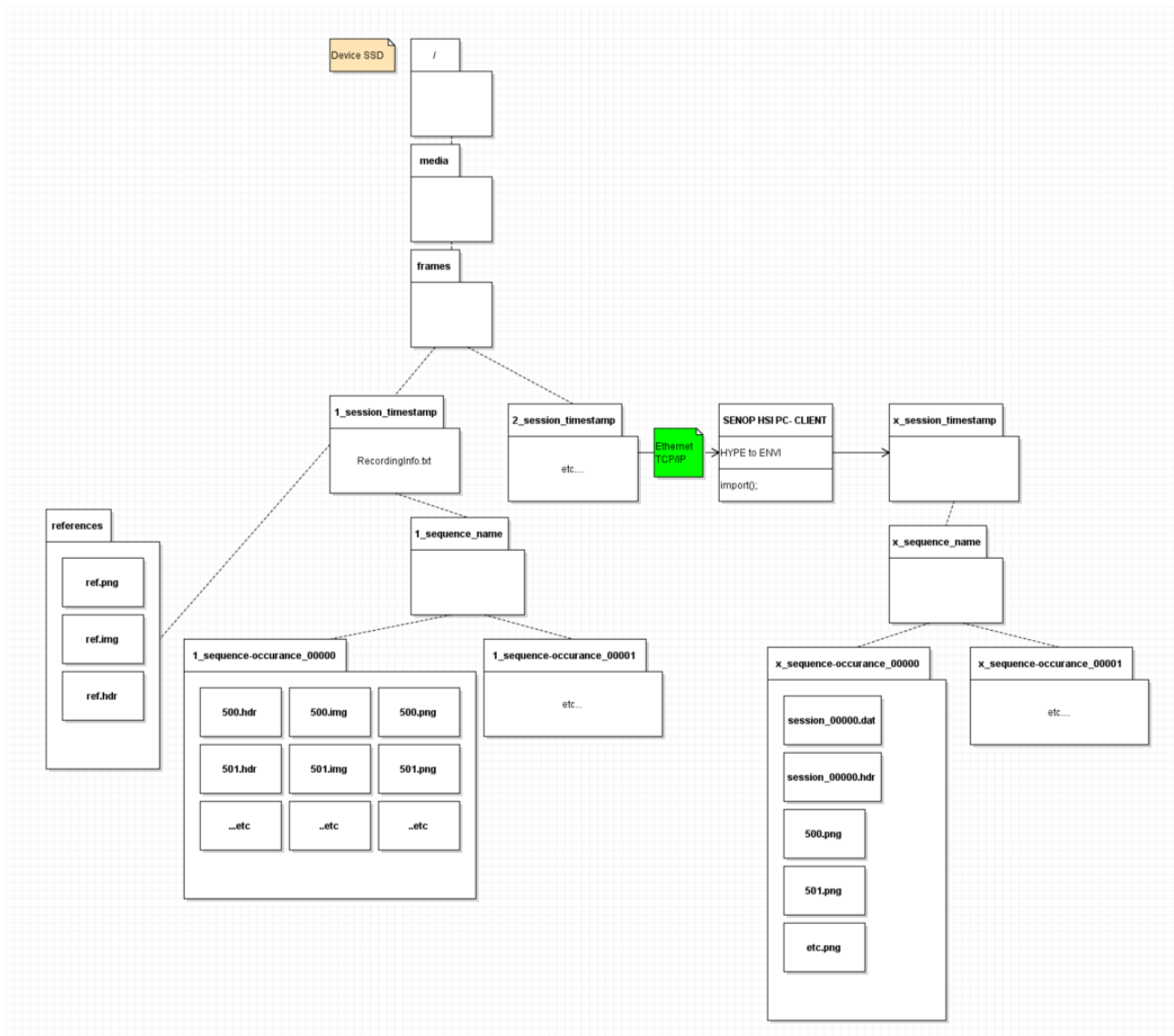


Image 2 Device to PC import

Files in camera

- **.png** the picture thumbnail
- **.img** is like ENVI data but fragmented in every measured wavelength
- **.hdr** header data for that wavelength (not ENVI standard)

SENOP- HSI Client uses this data and joins it to acceptable ENVI format for the PC Application.

Files after import



Image 3 Imported Session

- **.dat** file (ENVI data) raw binary data of the recording
- **.hdr** (ENVI header) meta data of the recording
- **.png** thumbnails for every wavelength.

Imported/ Translated data

dat- file

Contains the datacube. Flat binary stream of bytes. Saved in BSQ format. <https://www.harrisgeospatial.com/docs/enviimagefiles.html>

hdr- file

Contains information about the datacube.

Human readable txt- file.

Lines starting with “senop” are not of standard ENVI fields but are needed in SENOP HSI.

<https://www.harrisgeospatial.com/docs/ENVIHeaderFiles.html> describes the original ENVI headers.

Fields of the header

Description

Description of the file. "Senop HYPE2.0 Image" by default.

Wavelength units

Units used to represent wavelengths. Nanometers(nm) by default.

Samples

- The number of samples (pixels per image for each line)
 - Horizontal dimension pixels

Lines

- Number of lines per image for each band.
 - Vertical dimension pixels

File type

- "ENVI" by default

Format/Interleave

- **Band Sequential**
 - Used in SENOP HSI
 - BSQ format is the simplest format, where each line of the data is followed immediately by the next line in the same spectral band. This format is optimal for spatial (x,y) access of any part of a single spectral band. Visualization example in Image 4.

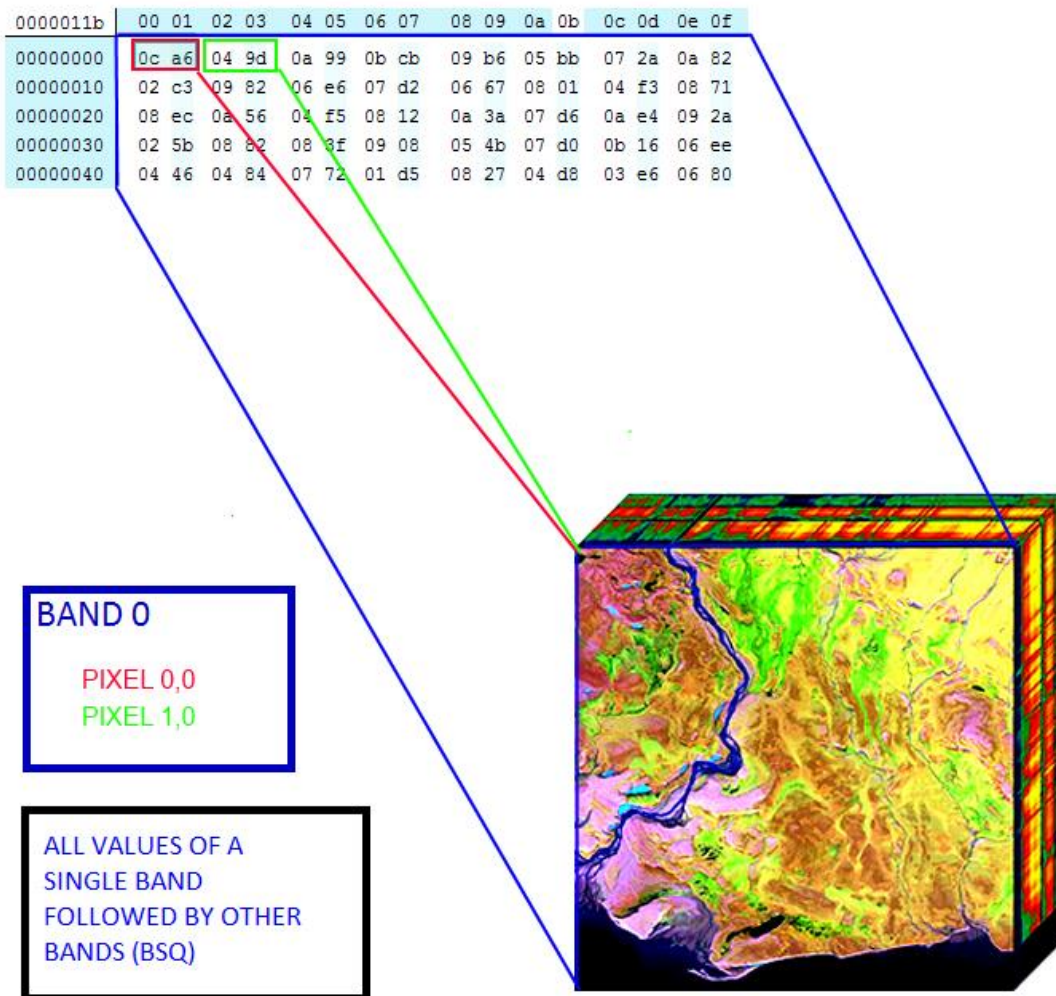


Image 4 BSQ data - Cube comparison

- Band-interleaved-by-pixel
- Band-interleaved-by-line

Byte order

- Byte order=1 (Network (IEEE) in the Header Info dialog) is most significant byte first (MSF) data (all other platforms).

Datatype

- The type of data representation:

- 1 = Byte: 8-bit unsigned integer
- 2 = Integer: 16-bit signed integer
- 3 = Long: 32-bit signed integer
- 4 = Floating-point: 32-bit single-precision
- 5 = Double-precision: 64-bit double-precision floating-point
- 6 = Complex: Real-imaginary pair of single-precision floating-point
- 9 = Double-precision complex: Real-imaginary pair of double precision floating-point
- **12 = Unsigned integer: 16-bit**
 - Used in SENOP HSI
 - 0x000 - 0xFFFF
- 13 = Unsigned long integer: 32-bit
- 14 = 64-bit long integer (signed)
- 15 = 64-bit unsigned long integer (unsigned)

Senop sequence name

- Name of the sequence used to make the capture.

Bands

- Number bands captured.

Acquisition time

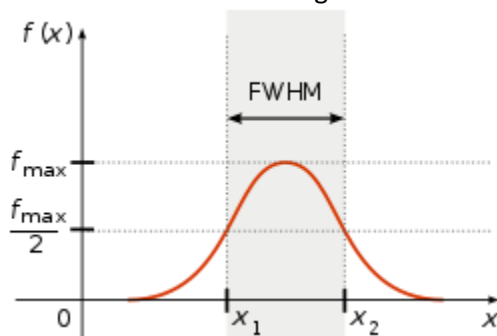
- Acquisition time of the sequence.

Wavelength

- List of wavelengths captured. Units nanometer(nm).

FWHM

- Lists full-width-half-maximum (FWHM) values of each band in an image. Units should be the same as those used for wavelength and set in the wavelength parameter. Units are in nanometers(nm).



- Value is taken from a calibration file in the camera.

data gain values

- Gain factor for each band
- Used to convert the datapoints to $W/(m^2 * \mu m * sr)$.
- Fixed with integration time.

Senop sequence order

- Indexing the “wavelength” field to the order that the bands were captured.

Senop order

- Order of the band
- Regards the position of the FPI

Senop timestamp

- Tells passed time from the start of the capture.
- nano seconds
- Indexing with the “wavelength” and “senop sequence order” fields.

Senop sensor

- Sensor used in the capture.
 - NIR
 - VIS
 - N/A
- To be deprecated

Senop frame counter

- Number/order of frame captured in the whole session.

Senop user counter

- To be deprecated
- Not used

Senop repeat

- To be deprecated
- Not used

Senop integration time

- List of exposure times for captured bands.
- Units milliseconds (ms)

Senop gps

- List of geolocations of the captured bands.

Senop acceleration

- List of x,y,z axis acceleration values.

Senop gyroscope

- List of x,y,z axis gyroscope position values.

Class and file description of SENOP HSI

This chapter describes the communication and data processing classes and functions in SENOP HSI Application.

```
class EnviHeader::Private : public QSharedData (enviheader.cpp)
```

The **EnviHeader** class is used to read, parse and store the ENVI header file data. It has helper functions for parsing data from strings and convert them to correct data types and lists. **EnviHeader** has a list of member functions to help modify and access the header information.

EnviHeader& EnviHeader::operator=(const EnviHeader &other)

- Overloads EnviHeader's "=" operator so EnviHeader can be assigned again.

`bool EnviHeader::parse(const QString &headerFile)`

- **Parameters:**
 - `const QString &headerFile` the absolute path of the .hdr file
- Uses the `bool parse(const QString &headerFile)` function.
 - Parses through the ENVI .hdr file line by line and sends each line to void `parse-Value(const QString &key, const QString &value)` for further parsing.
- **Returns:** true if the headers syntax was correct and the parsing was successful.

`const QString &EnviHeader::getDescription() const`

- **Returns:** the headers description `QString m_description`.

`const QString &EnviHeader::getSequenceName() const`

- **Returns:** the headers sequence name `QString m_sequenceName`.

`const QString &EnviHeader::getWavelengthUnits() const`

- **Returns:** the headers wavelength units `QString m_wavelengthUnits`.

`const EnviHeader::Time &EnviHeader::getAcquisitionTime() const`

- **Returns:** the headers acquisition time `EnviHeader::Time m_acquisitionTime`.

`const QList<qreal> &EnviHeader::getWavelength() const`

- **Returns:** list of the wavelengths used in the measurement defined in the header `QList<qreal> m_wavelength`

`int EnviHeader::getSamples() const`

- **Returns:** the number of samples, per image for each band `int m_samples`.

`int EnviHeader::getLines() const`

- **Returns:** the number of lines per image for each band.

`const QString &EnviHeader::getFileType() const`

- **Returns:** the file type defined in the header `QString m_fileType`.

const EnviHeader::InterleaveType &EnviHeader::getInterleave() const

- **Returns:** the interleave type defined in the header EnviHeader::InterleaveType m_interleave.
- See section .hdr Format/Interleave for more on interleave types.

int EnviHeader::getByteOrder() const

- **Returns:** the byte order defined in the header int m_byteOrder.
- See section .hdr byte order for more on byte order.

int EnviHeader::getDataType() const

- **Returns:** the data type defined in the header int m_dataType.
- See section .hdr data type for more on data types.

int EnviHeader::getBands() const

- **Returns:** the number of bands defined in the header int m_bands.

const QList<Frame::ESensor> &EnviHeader::getSensor() const

- **Returns:** list of sensors used in the measurements defined in the header QList<Frame::ESensor> m_sensor.
- Possible sensors

```
enum class ESensor
{
    NIR = 0,
    VIS = 1,
    LastValid = 1,
    Unknown = 0xFF
} quint8;
```

const QList<int> &EnviHeader::getFrameCounter() const

- **Returns:** list of the senop frame counters defined in the header QList<int> m_frameCounter.

const QList<qreal> &EnviHeader::getFWHM() const

- **Returns:** list of the fwhm (full-width-half-maximum) values defined in the header QList<qreal> m_fwhm.

const QList<int> &EnviHeader::getSequenceOrder() const

- **Returns:** list of the sequence order defined in the header QList<int> m_sequenceOrder.

`int EnviHeader::getSequenceOrder(int index) const`

- **Parameters:**
 - `int index` the index of the value `QList<int> m_sequenceOrder` that will be returned.
- **Returns:** the value from `QList<int> m_sequenceOrder` defined by the index.

`const QList<int> &EnviHeader::getOrder() const`

- **Returns:** `const QList<int>` the list of the senop orders defined in the header `QList<int> m_order`.

`int EnviHeader::getOrder(int index) const`

- **Parameters:**
 - `int index` the index of the value `QList<int> m_order` that will be returned.
- **Returns:** the `int` value from the list `QList<int> m_order` defined by the index.

`QString EnviHeader::getOrderAsString(int index)`

- **Parameters:**
 - `int index` the index of the value in the `QList<int> m_order` that will be returned as `QString`.
- **Returns:**
 - `QString("narrow")` if order is 0
 - `QString("wide")` if order is 2
 - `QString("normal")` if order is something else

`const QList<int> &EnviHeader::getUserCounter() const`

- **Returns:** The list of senop user counter values defined in the head `QList<int> m_userCounter`.

`const QList<int> &EnviHeader::getRepeat() const`

- **Returns:** The list of senop repeat values defined in the header `QList<int> m_repeat`.

`const QList<qint64> &EnviHeader::getTimeStamp() const`

- **Returns:** The list of senop timestamp values defined in the header `QList<qint64> m_timeStamp`.

`const QList<qreal> &EnviHeader::getIntegrationTime() const`

- **Returns:** The list of senop integration time values defined in the header `QList<qreal> m_integrationTime`.

`const QList<qreal> &EnviHeader::getDataGainValue() const`

- **Returns:** The list of data gain values defined in the header `QList<qreal> m_dataGainValue`.

`const QList<Frame::GPSFormat> &EnviHeader::getGPSFormat() const`

- **Returns:** The gps format used in senop gps values in the header `Frame::GPSFormat`
- Possible formats

```
enum class GPSFormat
{
    RMC = 0,
    Unknown = INT_MAX
};
```

`const QList<QString> &EnviHeader::getGPS() const`

- **Returns:** List of the senop gps values (measurement locations) defined in the header `QList<QString> m_gps`;

`QGeoPositionInfo EnviHeader::getGeoPosition(int index) const`

- **Parameters:**
 - `int index` index of the value in the `QList<Frame::GPSFormat> m_gpsFormat` and `QList<QString> m_gps` lists that will be used to generate a `QGeoPositionInfo` with `QGeoPositionInfo Frame::getGeoPosition(Frame::GPSFormat format, const QString &value)`
- **Returns:** `QGeoPositionInfo` that was generated.

`const QList<qreal> &EnviHeader::getGeoPoint(bool recalculate) const`

- **Parameters:** `bool recalculate`
 - **true** if you want to recalculate the geoposition
 - **false** if you dont want to recalculate
- See section `void recalculateGeoPoints()`
- See section `.hdr geo points` for more on geo points
- **Returns:** List of geo points `QList<qreal> m_geoPoint`

`const QList<QString> &EnviHeader::getAcceleration() const`

- **Returns:** List of senop acceleration values defined in the header `QList<QString> m_acceleration`.

`Frame::Vector3D EnviHeader::getAcceleration(int index)`

- **Parameters:**
 - The index of the value in the list `QList<QString> m_acceleration` that will be used by `Frame::Vector3D` `Frame::getAcceleration(const QString &value)` to return a specific `Frame::Vector3D`
- **Returns:** `Frame::Vector3D` specified by the index.

`const QList<QString> &EnviHeader::getGyroscope() const`

- **Returns:** List of the senop gyroscope values `QList<QString> m_gyroscope` defined in the header.

`Frame::Vector3D EnviHeader::getGyroscope(int index)`

- **Parameters:**
 - `int index` index of a specific senop gyroscope value in the `QList<QString> m_gyroscope`.
- **Returns:** `Frame::Vector3D` generated by `Frame::Vector3D` `Frame::getOrientation(const QString &value)` with the specific senop gyroscope value.

`void EnviHeader::reset()`

- Resets the header object with void **reset()**.

`bool parse(const QString &headerFile)`

- Parses through the ENVI .hdr file line by line and sends each line to void **parseValue**(const `QString &key`, const `QString &value`) for further parsing.
- **Returns** true if the header is valid and parsing was succesfull, otherwise returns false.

`void parseDescription(const QString &value)`

- Saves headers description to `QString m_description`

void **parseSequenceName**(const QString &value)

- Saves headers sequence name to QString m_sequenceName

void **parseWavelengthUnits**(const QString &value)

- Saves headers wavelength unit values in lower case to QString m_wavelengthUnits

void **parseAcquisitionTime**(const QString &value)

- Uses function void **parseISO8601**(EnviHeader::Time &time, const QString &str) to parse the acquisition time to ISO8601 date – time standard. Value is saved to EnviHeader::Time m_acquisitionTime.

```
struct Time
{
    QDateTime m_time;
    quint32 m_nanoSeconds;
};
```

void **parseWavelength**(const QString &value)

- Uses function bool **parseDoubleList**(QList<qreal> &list, const QString &value) to check that the wavelength values in the header are valid and saves the values to QList<qreal> m_wavelength
- Saves the number of wavelengths (bands) to int m_bands.

void **parseSamples**(const QString &value)

- Checks if QString &value (samples value from header) can be converted to int. If it can saves number of samples to int m_samples. Else gives it value -1.

void **parseLines**(const QString &value)

- Checks if QString &value (lines value from header) can be converted to int. If it can saves number of samples to int m_lines. Else gives it value -1.

void **parseFileType**(const QString &value)

- Checks if the file type is "ENVI" if it is saves that value to QString m_fileType. Otherwise the values is left empty "".

void **parseInterleave**(const QString &value)

- Checks the headers interleave value and saves it to EnviHeader::InterleaveType m_interleave.
- Possible types
 - EnviHeader::InterleaveType::BSQ (Used in SENOP HSI)
 - EnviHeader::InterleaveType::BIL
 - EnviHeader::InterleaveType::BIP
 - EnviHeader::InterleaveType::Unknown

void **parseBands**(const QString &value)

- Checks if the QString &value, bands value from header, can be converted to int. If can the values is saved to int m_bands. Else it saves value -1.

void **parseSequenceOrder**(const QString &value)

- Checks if the sequence order values from header QString &value can be converted to integers and saves them to QList<int> m_sequenceOrder.

void **parseOrder**(const QString &value)

- Checks if the order values from header can be converted to int with void **parseIntList**(QList<int> &list, const QString &value) and saves them to QList<int> m_order;
- m_order[i] is finally determined by static inline int Frame::checkOrder(int order)

void **parseSensor**(const QString &value)

- Checks the QString &value, sensor used in measurement and appends the value to QList<Frame::ESensor> m_sensor.
- Possible sensor types
 - Frame::ESensor::NIR
 - Frame::ESensor::VIS
 - Frame::ESensor::Unknown
- The m_sensor.length() == m_bands. Meaning that the used sensor is determined for every wavelength.

void **parseByteOrder**(const QString &value)

- Checks if const QString &value, byte order from header, can be converted to int. If it can saves it to m_byteOrder. Else it saves value -1.
- See section .hdr byte order for more on byte order.

void **parseDataType**(const QString &value)

- Checks if const QString &value, data type value from header, is valid data type and can be converted to int. If it can the value is saved to int m_dataType.
- See section .hdr data type for more on data types.

void **parseFrameCounter**(const QString &value)

- Uses void **parseIntList**(QList<int> &list, const QString &value) for checking that the senop frame counter values from header are valid and saves them to QList<int> m_frameCounter.

void **parseFWHM**(const QString &value)

- Uses void bool **parseDoubleList**(QList<qreal> &list, const QString &value) for checking that fwhm values const QString &value in the header are valid and can be converted to double and saves the values to QList<qreal> m_fwhm.
- If the header formatting is invalid or the value can't be converted to double returns false.

void **parseUserCounter**(const QString &value)

- Uses void **parseIntList**(QList<int> &list, const QString &value) for checking that senop user counter values const QString &value from header are valid and can be converted to int and saves the values to QList<int> m_userCounter.

void **parseRepeat**(const QString &value)

- Uses void **parseIntList**(QList<int> &list, const QString &value) for checking that senop repeat values const QString &value from header are valid and can be converted to int and saves the values to QList<int> m_repeat.

bool **parseDoubleList**(QList<qreal> &list, const QString &value)

- Helper function that parses the const QString &value parameter with "," separator and saves the double values to provided pointer parameter QList<qreal> &list. If successful returns true.
- If the header formatting is invalid or the conversion to double can't be done returns false.

void **parseIntList**(QList<int> &list, const QString &value)

- Helper function that parses the const QString &value parameter with "," separator and saves the int values to provided pointer parameter QList<int> &list.

void **parseStringList**(QList<QString> &list, const QString &value)

- Helper function that parses const QString &value parameter with "," separator and seves the QString values to provided pointer parameter QList<QString> &list.

void **parseStringListWithQuotes**(QList<QString> &list, const QString &value)

- Helper function that parses const QString &value parameter. Qstrings parsed by this function have "," separators in them. Function ignores "," separator when it is inside quote pairs.
- Example:
- "\$GPRMC,064216,A,6215.4701,N,02608.6898,E,1.0419,99.709,140119,,*14","\$GPRMC,064216,A,6215.4701,N,02608.6898,E,1.0419,99.709,140119,,*14"
- Gets parsed to:
 - "\$GPRMC,064216,A,6215.4701,N,02608.6898,E,1.0419,99.709,140119,,*14"
 - "\$GPRMC,064216,A,6215.4701,N,02608.6898,E,1.0419,99.709,140119,,*14"

void **parseTimestamp**(const QString &value)

- Parses const QString &value, senop time stamp values, from header and checks that they can be converted to longlong. Saves values to QList<qint64> m_timeStamp.

void **parseISO8601**(EnviHeader::Time &time, const QString &str)

- Parses const QString &str and checks if it can be a valid Qt::ISODate. If valid saves it to

```
struct Time
{
    QDateTime m_time;
    qint32 m_nanoSeconds;
};
```

void **parseValue**(const QString &key, const QString &value)

- Function used to determine what line from the header is being parsed and what parser function it needs.
- **Parameters**
 - **const QString &key** keyword in the header file that specifies what parser is used for the following string const QString &value
 - **const QString &value** string after the keyword in header file specifying the values.
- static QMap<QString, ParseFunc> s_map then directs the value to the correct function.

void **parseIntegrationTime**(const QString &value)

- Uses function void **parseUserCounter**(const QString &value) to parse senop integration time values from the header.
- Saves the values to QList<int> m_userCounter.

void **parseDataGainValue**(const QString &value)

- Uses function bool **parseDoubleList**(QList<qreal> &list, const QString &value) to parse data gain values from the header and saves them to QList<qreal> m_dataGainValue.

void **parseGPS**(const QString &value)

- Uses function void **parseStringListWithQuotes**(QList<QString> &list, const QString &value) to parse senop gps values from the header.
- Saves the values to QList<QString> m_gps.
- Determines the GPS- format and saves it to QList<QString> m_gps.
 - Possible formats
 - Frame::GPSFormat::RMC
 - Frame::GPSFormat::Unknown

void **parseGeoPoint**(const QString &value)

- Uses function bool **parseDoubleList**(QList<qreal> &list, const QString &value) to parse geo points values from the header.
- Saves the values to QList<qreal> m_geoPoint.

void **parseAcceleration**(const QString &value)

- Uses function void **parseStringListWithQuotes**(QList<QString> &list, const QString &value) to parse senop acceleration values from the header.
- Saves the values to QList<QString> m_acceleration.

void **parseGyroscope**(const QString &value)

- Uses function void **parseStringListWithQuotes**(QList<QString> &list, const QString &value) to parse senop gyroscope values from the header.
- Saves the values to QList<QString> m_gyroscope.

void **recalculateGeoPoints**()

- Recalculates the geo points QList<qreal> m_geoPoint to the ENVI format
- Uses QGeoPositionInfo Frame::**getGeoPosition**(Frame::GPSFormat format, const QString &value) to get new coordinates and sets them to pixel 1,1.
- See section .hdr geo points for more on geo points.

void **reset**()

- Resets the EnviHeader member values to these values.

```
m_description = "";
m_sequenceName = "";
m_wavelengthUnits = "";
```

```

m_acquisitionTime.m_time = QDateTime();
m_acquisitionTime.m_nanoseconds = 0;
m_wavelength.clear();
m_samples = -1;
m_lines = -1;
m_fileType = "ENVI";
m_interleave = EnviHeader::InterleaveType::Unknown;
m_byteOrder = -1;
m_dataType = -1;
m_bands = -1;
m_sequenceOrder.clear();
m_order.clear();
m_sensor.clear();
m_frameCounter.clear();
m_fwhm.clear();
m_userCounter.clear();
m_timeStamp.clear();
m_integrationTime.clear();
m_dataGainValue.clear();
m_gpsFormat.clear();
m_gps.clear();
m_geoPoint.clear();
m_acceleration.clear();
m_gyroscope.clear();

```

class Frame::Private : public QsharedData (*frame.cpp*)

void Frame::setOrientation(const Frame::Vector3D &orientation)

- **Parameters:**
 - const Frame::Vector3D &orientation to be set in Frame::Vector3D m_orientation.

void Frame::setOrientation(const QString &orientation)

- **Parameters:**
 - const QString &orientation value of the orientation to be set in Frame::Vector3D m_orientation in QString format.

class EnviReader::Private : public QsharedData (*envireader.cpp*)

EnviReader class is used to access, read and parse the .dat- file. EnviReader contains all necessary functions to use the .dat when its a spesific type. But lacks support for other formats at this point.

Supported type currently is:

```
(m_header.getDataType() == 12);
(m_header.getFileType() == "ENVI");
(m_header.getByteOrder() == 1);
(m_header.getInterleave() == EnviHeader::InterleaveType::BSQ);
```

EnviReader has functions for doing intensity calculations from the image data.

EnviReader& EnviReader::operator=(const EnviReader &other)

- Overloads the EnviReaders "=" operator so copies can be done correctly.

bool EnviReader::setWavelength(qreal wl)

- **Parameters:**
 - qreal wl selected wavelength.
- Sets the int m_wavelengthIndex to the selected wavelengths index.

QString getHeaderFileName() const

- **Returns:** the absolute path and name of the header file QFile m_file as QString.

bool setWavelength(qreal wl)

- **Parameter:** qreal wl wavelength to be set under observation.
- Iterates over QList<qreal> m_wavelength until it finds reletively close match between parameter wl and actual value in the list and sets m_wavelengthIndex to that wavelengths index.
- **Returns:**
 - **true** when the value is found
 - **false** if inline bool **parseHeader()** fails

void setWavelengthIndex(int wavelengthIndex)

- **Parameters:**
 - int wavelengthIndex index to be set in int m_wavelengthIndex.

int setDefaultWavelengthIndex()

- Sets int m_wavelengthIndex to 0.

HistogramData EnviReader::parseIntensityHistogram(HistogramBins bins)

- **Parameters:**
 - HistogramBins bins
 - Bins256 Used in Senop HSI
 - Possible HistogramBins

```
enum class HistogramBins
{
    Bins1 = 1,
    Bins2 = 2,
    Bins4 = 4,
    Bins8 = 8,
    Bins16 = 16,
    Bins32 = 32,
    Bins64 = 64,
    Bins128 = 128,
    Bins256 = 256,
    Bins512 = 512,
    Bins1024 = 1024,
    Bins2048 = 2048,
    Bins4096 = 4096
};
```

- **Returns:** parsed HistogramData m_histogramData

qreal getSpectrum(int sample, int line, int samples, int lines)

- **Parameters:**
 - int sample x- location of the pixel under study.
 - int line y- location of the pixel under study.
 - int samples dimension of pixels in x- axis to calculate the average intensity.
 - int lines dimension of pixels in y- axis to calculate the average intensity.

<div> <div>x</div> <div>sample</div> </div> <div> <div>y</div> <div>line</div> </div>	0	1	2	3	4	5	6	7	8
0									
1									

2									
3									
4									
5									
6									
7									

- Example with **getSpectrum(0, 3, 4, 4)** it calculates the average intensity of the nearby pixel like shown in the table. Red being the selected pixel and blue being the nearby pixels taken in to calculate the average.
- **Returns:** qreal spectrum that is calculated average intensity of the nearby are of the selected pixel defined by the parameters.

`const EnviHeader EnviReader::getHeader() const`

- **Returns:** the ENVI header EnviHeader m_header.

`void EnviReader::setImageFile(const QString &imageFile, bool parse)`

- **Parameters:**
 - `const QString &imageFile` name or path of the file
 - `bool parse`
 - **true** if you want the header to be parsed right after.
 - **false** if parsing is not wanted.
 - Sets new name `const QString &imageFile` for `QFile m_file`.

`QString EnviReader::getImageFile() const`

- **Returns:** `QString` file name of `QFile m_file`.

`QByteArray EnviReader::getImageData() const`

- **Returns:** Data of the .dat file `QByteArray m_imageData`

`bool EnviReader::isValid() const`

- Checks that the `EnviReader` is valid by trying to parse the header with inline `bool parseHeader()`
- **Returns:**
 - **true** if the header was valid

- **false** if the header was invalid

void EnviReader::swap(EnviReader &other)

- **Partameters:** EnviReader &other the EnviReader to be swapped to.
- Swap this instance's explicitly shared data pointer with the explicitly shared data pointer in *other*. (Qt Documentation - QExplicitlySharedDataPointer Class).

void EnviReader::reset()

- Resets the EnviReader object.
 - Clears filename
 - Clears header
 - Clears datamask
 - Resets wavelength

EnviVisualizer::EnviVisualizer(QObject *parent)

EnviVisualizer class is used to make visualizations from the ENVI data like the intensity diagram and histogram. It has helper functions to convert different image formats.

EnviReader EnviVisualizer::getReader(const QUrl &directory)

- **Parameters:** const QUrl &directory absolute path to the directory containing ENVI files.
- **Returns:** EnviReader constructed and defined by the selected directory.

void EnviVisualizer::activateWavelength()

- Activates the active wavelength defined by EnviReader
- Constructs a QImage from the QByteArray imageData and converts it to QImage::Format_Grayscale8
- Constructs a new histogram visualization
- Emits signals to put the new visualizations on display to the UI.

HistogramData EnviVisualizer::*getHistogram()* const

- **Returns:** HistogramData m_histogramData

QSize EnviVisualizer::*imageSize()* const

- Asks EnviHeader for the images dimensions and returns them.
- **Returns:** QSize with images dimensions.

QMap<qreal, qreal> EnviVisualizer::*getIntensities*(int sample, int line, int samples, int lines)

- **Parameters**
 - int sample x- location of the pixel under study.
 - int line y- location of the pixel under study.
 - int samples number of horizontal pixels taken for average.
 - int lines number of vertical pixels taken for average.
- Iterates function qreal EnviReader::**getSpectrum**(int sample, int line, int samples, int lines) const for every wavelength measured.
- **Returns:** QMap<qreal, qreal> of the wavelengths and intensities where <wavelength, intensity>

QList<qreal> EnviVisualizer::*getWavelengths()*

- **Returns:** QList<qreal> of measured wavelengths.

QList<int> EnviVisualizer::*getSequenceOrder()*

- **Returns:** QList<int> of senop sequence order values.

QList<int> EnviVisualizer::*getOrder()*

- **Returns:** QList<int> of senop order values.

QString EnviVisualizer::*getSequenceName()*

- **Returns:** QString senop sequence name value.

void EnviVisualizer::*setActiveWavelength*(qreal wl)

- **Parameters:**
 - qreal wl wavelength to be activated
- Sets qreal wl to EnviReader m_reader as the active wavelength.
- Uses void EnviVisualizer::**activateWavelength**()

void EnviVisualizer::rgbImage(const QMap<qreal, QGSequence::FalseRgbMark> &map)

- **Parameters:**
 - const QMap<qreal, QGSequence::FalseRgbMark> &map QMap used to determine if the used sequence actually has valid wavelengths for RGB.
- Reads image data and sends it to static bool **convertToRgb**(uchar *ch, int row, int columns, const QByteArray &r, const QByteArray &g, const QByteArray &b) to be converted
- Emits imageChanged with the new RGB- image.

void EnviVisualizer::setDirectory(const QUrl &directory)

- **Parameters:**
 - const QUrl &directory is used for constructing EnviReader m_reader with ther function EnviReader EnviVisualizer::getReader(const QUrl &directory).
- Sets and activates m_reader 's default wavelength.

static bool convertToGrayscale8(uchar *ch, int row, int columns, const QByteArray &data)

- **Parameters:**
 - uchar *ch single line of data to be converted.
 - int row row number to be converted.
 - int columns number of column data to be converted.
 - const QByteArray &data ENVI data stream.
- Converts ENVI data to gray scale image
- **Returns:**
 - bool
 - **true** if conversion was succesful.
 - **false** if conversion fails.

EnviWriter::EnviWriter(EnviStatusHandler *statusHandler)

EnviWriter open, close and write ENVI files.

EnviStatus EnviWriter::open(const QString &baseName, const QDir &outputFolder, const QString description)

- **Parameters:**
 - const QString &baseName name of the aquired snapshot
 - const QDir &outputFolder path of the aquired snapshot
 - const QString description of the aquired snapshot

- Checks if the acquired snapshot can be opened on the client
 - **Returns:**
 - EnviStatus
 - EnviStatus::Ok if snapshot can be opened
 - Descriptive EnviStatus if something prevents opening the snapshot

EnviStatus EnviWriter::addFrame(const FrameSource &source)

- **Parameters**
 - const FrameSource &source raw imagedata
- Converts single frame of HYPE to ENVI data.
- **Returns:**
 - EnviStatus
 - Descriptive EnviStatus

QString EnviWriter::getHeaderFileName() const

- **Returns:**
 - QString name of the ENVI header file.
 - QString empty string if name cannot be acquired.

QString EnviWriter::getImageFileName() const

- **Returns:**
 - QString name of the ENVI data file.
 - QString empty string if name cannot be acquired.

EnviStatus EnviWriter::close(bool keepFiles)

- **Parameters:**
 - bool keepFiles determine if files should be kept.
- Closes the current snapshot so new ones can be opened.
- **Returns:**
 - EnviStatus
 - Descriptive EnviStatus

class EnviWriter::Private

EnviWriter has many private helper functions that are used by its public functions.

QString **parentPath**(QString path)

- **Parameters:**
 - QString path url to chop
- Removes the last folder from the string

EnviStatus **checkStorageAvailability**(QString path)

- **Parameters:**
 - QString path palce to save temporary snapsnhot
- Checks if path is a valid directory path to make folder for temporary snapshot data.
- **Returns:**
 - EnviStatus
 - EnviStatus::Ok valid
 - EnviStatus::InvalidOutputFolder invalid

EnviStatus **openOutputImage**(const QString &baseName, const QDir &outputFolder)

- **Paramaters:**
 - const QString &baseName name of the snapshot
 - const QDir &outputFolder path of the snapshot
- Checks if aquired snapshot can be opened
- **Returns:**
 - EnviStatus
 - EnviStatus::Ok file can be opened
 - EnviStatus::CannotCreatelImage cannot open snapshot

DefaultEnviStatusHandler::DefaultEnviStatusHandler(): EnviStatusHandler()

Hold ENVI status and forward error messages.

Possible ENVI status

```
enum class EnviStatus
{
    Ok,
    NothingToDo,
    InvalidOutputFolder,
    CannotCreateHeader,
    CannotCreateImage,
    ReadOnlyFileSystem,
    FileSystemNotAvailable,
    HeaderFileWriteError,
    ImageFileWriteError,
    DeviceDisconnected,
    DeviceMemoryFull,
    DeviceGenericError
};
```

EnviStatus DefaultEnviStatusHandler::raiseStatus(EnviStatus error, const QString &message)

- **Parameters:**
 - EnviStatus error state to put EnviStatusHandler to.
 - const QString &message error message to forward to EnviStatusHandler.
- Keeps track of important system status regarding device and ENVI data handling.
- **Returns:** current EnviStatus.

void DefaultEnviStatusHandler::clearError()

- Clears ENVI error message QString m_message
- Sets EnviStatus m_status to EnviStatus::Ok

bool DefaultEnviStatusHandler::error() const

- **Returns:**
 - **True** if EnviStatus::Ok.
 - **False** if EnviStatus is on error state.

QString DefaultEnviStatusHandler::getErrorMessage()

- Clears error message
- **Returns:**
 - QString the error message

Ghost PC Interface API

GhostPCInterface.pdf documentation specifies the functionality of GhostInterface class and messages that can be used to control the device.

QGhostInterface class is described in the linked documentation. This chapter describes the different message clients that the QGhostInterface uses.

QGhostInterface is a communication interface that is used to communicate with the camera.

Ghost interface has three ways to communicate.

- **Synchronous** Message Client MessageClient
 - Sending and receiving data with device synchronously.
- **Asynchronous** Message Client MessageClient
 - Sending and receiving data with device asynchronously.
- **Frame** Client FrameClient
 - Receiving captured frames from the device.

All communication types use the TCP/IP protocol and are available when the device is connected to a controlling device ex. PC with SENOP HSI through an ethernet cable. All message clients are owned and used by QGhostInterface.

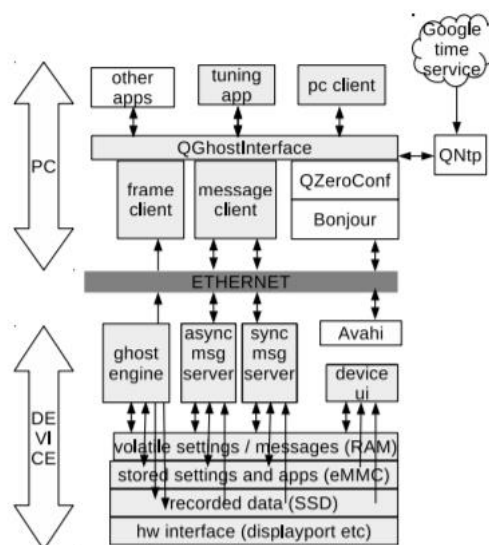


Image 5 Ghost Interface

```
MessageClient::MessageClient(int port, QObject *parent) :
QObject(parent), m_connected(false), m_responsePending(false),
k_magic("m4rugrmkcuiqotez")
```

bool MessageClient::doConnect(QHostAddress ip)

- **Parameters**
 - QHostAddress ip IP- address of the to be connected device.
- Configures the TCP-IP socket options and makes the connection.
- **Returns:**
 - bool
 - **true** if connection was made
 - **false** if could not connect.

bool MessageClient::doPoll(QHostAddress ip)

- **Parameters:**
 - QHostAddress ip IP- address of the to be polled device.
- Polls the connection if the connection takes too long.
- **Returns:**
 - bool
 - **true** if connection was made.
 - **false** if could not connect.

bool MessageClient::isConnected()

- **Returns:** bool
 - **true** if the MessageClient is connected.
 - **false** if MessageClient is not connected.

void MessageClient::setLogging(bool logging)

- **Parameters:** bool logging state to set bool m_logging
- Turn logging ON /OFF

bool MessageClient::writeData(QByteArray data)

- **Parameters:**
 - QByteArray data to be written on the socket.
- Writes data bytes on the socket
- **Returns:**
 - bool
 - **true** if at least one byte was written.
 - **false** if 0 bytes were written and time out was emitted

void MessageClient::disconnect()

- Disconnects the socket and clears data buffers.

void MessageClient::tick()

- Disconnects if time out is reached.
- If time out is not reached yet writes tick character ~ on the socket.

void MessageClient::processReceived()

- Validity check for messages.

void MessageClient::readyRead()

- Intermediary for QIODevice readyRead() signal and MessageClient processReceived() function.

QByteArray MessageClient::sendMessageSync(QByteArray data, int timeout)

- **Parameters:**
 - QByteArray data data to send.
 - int timeout = 5000 timeout in milliseconds .
- Sends message to the device synchronously.
- **Returns:**
 - QByteArray response to the message

bool MessageClient::sendMessageAsync(QByteArray data)

- **Parameters:**
 - QByteArray data message to send.
- Sends message to the device asynchronously.
- **Returns:**
 - bool
 - **true** if data was written to the socket.

- **false** if writing the data to socket failed.

```
FrameClient::FrameClient(QObject *parent) : QObject(parent),
m_connected(false), m_size(0), m_type(0), k_magic("m4rugrmkcuiqotez")
```

bool FrameClient::doConnect(QHostAddress ip)

- **Parameters**
 - QHostAddress ip IP- address of the to be connected device.
- Configures the TCP-IP socket options and makes the connection.
- **Returns:**
 - bool
 - **true** if connection was made
 - **false** if could not connect.

bool FrameClient::isConnected()

- **Returns:** bool
 - **true** if the FrameClient is connected.
 - **false** if FrameClient is not connected.

void FrameClient::disConnect()

- Disconnects the socket and clears data buffers.

void FrameClient::readyRead()

- Reads data from the socket and checks if received data is valid. If it is emits dataReceived.

