Hannu-Pekka Shemeikka

ULTRAÄÄNIKAMERAN PROTOTYYPIN KÄYTTÖSOVELLUS PC-YMPÄRISTÖSSÄ

ULTRAÄÄNIKAMERAN PROTOTYYPIN KÄYTTÖSOVELLUS PC-YMPÄRISTÖSSÄ

Hannu-Pekka Shemeikka

Opinnäytetyö

Syksy 2010

Tietotekniikka

Oulun seudun ammattikorkeakoulu

ALKULAUSE

Tämä opinnäytetyö on tehty pääosin 1.1.2010 - 15.9.2010 välisenä aikana Simsonar Oy:lle.

Työn ohjaavana opettajana on toiminut Esko Harvala Oulun seudun ammattikorkeakoulusta ja työn tilaajan puolelta valvojana on toiminut Pertti Seppänen.

Haluaisin kiittää erityisesti Pertti Seppästä avusta ja tuesta tämän opinnäytetyön teossa.

Oulussa 14.9.2010

Hannu Shemeikka

TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, Tietoturva

Tekijä: Hannu-Pekka Shemeikka
Opinnäytetyön nimi: Ultraäänikameran prototyypin käyttösovellus PC-ympäristössä
Työn ohjaaja: Esko Harvala
Työn valmistumislukukausi ja -vuosi: Syksy 2010 Sivumäärä: 40 + 5

TIIVISTELMÄ

Tämän opinnäytetyön tilaajana on Simsonar Oy, joka kehittää ultraäänikameran prototyyppiä vedenalaiseen kuvantamiseen. Tässä työssä on tarkoitus toteuttaa kyseiselle kameralle käyttösovellus PC-ympäristöön. Tavoitteena on saada aikaan toimiva sovellus, joka muodostaa kameran tuottamasta kaikuinformaatiosta liikkuvaa kuvaa tietokoneen näytölle ja pystyy myös tallentamaan tätä muodostettua kuvaa sekä toistamaan tallennettua kuvaa.

Ultraäänikameran prototyyppi perustuu USRP-kortille rakennettaviin tytärkortteihin. USRP-kortin kanssa kommunikoimiseen käytetään Gnu Radio:ta. Se on avoimen lähdekoodin ohjelmistokehityspaketti ohjelmistoradion kehittämiseen. Sitä käytetään yleensä yhdessä USRP-kortin kanssa. Kommunikoimiseen Gnu Radion ja sovelluksen välillä käytetään Python-skriptejä. Varsinainen sovellus toteutetaan Qt-kehitysympäristöllä ja C++-ohjelmointikielellä.

Työn tuloksena valmistui sovellus, joka toteuttaa sille määrätyt toiminnallisuudet ja toimii hyvänä perustana jatkokehitystä ajatellen. Sovellusta hyödynnetään päivittäin ultraäänikameran tuottaman kaikuinformaation tutkimisessa ja siitä muodostetun kuvan piirtämisessä.

Asiasanat: ultraäänikamera, qt, usrp, Gnu Radio

ABSTRACT

Oulu University of Applied Sciences
Information Technology, Information Security

Author: Hannu-Pekka Shemeikka
Title of thesis: The Application for an Ultrasound Camera Prototype in a PC Environment
Supervisor: Esko Harvala
Term and year when the thesis was submitted: Autumn 2010 Number of pages: 40 + 5

ABSTRACT

This Bachelor's thesis is commissioned by Simsonar Oy, which develops an ultrasound camera for underwater imaging. The purpose of this thesis is to build an application for this ultrasound camera for PC environment. The aim is to create a working application which is capable of forming a moving image from a signal produced by the camera. Secondary aims are saving the image to a file and playing the image from the file.

The ultrasound camera prototype is based on a USRP card and its customized daughter boards. Communication with the USRP card is handled by a Gnu Radio. It is a free software development toolkit for implementing software radios. It is mainly used together with the USRP card. Python scripts are used for communication between the application and the Gnu Radio. Qt framework and C++ are used to build the application.

The result of this thesis is a working application which achieved its objectives and works as a good foundation for future development. The application is used in everyday work, which includes studying the ultrasound camera acoustics information and displaying the image on display.

Keywords: ultrasound camera, qt, Gnu Radio, usrp

SISÄLLYS

KÄYTETYT TERMIT JA LYHENTEET

AD	Analog to Digital
DA	Digital to Analog
AFE	Analog Front End
FPGA	Field-programmable Gate Array
POSIX	Portable Operating System Interface for Unix
USRP	Universal Software Radio Peripheral

1 JOHDANTO

Simsonar Oy on pieni, keväällä 2009 perustettu yritys, jonka tavoitteena on tuotteistaa vedenalaiseen ultraäänikuvantamiseen liittyviä innovaatioita. Ultraäänen käyttö veden luotaamisessa on tunnettua ja laajalti käytettyä tekniikkaa. Koska ultraääni etenee vedessä paremmin kuin valo, voidaan sen avulla luodata sameissa vesissä tai pitkillä etäisyyksillä, joissa optiset kamerat eivät toimi. Tyypillinen ultraääntä käyttävä laite on kaikuluotain tai viistokaikuluotain, jossa on 1-3 äänikeilaa ja jolla kuvantaminen perustuu liikkuvaan kantolaitteeseen (pinta-alus, sukellusrobotti).

Simsonar Oy kehittää ultraääntä käyttävää kameraa, joka pystyy yhtä aikaa kuuntelemaan suurta määrää kapeita ultraäänikeiloja ja muodostamaan keilojen kaikuinformaatiosta kuvia tai videotyyppistä liikkuvaa kuvaa. Tyypillisessä käytössä kamera suunnataan alaviistoon veteen ja se muodostaa kuvan ilman kantolaitteen liikettä. Kuva muodostetaan samalla tavalla kuin tutkassa, ja se on tutkakuvan kaltainen.

Ultraäänikameroita, jotka pystyvät muodostamaan kuvaa ilman kantolaitteen liikettä, voidaan käyttää monissa erilaisissa vedenalaisissa tehtävissä:

- kohteen etsintä ja identifiointi
- vedenalainen rakentaminen ja vedenalaiset tarkastukset
- kalatutkimus, populaatiotutkimus
- navigoinnin, paikantamisen tuki, vedenalaisten esteiden havaitseminen
- pelastustoimi
- yrityskäyttö, viranomaiskäyttö, sotilaskäyttö.

Kamera voidaan kiinnittää alukseen, sukellusrobottiin, kiinteään vedessä olevaan telineeseen tai sukeltaja voi käyttää sitä käsivaralla.

Syksyllä 2009 Simsonar Oy teki mittauksia koejärjestelmällä. Mittausten tarkoituksena oli todentaa suunnttelun pohjana olevien innovaatioiden toiminta

käytännössä. Mittaukset onnistuivat hyvin ja niissä voitiin todeta perusratkaisujen toimivan suunnitellusti.

Markkinaselvitysten ja koejärjestelmämittausten tulosten pohjalta tehtiin päätös seuraavan vaiheen, prototyypin kehittämisen, käynnistämisestä. Prototyypin kehittämisessä joudutaan rakentamaan kaikki ultraäänikameran tekniset ratkaisut: keilojen muodostus ja ultraäänioptiikka, mekaniikka, elektroniikka ja ohjelmisto. Keilojen muodostus ja ultraäänioptiikka perustuvat koejärjestelmän ratkaisuihin ja niiden edelleenkehittämiseen, muut alueet on käynnistetty alkutekijöistään.

Prototyypin kehittämisessä hyödynnetään open source -ratkaisuja sekä ohjelmistossa että elektroniikassa. Molemmissa alueissa joudutaan kuitenkin kehittämään sovellusspesifisiä osuuksia. Mekaniikka ja ultraäänioptiikka tehdään täysin oman suunnittelun pohjalta. Resursseina ovat Simsonar Oy:n osakkaat, opinnäytetyön tekijät ja alihankkijat.

Tämän opinnäytetyön kohteena on kameran prototyypin ohjelmisto-osuus. Ohjelmistotyössä kehitetään PC-laitteistossa toimiva järjestelmä, jolla

- hallitaan kokonaisuutta
- muodostetaan kaikuinformaatiosta kuva
- hallitaan kameran käyttöä ja sen parametrointia
- rakennetaan ohjelmisto-laitteisto-rajapinta ja tietoliikennerajapinta
- mahdollistetaan DSP- ja kuvanparannusmenetelmien käyttö sekä kehitetään ensimmäiset algoritmit.

Tässä työssä keskitytään kolmeen ensimmäiseen alueeseen, sovellukseen ja kuvanmuodostukseen. Muita alueita käsitellään siinä laajuudessa, kuin se on kokonaisuuden ymmärtämisen takia tarpeellista.

2 MÄÄRITELMÄ

Simsonar Oy kehittää ultraäänellä toimivaa vedenalaista kameraa, jolla voidaan korvata optinen kamera, kun olosuhteet eivät ole otolliset optisen kameran käytölle. Ultraäänikameran toiminnan periaatteena on lähettää veteen ultraäänipulssi, joka osuessaan kohteeseen heijastuu takaisin kameraan. Tätä periaatetta on kuvattu liitteessä 2. Käyttökohteita ovat erityisesti sameat tai humuksiset sisävedet, rannikot ja rakennetut vesialueet, kuten satamat. Syvillä merialueilla, jossa luonnonvaloa ei enää ole, voidaan ultraäänikameraa myös käyttää.

Opinnäytetyön aiheen määrittely pohjautuu kameran kokonaistoimintaan ja rakenneratkaisuihin. Tämän vuoksi ne käydään aluksi lyhyesti läpi. Kameran järjestelmässä (liite 1) on seuraavat pääosat:

- kamerapää, veden alla
- mahdollinen liityntäyksikkö maalla
- tietokone kameran ohjaamiseen ja kuvan näyttöön
- kaapelointi.

Kamerapäässä on ultraäänen tuottamiseen ja vastaanottamiseen tarvittavat toiminnallisuudet ja rakenteet:

- ultraäänilähetin
- monikeilainen ultraäänivastaanotin
- analogia- ja digitaalielektroniikkaa ultraäänisignaalin lähettämiseen ja vastaanottamiseen
- tiedonsiirto-ohjelmisto
- vedenpitävä mekaniikka.

Kuivan pään liityntäyksiköllä voidaan toteuttaa erityisiä tietoliikenne- ja virransyöttötarpeita.

Kameraa ohjataan ja käytetään tietokoneen avulla. Tietokone toimii myös

näyttölaitteena, jonka kuvaruudulla kameran muodostamaa kuvaa näytetään. Järjestelmää käytetään lähes yksinomaan ulkokäytössä, joten kannettava tietokone on luonnollinen valinta.

Kamerajärjestelmän kaapeloinnille on tiettyjä perusolettamuksia ja -vaatimuksia.

- Tavoitteena on käyttää standardikaapeleita ja -liityntöjä
- Märän pään liittimien pitää kuitenkin olla vedenpitäviä
- Kaapeloinnin pituus ja tietoliikenteen nopeusvaatimukset vaikuttavat kaapelointiratkaisuihin.

Kameran perustoiminnallisuudet ovat

- muodostaa kuvaa ultraäänen kaikuinformaatiosta
- kuva on perimmiltään monokromaattinen, värejä voidaan käyttää tehostimena
- liikkuva kuva ja still-kuva
- kuvan talletus tiedostoon ja toisto tiedostosta
- lisätoiminnallisuudet kuten web-kamera ja on-line sharing.

Opinnäytetyö kohdistuu kamerajärjestelmän prototyypin hallintasovelluksen suunnitteluun, toteutukseen ja testaukseen. Jatkossa tässä työssä toteutettua ohjelmistoa kutsutaan sovellukseksi. Muita ohjelmisto-osuuksia ovat ohjelmisto-laitteisto -rajapinta ja tietoliikennerajapinta, jota kutsutaan tuotealustaksi, ja algoritmit kuvanlaadun parantamiseksi, jota kutsutaan kuvanparannukseksi. Tuotealustan ja kuvanparannuksen suunnittelu, toteutus ja testaus on tehty eri resursseilla mutta käsi kädessä sovelluksen kehittämisen kanssa, ja molemmat ovat vaikuttaneet toistensa ratkaisuihin.

2.1 Sovelluksen päävaatimukset

Sovelluksen päävaatimukset ovat:

- toiminta yleiskäyttöisessä (kannettavassa) tietokoneessa
- siirrettävyys ja laajennettavuus

- helppokäyttöisyys ja toimintalogiikka, joka vastaa kokonaisjärjestelmän rakennetta ja käyttöä
- kyky näyttää liikkuvaa kuvaa 1...10 kuvaa sekunnissa taajuudella
- kuvatiedon tallennus ja toisto.

Prototyypin järjestelmäsuunnittelussa Simsonar Oy päätyi käyttämään tuotealustan pohjana open source -lisenssillä saatavana olevaa muunninkorttia Universal Software Radio Platform (USRP) ja ko. korttia tukevaa open source -ohjelmistoa nimeltä Gnu Radio. Vaikka Gnu Radio ja USRP ovat suunniteltu ja kehitetty ohjelmistoradioratkaisuja varten, tarjoaa USRP DA- ja AD-muuntimineen muunninratkaisun ja tietoliikenneratkaisun, joiden varaan kehitettävän kameran tuotealusta voidaan kohtuullisella työllä rakentaa. Gnu Radio -ohjelmisto puolestaan tarjoaa valmiin toimivan ratkaisun tietoliikenteen alemmille kerroksille (USB2.0), USRP:n ohjaukselle sekä sovelluksen ja tuotealustan ohjelmarajapinnalle.

2.2 Sovelluksen yleismäärittely

Tuotealustan osalta tehty valinta antaa rajat sovelluksen kehittämiselle päävaatimuksista lähtien.

- Yleiskäyttöinen tietokoneympäristö: Gnu Radio -ohjelmisto on kehitetty Linux-ympäristössä ja sen Windows-version toiminnallisuudessa on vielä puutteita, joten sovellus rakennetaan Linux-ympäristössä.
- Siirrettävyys ja laajennettavuus, helppokäyttöisyys: Linux-ympäristö tarjoaa kaksi graafisen käyttöliittymän kehittämisympäristöä, GTK+-ympäristön ja Qt-ympäristön. Edellinen on täysin open source -ohjelmisto, jälkimmäinen nykyisin Nokia Oyj:n omaisuutta ja tarjolla kahden lisenssin systeemillä: open source -lisenssi harrastekäyttöön ja ammattilisenssi liiketoimintakäyttöön. Qt on alun alkaen rakennettu toimimaan erilaisissa käyttöjärjestelmäympäristöissä, toisin kuin GTK+, joka on Linux-spesifinen. Kehitettävä sovellus on graafinen ja perustuu Qt-ympäristön käyttöön.

- Helppokäyttöisyys: kameran loppukäyttäjät ovat ihmisiä, jotka tuntevat hyvin PC-laitteiden käytön mutta eivät ole varsinaisia tietokoneasiantuntijoita. Sovellus rakennetaan tavanomaisen PC-sovelluksen tyyliin, eli se sisältää ikkunoinnin, valikot, valintanäppäimet, tekstinsyöttöruudut ja ilmoitusikkunat.
- Helppokäyttöisyys ja toimintalogiikka, joka vastaa kokonaisjärjestelmän rakennetta ja käyttöä: sovelluksen ainoa tarkoitus on toimia yhdessä Simsonar Oy:n kameran kanssa. Tästä syystä sovelluksen looginen toiminta, asetustiedot, statustiedot, tietojen käsittely, esitysmuoto ja tarkkuus ovat kokonaisjärjestelmän toiminnoista johdettuja ja niitä tukevia.
- Kyky näyttää liikkuvaa kuvaa 1...10 kuvaa sekunnissa taajudella, tuotealusta, erityisesti Gnu Radio -ohjelmisto ja käytetty USB2.0-liityntä yhdessä USRP-kortin korkean muunnoskapasiteetin ja suuren datamäärän kanssa kuormittavat tavanomaista kannettavaa tietokonetta paljon. Lisäkuormitusta aiheuttavat kuvanparannuksessa käytettävät menetelmät. Näistä syistä kuvan muodostuksen pitää olla mahdollisimman tehokasta, jotta tavoite 10 kuvaa sekunnissa saavutetaan.
- Kuvatiedon tallennus ja toisto: sovelluksen pitää pystyä tallentamaan kameralta tuleva kuvainformaatio tiedostoon ja toistamaan se tiedostosta ilman, että kamera on kytkettynä tietokoneeseen. Määrittelyn mukaisesti kuvaa pitää pystyä tallentamaan yhtä aikaa reaaliaikaisen kuvanäytön rinnalla. Tallennettavaan tiedostoon on myös tallennettava informaatiota, jota tarvitaan kuvatiedoston toistossa ja tunnistamisessa. Sovelluksessa pitää myös olla videosoittimen perustoiminnallisuudet.

3 TOIMINTAYMPÄRISTÖ

Alkuperäisenä suunnitelmana oli toteuttaa sovellus Windows-käyttöjärjestelmälle tehtynä käyttäen C#-ohjelmointikieltä. Tästä suunnitelmasta jouduttiin kuitenkin luopumaan heti työn alkuvaiheessa Gnu Radion vaatimusten takia. Gnu Radiosta ei ollut tarpeeksi vakaata ja helposti asennettavissa olevaa versiota Windows-ympäristöön, vaan se toimi lähes yksinomaan Linux-käyttöjärjestelmissä. Tästä syystä sovellus päädyttiin tekemään Linux-käyttöjärjestelmässä. Jakeluksi valittiin Ubuntu 9.10. Valintaperusteina käytettiin sen hyvää yhteensopivuutta Gnu Radion kanssa sekä helposti saatavilla olevaa käyttötukea.

Lopullinen tuote tulee toimimaan Windows-käyttöjärjestelmässä, joten tämän prototyyppivaiheen sovelluksen tulisi olla mahdollisimman alustariippumaton uudelleenohjelmoinnin vähentämiseksi. Tästä syystä päädyimme käyttämään sovelluksen kehityksessä Qt-kehitysympäristöä, josta käytetään versiota 4.6.

Käytettävinä ohjelmointikielinä ovat C++ ja Python. C++:lla toteutetaan itse varsinainen sovellus ja Pythonilla toteutetaan välikerros Gnu Radion ja USRP:n kanssa kommunikoimiseen. Graafinen käyttöliittymä luodaan Qt:lla.

Ultraäänikameran prototyyppi käyttää USRP-korttia ja siihen erikseen rakennettavia tytärkortteja kaikuluotainsignaalin lähettämiseen ja vastaanottamiseen. Ultraäänikameran kuvauksen periaattetta on selostettu liitteessä 2.

Sovelluksella tulee voida vastaanottaa USRP:ltä tuleva signaali, muuntaa se ymmärrettävään muotoon, muodostaa tästä datasta liikkuvaa kuvaa sekä tallentaa ja toistaa sitä. Sovelluksen tulee myös voida lopettaa ja käynnistää Gnu Radio hallitusti sekä parametreja tulee voida välittää USRP:lle asti.

3.1 Qt

Qt on alustariippumaton kehitysympäristö ohjelmistojen ja graafisten käyttöliittymien tekemiseen. Norjalainen Trolltech aloitti Qt:n kehittämisen vuonna 1991 ja jatkoi sen kehitystä aina vuoteen 2008, jolloin Nokia Oyj osti Trolltechin. Nykyään Qt:n kehitys jatkuu Nokian alaisuudessa (Qt, Nokia Oyj, hakupäivä 11.9.2010).

Qt-kirjastoa voidaan käyttää useiden ohjelmointikielien kanssa, kuten C++, Java ja C#. Qt on avoimen lähdekoodin ohjelmisto ja se on saatavilla kahdella eri lisenssillä riippuen käyttötarkoituksesta. Kaupalliseen käyttöön on oma kaupallinen lisenssi ja muuhun käyttöön on ilmainen GNU LGPL -lisenssi (Qt, Nokia Oyj, hakupäivä 11.9.2010).

Qt:tä voidaan käyttää Windows-, Linux- ja Mac Os X -käyttöjärjestelmillä sekä myös Symbian S60- ja Maemo -alustoilla.

3.2 Gnu Radio

Gnu Radio on GPL-lisenssillä levitettävä vapaa ohjelmistokehityspaketti ohjelmistopohjaisen radion kehittämiseen (Welcome to GNU Radio, GNU Radio, hakupäivä 11.9.2010).

Gnu Radiolla kehitettävät ohjelmat koostuvat erilaisista datan käsittelylohkoista, joita käyttäjä yhdistelee saadakseen haluamansa lopputuloksen. Nämä lohkot ovat jaettu signaalilähde-, Signaalinlähetin- ja signaalinkäsittelylohkoihin (Welcome to GNU Radio, GNU Radio, hakupäivä 11.9.2010).

Gnu Radion kehittämisen idea on, että kukin käyttäjä voi rakentaa omia lohkojaan C++-kielellä ja sisällyttää nämä lohkot omalle koneelleen asennettuun Gnu Radioon. Lohkojen liittäminen toisiinsa ja Gnu Radion käynnistys sekä sammutus hoidetaan Python-skriptillä.

3.2.1 Signaalinvastaanotinlohkot

Signaalinvastaanotinlohkot ovat käyttäjän määrittelemien signaalien lähteitä. Näitä voivat olla digitaalisesti määritellyt ja luodut signaalit, tiedostot tai ulkopuolisesta lähteestä, kuten mikrofonista, tulevat signaalit (Welcome to GNU Radio, GNU Radio, hakupäivä 11.9.2010).

3.2.2 Signaalinlähetinlohkot

Signaalinlähetinlohkot ovat Gnu Radio ohjelman ulostuloja. Näitä ovat yleensä binaaritiedostot, äänikortit tai graafiset moduulit, kuten Pythonin Wx widget-kirjasto, jonka avulla Gnu Radion data saadaan suoraan graafiseen muotoon näytölle (Welcome to GNU Radio, GNU Radio, hakupäivä 11.9.2010).

3.2.3 Signaalinkäsittelylohkot

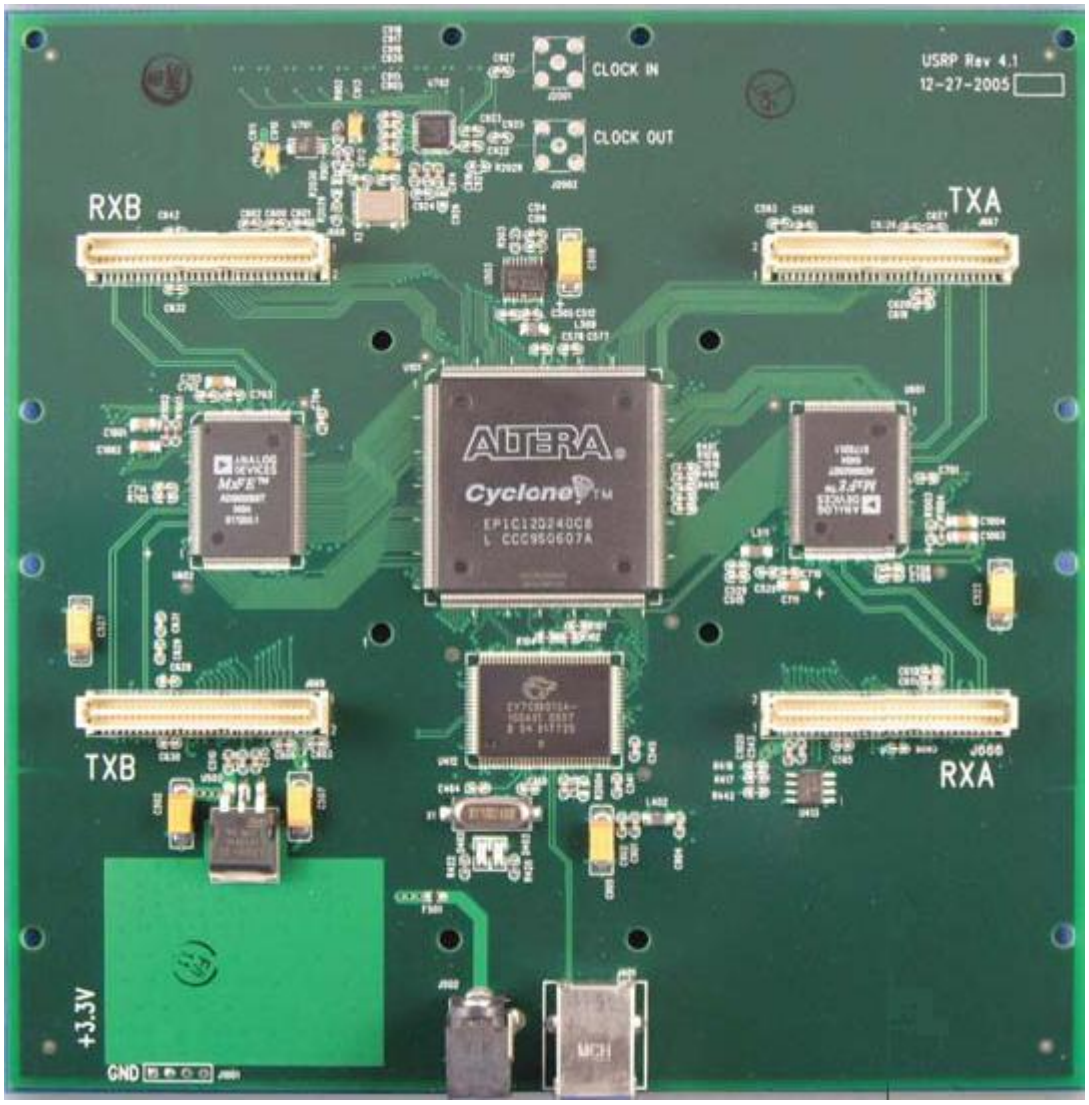
Signaalinkäsittelylohkot ovat lähdelohkon ja sinkkilohkon välissä. Nimensä mukaisesti tässä lohkossa käsiteltään lähdelohkosta tuleva datavirta käyttäjän määrittelemien asetusten mukaisesti.

Gnu Radio sisältää lukuisia käsittelylohkoja eri tarkoituksiin. Saatavilla on mm. suodattimia, vahvistimia sekä modulaattoreita. Kaikki käsittelylohkot on kirjoitettu C++:lla ja ovat julkisesti nähtävissä (Welcome to GNU Radio, GNU Radio, hakupäivä 11.9.2010).

3.3 Universal Software Radio Peripheral

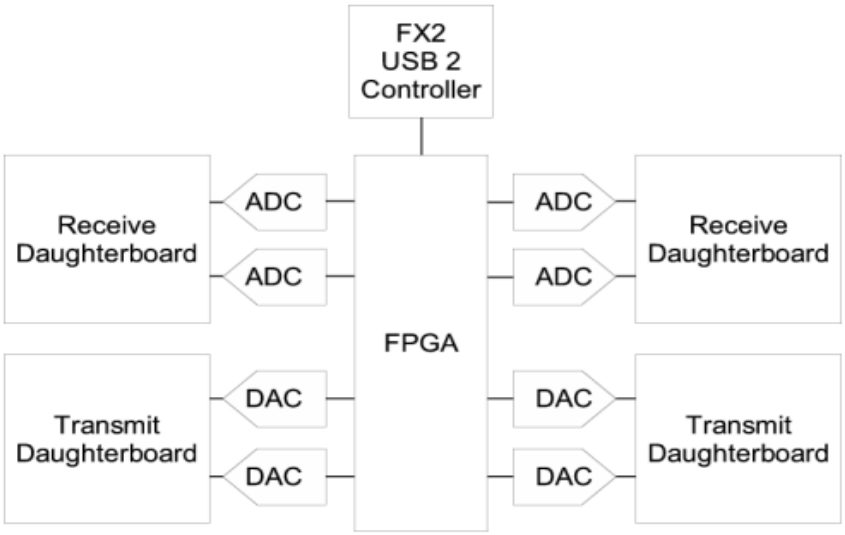
USRP on Ettus Research LLC kehittämä ja myymä USB-väyläinen laite, joka on tarkoitettu ohjelmistopohjaisten radioiden alustaksi. USRP:tä voidaan käyttää muidenkin ohjelmien tai kehityspakettien kanssa kuin Gnu Radion mutta suositeltavaa on käyttää Gnu Radiota, sillä USRP-kortti on kehitetty läheisessä yhteistyössä Gnu Radio -projektin kanssa (Ettus Research LLC, Ettus Research LLC, hakupäivä 9.9.2010)

Simsonar Oy:n kehittämässä ultraäänikameran prototyypissä USRP:n rooli on toimia muunninkorttina, josta käytetään AD-muuntimia ja DA-muunninta.



Kuva 1. Paljas USRP-kortti (USRP Motherboard, Gnu Radio, hakupäivä 10.9.2010).

Kuvasta 1 näkyy paljaan USRP-kortin ulkoinen rakenne. Se sisältää 4 AD-muunninta ja 4 DA-muunninta. Nämä muuntimet ovat yhdistetty FPGA-prosessoriin. Kortti sisältää myös USB 2.0 -ohjaimen sekä 4 tytärkorttiliitintä. Näiden komponenttien tarkempi liityntä selviää kuvasta 2.



Kuva 2. USRP-kortin rakenne (Blossom, Exploring GNU Radio, hakupäivä 9.9.2010).

4 TOTEUTUS

Työn kohteena on sovellus, jolla voidaan hallita kaikuluotainta ja toistaa kaikuluotaimen lähettämää informaatiota näytöllä liikkuvana kuvana. Kokonaisuus koostuu itse sovelluksesta, Python-rajapinnasta sekä Gnu Radiosta. Seuraavaksi tutustumme tarkemmin tähän kokonaisuuteen ja sen toteutukseen pääpainon ollessa sovelluksessa.

4.1 Yleistä

Työn aloittaminen alkoi perehtymisellä ultraäänen, Gnu Radion, Pythonin ja Qt:n maailmoihin. Jokainen näistä oli entuudestaan tuntematon, joten lähtökohdat työlle olivat haastavat.

Myös toimintaympäristön saaminen toimintakuntoon oli oma haasteensa. Kun kyseessä on pienistä erillisistä paketeista koostuva kokonaisuus, on niiden välillä yleensä pientä yhteensovittamisen tarvetta. Pahimmat ongelmat aiheutuivat puutteellisesta dokumentaatiosta, jolloin apua asennusongelmiin joutui etsimään Internetistä ja eri sähköpostilistoilta. Useiden päivien asennusongelmien jälkeen toimintaympäristö saatiin lopulta kuntoon ja varsinainen työ saattoi alkaa.

4.2 Sovellus

Sovellus on toteutettu Qt-kehitysympäristöllä ja sisältää yli 3500 lähdekoodiriviä. Sovelluksen rakenne jakaantuu kahteen osaan, reaaliaikaiseen kuvan toistamiseen suoraan kaikuluotaimelta ja nauhoitetun kuvan toistamiseen tiedostosta.

Sovellus on toteutettu siten, että käynnistyessään se ei vaadi kaikuluotaimen olevan kytkettynä tietokoneeseen. Tämä mahdollistaa sovelluksen käyttämisen ja tallennetun tiedoston toistamisen ilman kaikuluotainta. Sovellus myös

käynnistyy nopeammin ja vie vähemmän resursseja, kun kaikuluotaimen kanssa kommunikoimiseen tarkoitettuja säikeitä ei tarvitse luoda ja käynnistää.

Sovellus ja kaikuluotain ovat vielä prototyyppiasteella, joten sovellus ei vielä sisällä kaikkia valmiille sovellukselle määriteltyjä ominaisuuksia, kuten kuvainformaation tallentamista jossakin yleisesti käytössä olevassa videoformaatissa tai kuvainformaation katsomista verkon ylitse toiseen koneeseen kytkettynä olevasta kaikuluotaimesta.

Sovellus käyttää kaikuluotaimen kanssa kommunikoimiseen kolmea rajapintaa:

- Python-skriptirajapintaa
- jaettua muistia käynnistyksenaikaisille ja toiminnanaikaisille parametreille
- rengaspuskuria kuvainformaatiolle.

Python-skriptirajapinta on funktiokutsutyyppinen synkroninen rajapinta ja muut kaksi rajapintaa ovat säikeiden välisiä asynkronisia rajapintoja.

Ensimmäinen rajapinta on yksinkertainen Python-skripti, jonka tehtävänä on käynnistää ja sammuttaa Gnu Radio sekä välittää käynnistyksenaikaista tietoa sovelluksen ja Gnu Radion välillä. Varsinaiseen toimintaan ja tiedonsiirtoon Python-skripti ei osallistu lainkaan.

Varsinainen toiminnanaikainen rajapinta on toteutettu jaetulla muistilla sovelluksen ja Gnu Radion välillä. Tiedonvälitys Gnu Radiolta sovellukselle hoidetaan rengaspuskurin avulla. Jaetun muistin kautta sovellus välittää Gnu Radiolle:

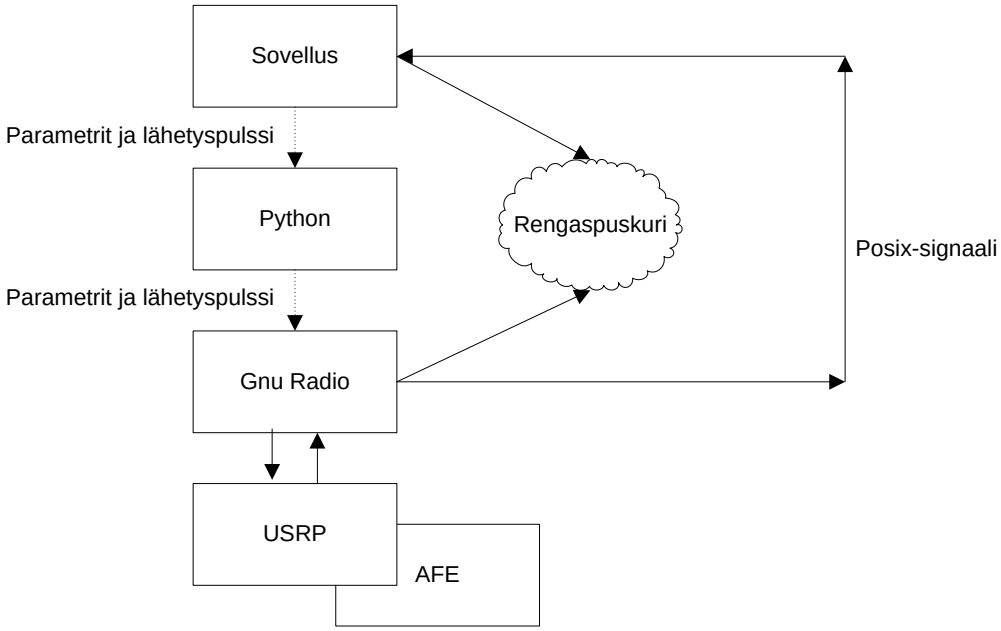
- käynnistyksen- ja toiminnanaikaisia parametreja
- lähetettävän ultraäänipulssin koodattuna.

Rajapinta on rakennettu niin, että vain sovellus pystyy kirjoittamaan tähän jaettuun muistiin Gnu Radion ollessa vastaanottajana mutta lähetettävän pulssin käsittely on lisäksi suojattu Posixin mutex-suojauksella.

Rengaspuskurin tarkoitus on siirtää Gnu Radion USRP:n kautta keräämä kaikuinformaatio sovellukselle. Gnu Radio vain kirjoittaa rengaspuskuriin ja sovellus vain lukee rengaspuskurista. Tämän lisäksi käytetään mutex-suojausta. Rengaspuskurirajapinta on toteutettu sovelluksen puolella siten, että sitä lukeva säie on odotustilassa ja aktivoituu vain Gnu Radion lähettämästä Posix-signaalista.

4.2.1 Rakenne

Sovellus koostuu useista säikeistä, joista jokaisella on oma erityinen tehtävänsä, sekä rengaspuskurista sovelluksen ja Gnu Radion välillä. Gnu Radio vastaanottaa USRP:n sille lähettämän kaikuinformaation ja kirjoittaa sen rengaspuskuriin. Tällä tavoin voidaan informaatiota siirtää yksinkertaisesti ja tehokkaasti sovelluksen ja Gnu Radion välillä.



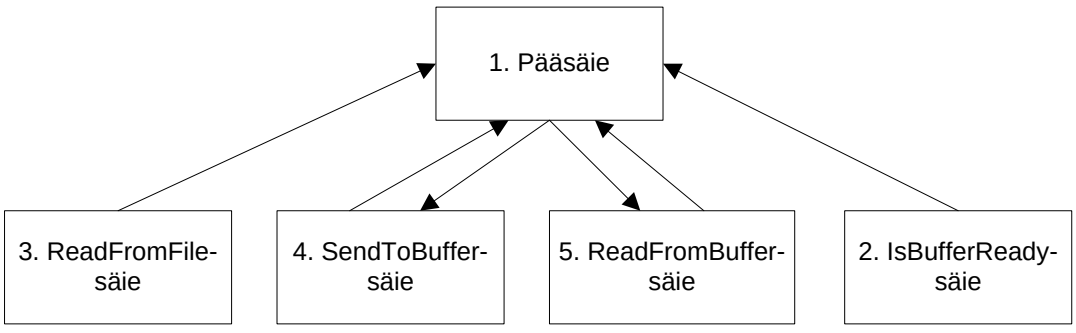
Kuva 3. Sovelluksen toiminnallisuuden rakenne.

Kuvassa 3 on esitetty sovelluksen toiminnallisuuden rakenne. Sovellus lähettää parametreja Python-skriptille, joka käynnistää Gnu Radion. Gnu Radio pyörittää USRP:tä ja vastaanottaa siltä saatavan kaikuinformaation. Tämän

kaikuinformaation Gnu Radio kirjoittaa rengaspuskuriin, josta sovellus lukee sen ja muodostaa siitä informaatiosta liikkuvan kuvan näytölle esitettäväksi.

Sovelluksessa on säikeita on viisi kappaletta (kuva 4). Säikeet ovat toteutettu Qt:n omalla QThread-luokalla. Seuraavaksi käydään läpi säikeiden toiminta ja tarkoitus.

- Ensimmäinen säie on Qt:n pääsäie, joka on ainoa säie, jonka on sallittua hoitaa graafiseen käyttöliittymään liittyviä operaatioita, kuten näytölle piirtämistä.
- Toinen säie on kaikuinformaatin rajapinta Qt:n ja Gnu Radion välillä. Säie odottaa Gnu Radiolta tulevaa Posix-signaalia. Signaali lähetetään Gnu Radiosta aina, kun kokonaisen kuvakehyksen edellyttämän määrä raakadataa on valmiina rengaspuskurissa. Tämän signaalin ansiosta sovellus tietää rengaspuskurin olevan täynnä ja voi lukea sen sisällön.
- Kolmas säie on tarkoitettu nauhoitetun kuvainformaation toistamiseen tiedostosta. Tämä säie lukee ja lähettää tiedostosta luetun raakadatan edelleen muille sovelluksen osille. Tällä rakenteella varmistetaan, että käyttöliittymä ei jäädy kesken tiedoston toiston.
- Neljäs säie käynnistää Python-skriptin (liite 4) ja syöttää sille parametreja välitettäväksi Gnu Radiolle ja USRP:lle. Tästä toiminnasta kerrotaan lisää kappaleessa 4.4.
- Viides säie lukee rengaspuskurin sisällön ja muokkaa tätä sisältöä esityskelpoiseen muotoon sekä lähettää sen lopulta pääsäikeelle.

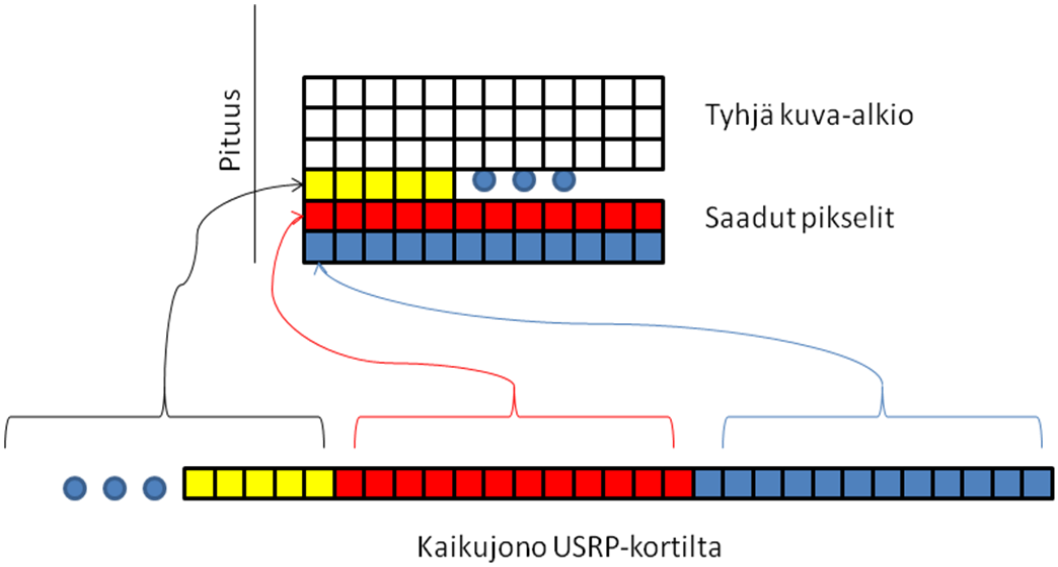


Kuva 4. Sovelluksen säikeet ja niiden välittämien tietojen kulkusuunnat.

4.2.2 Kuvanmuodostus

Sovelluksen ehkäpä tärkein ominaisuus on muodostaa kaikuluotaimelta tulevasta kaikuinformaatiosta liikkuvaa kuvaa. Määritysten mukaisesti muodostettava kuva on harmaasävyistä.

Jotta kuvanmuodostuksesta pystyttäisiin ymmärrettävästi kertomaan, on ensin kerrottava hieman tämän ultraäänikameran prototyypin tavasta vastaanottaa kaikuinformaatiota. USRP-korttiin kytketyt AFE-tytärkortit sisältävät vastaanottokeiloja, joiden lukumäärän mukaisesti kaikuinformaatio pilkotaan ja kopioidaan taulukkoon tiettyjen asetusten mukaisesti (kuva 5). Vastaanottokeilojen lukumäärä määrittelee muodostettavan kuvan leveyden. Kun taulukko tulee täyteen, kirjoitetaan tämän taulukon sisältämä kaikuinformaatio rengaspuskuriin.



Kuva 5. Kaikuinformaation sijoitus taulukkoon.

Kuvanmuodostuksen periaate on samanlainen kuin kuvaputkitelevisiossa tai -näytössä: jatkuva signaali pilkotaan raidoiksi kuva-alueelle. Kuva alkaa vasemmasta alalaidasta ja edustaa lähinnä kameraa olevaa vesialuetta, josta kaiku tulee ensimmäisenä.

Kuvanmuodostus sovelluksen puolella sisältää neljä vaihetta:

- taulukon lukeminen rengaspuskurista
- taulukon alkioiden arvojen tyyppimuunnos
- pikselien sijoitus oikeille paikoilleen
- napakoordinaatistomuunnos ja alkioiden kopioiminen uuteen taulukkoon.

Kuvanmuodostus aloitetaan sovelluksen puolella lukemalla rengaspuskurin sisältö kuvataulukkoon. Taulukon jokaisen alkion arvo on short-tyyppistä. Sovelluksessa käytettävä kuvanmuodostusfunktio pystyy kuitenkin käyttämään vain unsigned char -tyyppisiä lukuarvoja, joten alkion arvoille täytyy tehdä tyyppimuunnos. Tämä muunnos suoritetaan jakamalla jokainen arvo luvulla 64. Sovelluksen vastaanottamat lukuarvot ovat korkeintaan 2^{14} eli 16384. Tämä arvo jaetaan unsigned char -tyypin maksimiarvolla 2^{8} eli 256. Tästä jakolaskusta saatu arvo on 64 eli 2^{8}. Tyyppimuunnoksen jälkeen taulukon alkioiden arvot sijoittuvat välille 0 - 255, eli ovat unsigned char -tyyppisiä.

Kuvataulukon jokainen alkio vastaa yhtä kuvapikseliä. Alkiossa olevan raakadatan arvo kertoo minkä sävyinen kyseinen pikseli on. Jokainen pikseli voi saada arvon väliltä 0 - 255. Kuten taulukosta 1 käy ilmi, on pikselin arvo suoraan verrannollinen kaikuinformaation sisältämän signaalin tasoon. Mikäli alkion arvo on lähellä lukua 255, on pikselin kohdalle osunut kohde, josta on palautunut voimakas osumasignaali ultraäänikameralle. Tällainen kohde on esimerkiksi kala tai muu tiivis esine. Lähellä nollaa olevat pikselien arvot eivät sisällä mitään merkittävää informaatiota, vaan ovat taustakohinaa, jota syntyy esimerkiksi veden liikkeestä.

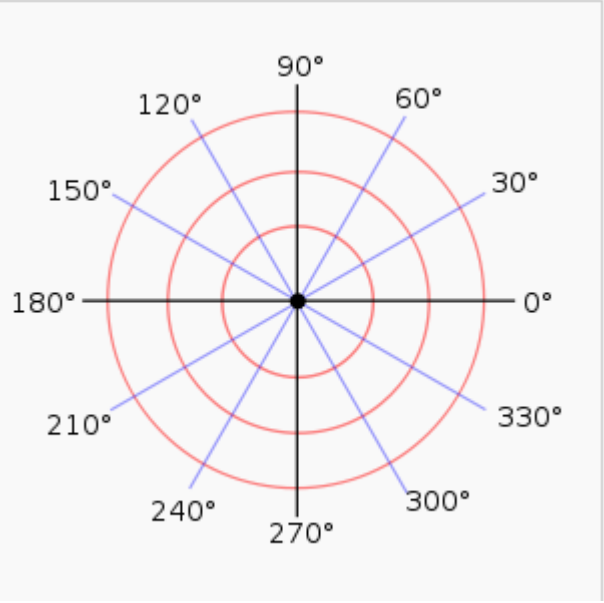
TAULUKKO 1. Pikselin arvon selitys.

Arvo	Väri	Merkitys
0	Musta	Taustakohinaa, ei osumasignaalia
128	Harmaa	Heikko osumasignaali
255	Valkoinen	Erittäin voimakas osumasignaali

Rengaspuskurista luetun kuvataulukon jokainen rivi vastaa tietyllä ajanhetkellä vastaanotinliuskoilta talteen otettuja kaikuja. Taulukon ensimmäinen rivi sisältää erittäin läheltä tulleita kaikuja ja viimeinen rivi hyvin kaukaa tulleita kaikuja.

Ennen varsinaista kuvanmuodostusfunktion (liite 3) kutsumista joudutaan taulukon jokaisen rivin sisältö asettamaan oikeille paikoilleen ennalta määrättyjen asetusten mukaisesti. Kun rivin jokainen pikseli on oikealla paikallaan ja vastaa näin käytännössä ultraäänikameran yhden keilan sijoitusta keilarivissä, tehdään taulukon sisällölle napakoordinaatistomuunnos.

Napakoordinaatisto on kaksiuloitteinen koordinaatisto, jossa jokainen piste on määritetty kiertokulman θ ja säteen r funktiona (Wikipedia 1).



Kuva 6. Napakoordinaatisto (Napakoordinaatisto, Wikipedia. 2010. Hakupäivä 8.9.2010).

Ultraäänikameran lähettimen lähettämä äänipulssi etenee kamerasta poispäin ympyrägeometrisesti, kuten ympärisäteilevässä antennissakin. Palautuvia kaikuja kuunnellaan keiloilla. Nämä muodostavat yhdessä ympyräsektorin. Kameran näkemä kuva on napakoordinaatistomuodossa (kuva 6), joten sen

tuottama kaikuinformaatio on myös napakoordinaatistossa ja muotoa (r,n), jossa:

- r on kohteen etäisyys kamerasta eli taulukon rivi
- n on keilan numero.

Sovelluksen kaikuinformaatiosta muodostama kuva on kuitenkin karteesinen, eli tavallisessa x,y -koordinaatistossa, joten kuvataulukon pikselien paikat joudutaan muuntamaan vastaamaan karteesista koordinaatistoa. Tällä muunnoksella saadaan myös muodostettavasta kuvasta ympyräsektorin muotoinen.

Kuvataulukon sisältämät alkiot kopioidaan uuteen, isompaan taulukkoon niille paikoille, jotka napakoordinaatistomuunnoksella saadaan. Tämä muunnos suoritetaan jokaiselle kuvataulukon alkiolle käyttäen hyväksi kahta kaavaa:

$$y = (\mathrm{K} + 115) * \sin((114 - \mathrm{I} * 0.5) * (\frac{2 * \pi}{360})$$

$$x = (\mathrm{K} + 115) * \cos((114 - \mathrm{I} * 0.5) * (\frac{2 * \pi}{360}) \quad (1),$$

missä

y	on y-akselin piste,
x	on x-akselin piste,
K	on taulukon rivi,
I	on taulukon sarake,
π	on pii,
0.5	on yhden keilan leveys asteina,
114	on vakio, jota tarvitaan kiertämään kuvan keskiakseli x-akselista y-akselin suuntaiseksi. Näin saavutetaan pystysuorakuva.

Kaavat (1) ovat johdettu kaavoista:

$$x = r \cos \theta$$
$$y = r \sin \theta \quad (2),$$

missä

x on x-akselin piste,

y on y-akselin piste,

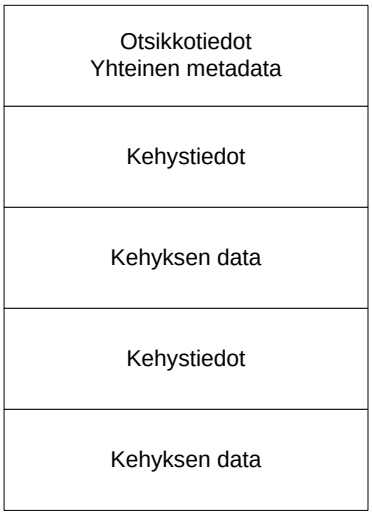
r on säde,

θ on kulma.

Kuvataulukon jokaisen rivin (K) jokainen alkio (I) saa x ja y -arvot, joiden avulla taulukon alkio sijoitetaan uuteen taulukkoon oikealle paikalleen. Tästä uudesta taulukosta muodostetaan Qt:n omalla QImage-funktiolla kuva, joka lähetetään pääsäikeelle näytölle piirtämistä varten. Jokainen piirrettävä kuva muodostetaan ja piirretään erikseen. Toistamalla tätä prosessi saadaan aikaan liikkuvaa kuvaa.

4.2.3 Tiedostoon tallennus

Sovellus voi tallentaa tiedostoon reaaliaikaista kuvaa kaikuluotaimelta. Kuvainformaatio tallennetaan tiedostoon raakadatana. Jotta tallennettua kuvaa voidaan myös toistaa sovelluksella, pitää tiedostoon tallentaa useita muuttujia varsinaisen kuvainformaation lisäksi. Sovellus käyttää omaa tiedostoformaattia, jonka rakenne selviää kuvasta 7.



Kuva 7. Tiedostoformaatin rakenne.

Tiedosto sisältää ensin yleistä metainformaatiota kameran ja sovelluksen asetuksista, jonka jälkeen tulevat kuvakehyksen metainformaatio ja varsinainen kuvainformaatio raakadatana.

Kuvainformaation tallennus tiedostoon raakadatana tarjoaa seuraavia etuja.

- Sovelluksen rakenne pysyy yksinkertaisena. Tallennettua tiedostoa toistettaessa voidaan käyttää hyväksi samoja funktioita ja luokkia, joita käytetään reaaliaikaisen kuvainformaation toistamiseen näytöllä.
- Tallennettua kuvainformaatioa toistettaessa voidaan sille suorittaa samoja toimintoja kuin reaaliaikaiselle kuvallekin. Esimerkkeinä mainittakoon kuvan värisävyn muuttaminen sekä erilaisten kuvanparannusalgoritmien käyttö toiston aikana käyttäjän haluamilla tavoin.
- Tallennus on nopeaa. Koska kuvainformaatiota ei muokata mitenkään, vaan se kirjoitetaan suoraan tiedostoon sellaisenaan, säästyy tässä kallista prosessointiaikaa.

4.2.4 Tallennetun tiedoston toistaminen

Tiedoston toistaminen tapahtuu samoilla tavoin kuin normaalin videotiedoston toistaminen tavallisella multimediasoittimella. Kun tiedosto on valittu ja Play-nappia on painettu, käynnistää sovellus lukusäikeen, joka lukee tiedostosta yhden kuvakehyksen kerrallaan ja lähettää tämän kehyksen sisältämän raakadatan edelleen sovelluksen muille säikeille käsiteltäväksi ja lopulta näytölle piirrettäväksi.

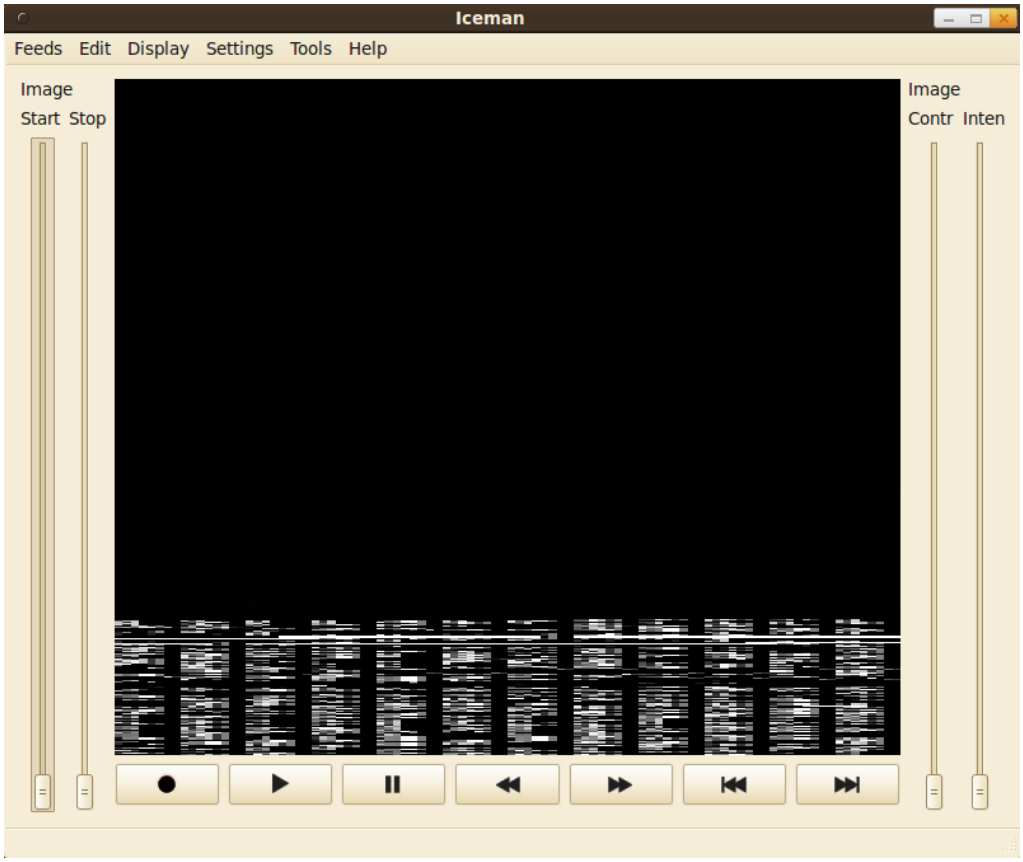
Soitto loppuu Stop-nappia painettaessa. Taukonäppäin ja kelausnäppäimet eivät sisällä vielä mitään toiminnallisuuksia.

4.3 Käyttöliittymä

Työ alkoi käyttöliittymän toteuttamisella käyttöliittymämäärittelyn mukaisesti. Käyttöliittymän tuli olla mahdollisimman modulaarinen, jolloin mahdolliset tulevat muutokset eivät aiheuttaisi suurta vaivaa eivätkä sekoittaisi jo olemassa olevaa rakennetta.

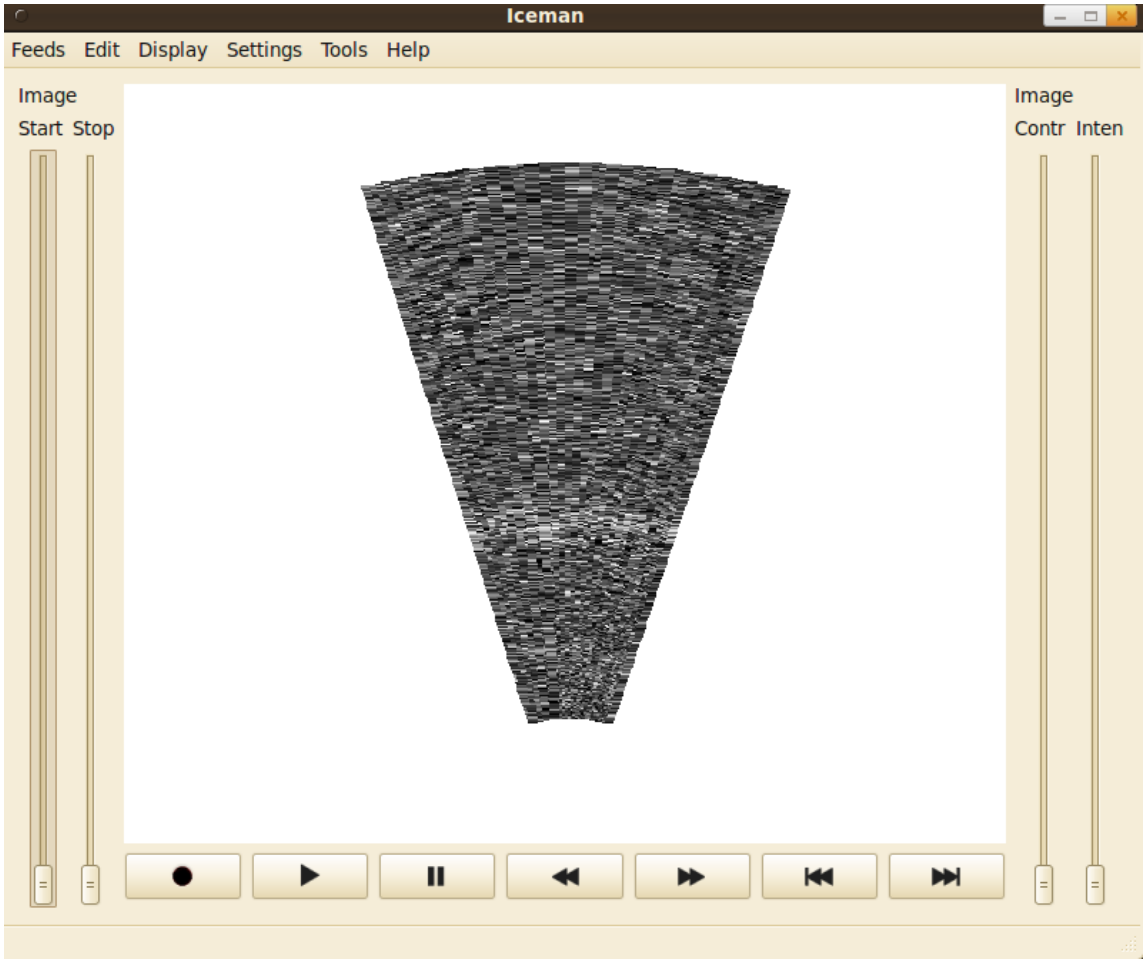
Käyttöliittymän kieleksi valittiin englannin kieli, jolloin sovellus vastaa paremmin kansainvälisten markkinoiden vaatimuksia.

Alla olevassa kuvassa (kuva 8) näkyy sovelluksen peruskäyttöliittymä. Tämä on sovelluksen pääikkuna, joka aukeaa heti sovelluksen käynnistyttyä. Käyttöliittymä on jaettu viiteen osa-alueeseen: työkaluriviin, kuvaruutuun, videotoistoon, kuvan väriavaruuden muuttamiseen sekä katseluetäisyyden muuttamiseen.



Kuva 8. Peruskäyttöliittymä.

Kuvassa 9 on kuvakaappaus sovelluksen USRP-kortilta lähettämästä datasta muodostetusta liikkuvasta kuvasta. Kuvaruudun tausta on valkoisena ja kaikuinformaatiosta muodostettu kuva näkyy keskellä ympyräsektorin muotoisena.

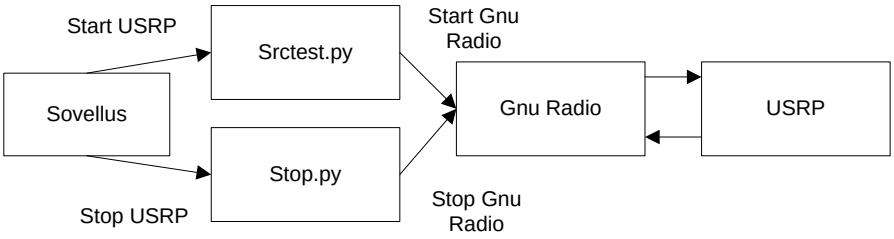


Kuva 9. Kuvakaappaus sovelluksen muodostamasta liikkuvasta kuvasta.

4.4 Python-rajapinta

Tässä kappaleessa käsitellään Python-rajapintaa, sen rakennetta ja toiminnallisuutta siinä määrin, kuin niiden ymmärtäminen on tarpeen tämän opinnäytetyön kannalta.

Python-rajapinta toimii välikerroksena sovelluksen ja Gnu Radion välillä. Tämä rajapinta on mukana, sillä Gnu Radio käyttää oletusarvoisesti Python-kieltä ja -skriptejä sovelluskerroksen toteuttamiseen. Rajapinta koostuu kahdesta Python-skriptistä: stopr.py ja srctest.py. Tämän rajapinnan yleisrakenne on esitetty kuvassa 10.



Kuva 10. Python-rajapinnan yleisrakenne.

Srctest.py-skriptin toiminta ja tarkoitus:
Kun yhteys kaikuluotaimeen halutaan käynnistää, kutsutaan sovelluksessa kyseistä skriptiä. Sille annetaan käynnistysparametreina muun muassa rengaspuskurin osoite ja kuvakehyksen koko. Skripti käynnistää Gnu Radion ja välittää saadut parametrit Gnu Radiolle.

Stop.py-skriptin toiminta ja tarkoitus:
Kyseinen skripti käynnistetään sovelluksessa, kun halutaan sammuttaa ultraäänikamera. Skripti käskee Gnu Radiota sammuttamaan USRP-kortti ja lopettamaan kaikuinformaation lähettämisen eteenpäin. Erillinen pysäytystoiminnallisuus tarvitaan, koska Gnu Radiota ei voida pysäyttää samasta säikeestä, mistä on käynnistetty.

4.5 Gnu Radio

Tässä kappaleessa käsitellään Gnu Radion sekä siihen lisättyjen sinkki- ja lähdelohkojen toimintaa lyhyesti sovelluksen toiminnallisuuden kannalta tärkeiltä osa-alueita.

Gnu Radio sisältää päälohkon, jota tässä työssä käytetään sellaisenaan. Tämä lohko on oma säikeensä, joka alustaa USRP-kortin ja hoitaa sekä vastaanottopuolen että lähetyspuolen lohkojen suorittamisen USRP:n määräämässä tahdissa.

Gnu Radio sisältää erilliset lähetys- ja vastaanottopuolet. Lähetyspuoli lähettää tietoa USRP:lle ja vastaanottopuoli lukee USRP:ltä tulevaa tietoa. Gnu Radioon lisättiin yksi itse tehty lähdelohko lähetyspuolelle ja yksi itse tehty sinkkilohko vastaanottopuolelle.

Lähetyspuoli lähettää sovelluksen koodaaman lähetyspulssin USRP:n DA-muuntimen kautta kameran ultraäänilähettimelle. Tämä toteutetaan itse tehdyssä lähdelohkossa.

Vastaanottopuoli lukee kaikunäytteitä USRP:n AD-muuntimen kautta, tekee niille muutamia manipulaatioita, muodostaa tästä datasta sovellukselle pikseleiden raakadataa ja kirjoittaa tämän raakadatan rengaspuskuriin, kun kuvakehyksen verran raakadataa saadaan luettua. Rengaspuskuriin kirjoittamisen jälkeen lähetetään Posix-signaali sovellukselle vastaanoton käynnistämiseksi. Tämä toteutetaan itse tehdyssä sinkkilohkossa.

5 TESTAUS

Sovelluksen testauksen päätavoitteina on määritelmän mukaisten toiminnallisuuksien toiminnan varmistaminen ja signaalitien kulku USRP-kortilta sovellukselle.

Sovelluksen testaus jaettiin neljään vaiheeseen:

- signaalitien testaukseen
- testaukseen ilman AFE-kortteja käyttäen pelkkää USRP-korttia
- testaukseen käyttäen AFE-kortteja
- kuormitustestaukseen.

5.1 Signaalitien testaus

Ensimmäisessä vaiheessa testattiin signaalitien toiminnallisuus. Jotta sovellus pystyisi toimimaan vaatimusten mukaisella tavalla ja muodostamaan kuvaa kaikuluotaimelta saatavasta kaikuinformaatiosta, tuli sen pystyä vastaanottamaan USRP-kortin lähettämää kaikuinformaatiota. Tämä vaatii toimivaa signaalitietä USRP-kortin ja sovelluksen välillä.

Testaus suoritettiin lähettämällä sovellukselta int-tyypin muuttujia Python-skriptin välityksellä Gnu Radion lohkolle. Sovellus tulosti näytölle lähetettävien muuttujien arvot sovelluksen päässä ja myös Gnu Radiossa vastaanotettujen muuttujien arvot. Näiden muuttujien arvojen ollessa samat pystyttiin varmistamaan signaalitien toiminta sovellukselta USRP-kortille päin.

Signaalitien testaus toiseen suuntaan toteutettiin lähettämällä sini-muotoista signaalia USRP-kortille, joka asetettiin toimimaan digitaalisessa signaalin kierrätystilassa. Tämä tarkoittaa sisääntulevan signaalin kierrätystä USRP:n sisäisten rakenteiden läpi ulospäin meneväksi signaaliksi. USRP-kortilta saadut paluuarvot tulostettiin sekä Gnu Radion että sovelluksen puolella ja näiden

arvojen havaittiin noudattavan sini-muotoista käyrää. Näin oli saatu testattua signaalitien kulku molempiin suuntiin ja todettua se toimivaksi.

5.2 Testaus ilman AFE-kortteja

Toisessa vaiheessa keskityttiin kuvanmuodostuksen, videon tallennuksen ja toiston toiminnallisuuksien testaamiseen. Nämä testit suoritettiin ilman AFE-kortteja, sillä tarvittavat mekaniikkapiirit olivat vielä kehitysvaiheessa eikä niitä näin ollen pystynyt käyttämään alkuvaiheen testauksessa. Testaus toteutettiin käyttäen hyväksi USRP-kortin digitaalista signaalin kierrätystä.

Ensimmäiset testit koskivat kuvanmuodostusta ja ne toteutettiin aluksi ilman napakoordinaatistomuunnosta. USRP-kortille lähetettiin kanttiaallon muotoista signaalia. Paluuarvoista muodostettiin kuva, joka muistutti mustavalkoista kohinaa. Muodostunut kuva oli odotusten mukainen.

Seuraavaksi kuvanmuodostusta testattiin napakoordinaatistomuunnoksen kanssa samoilla parametreilla. Nyt muodostettu kuva muistutti ympyräsektoria ja oli mustavalkoista säännöllistä kohinaa, eli juuri sellaista kuin kuvasta odotettiinkin muodostuvan.

Kuvanmuodostuksen oikeasta toiminnallisuudesta ei kuitenkaan tässä vaiheessa vielä voitu olla täysin varmoja, sillä mitään konkreettista esimerkkiä ei ollut siitä, mihin muodostettua kuvaa voisi verrata. Verrataan tilannetta esimerkiksi toimivalla ultraäänikameran prototyypillä otettuun kuvaan testialtaassa olevasta esineestä. Tästä tilanteesta muodostetusta kuvasta näkyisi tietokoneen näytöllä kyseisen esineen kuva ja tästä voisi päätellä kuvanmuodostuksen toimivan moitteettomasti. Tässä vaiheessa kuitenkin käsitys kuvanmuodostuksen toimivuudesta perustui ymmärrykseen sen teoriasta ja saaduista testituloksista.

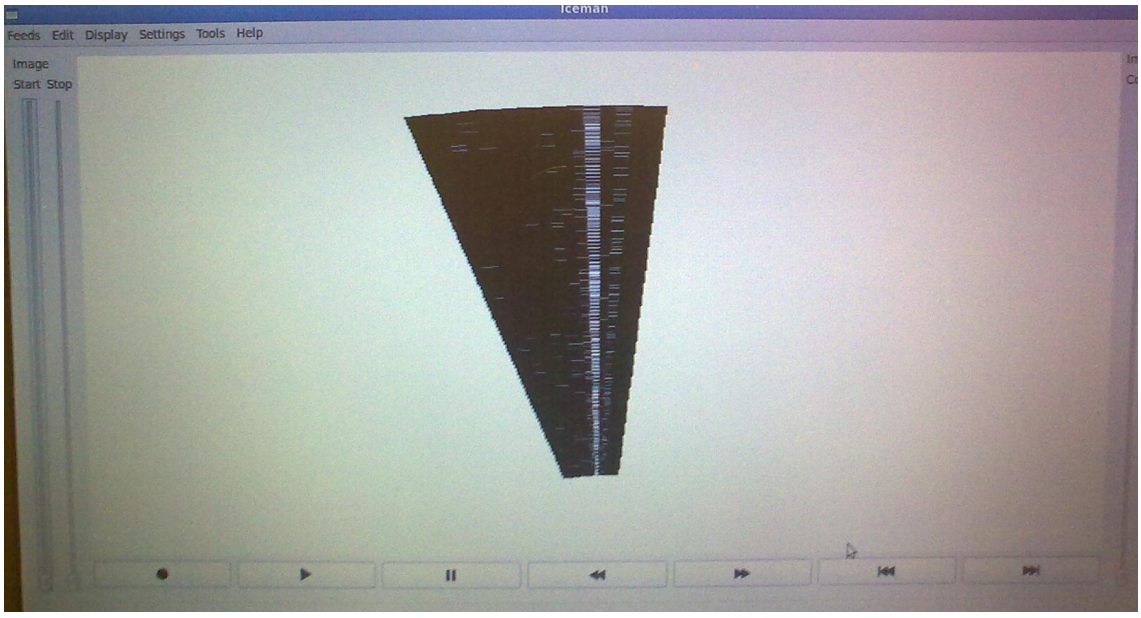
Videon tallennusta ja toistoa testattiin nauhoittamalla muodostettua liikkuvaa kuvaa sovellukseen rakennetulla tallennus-ominaisuudella. Tätä tallennettua

tiedostoa toistettiin sovelluksen toisto-ominaisuudella ja havaittiin toistettavan kuvan olevan täsmälleen samankaltainen kuin tallennuksen kohteena ollut kuva. Tästä pääteltiin tallennuksen ja toiston toimivan moitteetta.

5.3 Testaus käyttäen AFE-kortteja

Kolmannen vaiheen testaus suoritettiin käyttämällä USRP-korttiin kytkettyjä AFE-kortteja.

Testi toteutettiin kytkemällä korttien lähdöt maahan, jolloin sovellukselle saapui vain kohinaista kaikuinformaatiota, josta muodostettu kuva ei sisältänyt mitään jaksottaista informaatiota. Kun yhden AFE-kortin yhden linjan lähtöön kytkettiin jännite, saatiin sovellukselle tulevasta kaikuinformaatiosta muodostetusta kuvasta näkyviin yksi vaalea liuska kohinan keskellä (kuva 11). Testi osoitti, että tämän AFE-kortin linja, eli käytännössä yksi keila, toimii ja siitä vastaanotetusta informaatiosta pystytään muodostamaan liikkuvaa kuvaa, joka on käsitysten mukaan oikeanlaista.



Kuva 11. Kuvaruutukaappaus testitilanteesta.

5.4 Kuormitustestaukset

Neljännen vaiheen testauksessa keskityttiin sovelluksen kuormituksen mittaamiseen. Tavoitteena oli mitata sovelluksen aiheuttama kuormitus prosessorille, löytää sovelluksen osa-alueet, jotka aiheuttavat suurta kuormitusta sekä pyrkiä samalla vähentämään näiden osa-alueiden aiheuttamaa kuormitusta.

Testilaitteena käytetyn tietokoneen tiedot:

- prosessori: Intel Q9550 2.8 GHz (neliytiminen)
- muistia: 6 GB
- käyttöjärjestelmä: Ubuntu 9.10

Kuormitustestien kuormituksen seuranta suoritettiin ajamalla Ubuntun terminaalissa komento "top | grep app2". Kyseinen komento käynnistää top-ohjelman, joka näyttää tietoja käynnissä olevista ohjelmista. Lisäämällä komennon perään "| grep app" saadaan näkyviin vain app2-ohjelman tiedot. Tässä tapauksessa app2 on sovelluksen kehitysnimi. Näkyviin tulostuvissa tiedoissa on muun muassa ohjelman prosessorikuormitus.

Sovellus pyörittää taustalla Gnu Radiota ja sen aiheuttama kuormitus näkyy top-komennolla yhtenä tämän sovelluksen itsensä aiheuttaman kuormituksen kanssa.

Jokaisessa testauksessa sovellus käynnistettiin ja sillä otettiin yhteyttä USRP-korttiin. Sovellusta ajettiin 60 sekunnin ajan, jonka jälkeen laskettiin sovelluksen aiheuttaman keskimääräisen prosessorikuormituksen aste.

Ensimmäinen kuormitustesti sisälsi pelkästään rengaspuskurista lukemisen ilman muita toiminnallisuuksia, kuten kuvanmuodostusta tai kuvan näytölle piirtämistä. Tässä testissä keskimääräiseksi kuormitusasteeksi saatiin 21%.

Toinen testi suoritettiin kaikki sovelluksen toiminnallisuudet mukaan ottaen. Keskimääräiseksi kuormitusasteeksi saatiin 24%.

Kolmas testi sisälsi muutoin kaikki toiminnallisuudet paitsi näytölle piirtämistä ei otettu mukaan. Tällä kertaa keskimääräiseksi kuormitusasteeksi saatiin 22%.

Näiden kolmen testin tulokset olivat hyvin lähellä toisiaan. Pelkkä perustoiminnallisuus, eli Gnu Radion pyörittäminen ja rengaspuskurista kaikuinformaation lukeminen kuormittaa prosessoria hyvinkin paljon.

Sovelluksen muiden toimintojen aiheuttama lisäkuormitus oli hyvin vähäistä tai olematonta verrattuna perustoiminnallisuuteen. Tämän johdosta ei ollut tarpeen suorittaa yksityiskohtaisempia rasitustestejä sovelluksen eri osa-alueille.

Rasitustestien johtopäätöksenä voidaan todeta sovelluksen aiheuttaman prosessorikuormituksen olevan suurta, etenkin ottaen huomioon testilaitteen prosessorin neliytimisyyden, mutta lisätoimintojen kuormitus ei vaikuta juuri lainkaan perustoiminnallisuuden aiheuttamaan kuormitukseen. Sovelluksen rakenne ja ohjelmointi ovat siis onnistuneita ja huolimatta useista säikeistä sekä runsaasta laskennasta eivät sovelluksen toiminnallisuudet aiheuta mainittavaa kuormitusta.

6 JATKOKEHITYSMAHDOLLISUUDET

Jatkokehittelyssä sovelluksen osalta voidaan tehostaa kuvanmuodostusta ja kuvan näytölle piirtämistä esimerkiksi siirtämällä koko prosessi näytönohjaimen prosessorien hoidettavaksi. Tämä nopeuttaisi prosessia huomattavasti ja säästäisi tietokoneen oman prosessorin tehoja. Haittapuolina voidaan mainita kyseisen rakenteen vaatima monimutkaisempi koodi ja rautatuki näytönohjaimelta.

Python-rajapinta on myös mahdollista korvata C++:llä toteutetulla välikerroksella, jota voidaan kutsua suoraan sovellukseltä käsin ilman, että tarvitaan erillisiä Python-skriptejä. Tämä toteutus mahdollistaisi yksinkertaisemman kokonaisrakenteen, poistaisi Pythoniin liittyvät epävarmuustekijät ja nopeuttaisi kokonaisprosessointiaikaa, kun Gnu Radiota voidaan kutsua suoraan sovellukselta käsin.

Lienee myös mahdollista korvata Gnu Radio itse tehdyllä rakenteella. Tämä rakenne kommunikoisi suoraan sovelluksen ja USRP-kortin välillä ja vähentäisi näin sovelluksen riippuvuutta kolmannen osapuolen ohjelmista. Tällä saavutettaisiin myös parempi alustariippumattomuus sekä yksinkertaisempi ohjelmistoasennus.

Sovellukseen voidaan myös lisätä uusia ominaisuuksia, joita ei vielä tässä prototyypin versiossa ole olemassa. Tällaisia ominaisuuksia olisivat muun muassa kaikuluotaimen käyttö verkon ylitse, videon tallennus ja toisto pakattuna formaattina, mahdollisuus useamman kameran yhtäaikaiseen käyttöön ja kaikuluotaimen kytkemisen tunnistus.

7 YHTEENVETO

Tätä opinnäytetyötä tehdessäni sain tuntuman alkuvaiheessaan olevan yrityksen toimintaan, johtamiseen ja ongelmiin. Ymmärrän nyt myös paremmin uuden tuotteen kehittämiseen liittyviä haasteita ja ohjelmiston kehittämisen vaikeuksia.

Opinnäytetyön tuloksena kehitetty sovellus on toteutettu alkuperäisten määritysten mukaisesti. Kaikki sovellukselle asetetut tavoitteet saavutettiin ja sovellus sisältää muutaman ylimääräisen ominaisuuden. Sovellus pystyy kommunikoimaan ultraäänikameran prototyypin kanssa ongelmitta ja myös sen keskeinen ominaisuus, kuvanmuodostus kaikuinformaatiosta ja sen toistaminen näytöllä, toimii sujuvasti.

Kuormitustestien perusteella voidaan sanoa sovelluksen toiminnallisuuksien aiheuttavan vain vähäistä kuormitusta tietokoneelle. Suurimman kuormituksen sovelluksessa aiheuttaa Gnu Radion pyörittäminen taustalla.

Sovellus itsessään on alustariippumaton ja toimii sekä Windows- että Linux-käyttöjärjestelmissä. Gnu Radion huono toimivuus Windows-käyttöjärjestelmän kanssa estää sovelluksen tehokkaan kehittämisen ja käytön Windows-ympäristössä. Sovelluksen seuraavien versioiden tulee kuitenkin pystyä toimimaan myös Windows-ympäristössä, joten Gnu Radiosta pitää tulla julki toimiva versio Windows-käyttöjärjestelmässä tai sitten siitä pitää päästä kokonaan eroon esimerkiksi kirjoittamalla itse sen korvaava rakenne.

Kehitetty sovellus on käytössä päivittäin Simsonar Oy:n toiminassa ja toimii hyvänä pohjana tulevaisuuden sovellusversioita kehitettäessä. Yritys pystyy esittelemään ultraäänikameran prototyypin toimintaa ja sen tuottamaa kuvaa potentiaalisille tuleville asiakkaille käyttäen tämän opinnäytetyön tuloksena syntynyttä sovellusta.

LÄHDELUETTELO

Blossom, E. 2004. Exploring GNU Radio. Hakupäivä 9.9.2010
http://www.gnu.org/software/gnuradio/doc/exploring-gnuradio.html

Ettus Research LLC, Ettus Research LLC. 2010. Hakupäivä 9.9.2010
http://www.ettus.com/

Liite 1, Ultraäänikameran kokonaisjärjestelmä. Pertti Seppänen. 2010

Liite 2, Ultraäänikameran toiminnallisuuden periaate. Pertti Seppänen. 2010

Liite 3, Kuvanmuodostus-funktion ohjelmakoodi. 2010

Liite 4: Python-skriptin kutsuminen. 2010

Napakoordinaatisto, Wikipedia. 2010. Hakupäivä 8.9.2010
http://fi.wikipedia.org/wiki/Napakoordinaatisto

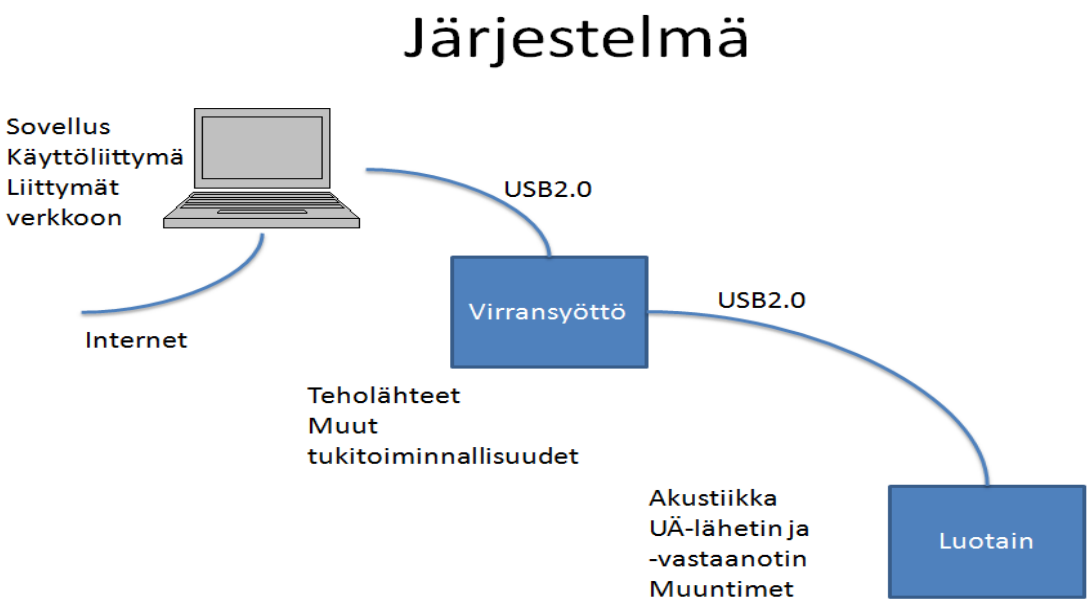
Qt, Nokia Oyj. 2010. Hakupäivä 11.9.2010
http://qt.nokia.com

USRP Motherboard, GNU Radio. 2010. Hakupäivä 10.9.2010
http://gnuradio.org/redmine/wiki/gnuradio/UsrpFAQIntroMobo

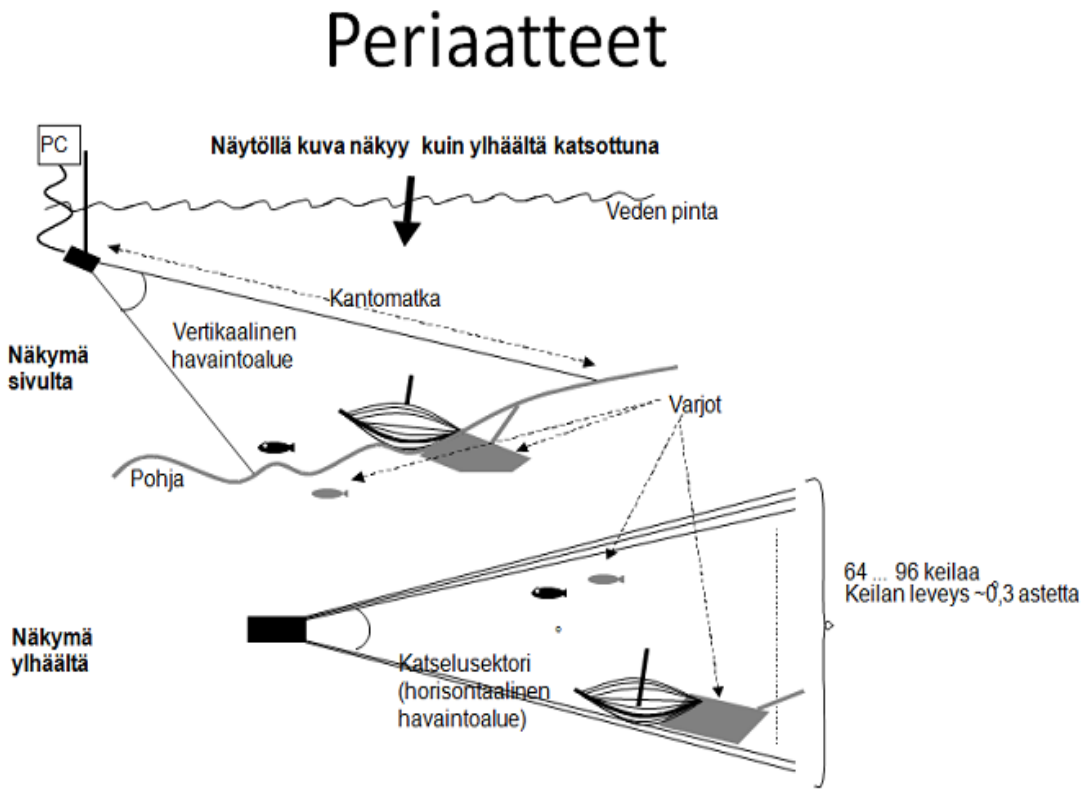
Welcome to GNU Radio, GNU Radio. 2010. Hakupäivä 11.9.2010
http://gnuradio.org/redmine/wiki/gnuradio

LIITTEET

LIITE 1: Ultraäänikameran kokonaisjärjestelmä



LIITE 2: Ultraäänikameran toiminnallisuuden periaate



LIITE 3: Kuvanmuodostus-funktion ohjelmakoodi

```
void PixelMapper::mapPixs(uchar *buffer, uchar *imageBuffer, uchar
*diagnosticsDistanceBuffer, int p_iRows, int p_iColumns, int p_iImageRows, int
p_iImageColumns, int p_iAFE)
{
  this->iColumns = p_iColumns;
  this->iRows = p_iRows;
  this->iImageColumns = p_iImageColumns;
  this->iImageRows = p_iImageRows;

  // Kopioidaan kuvabuffer dynaamiseen taulukkoon
  int e = 0;
  for(int j=0; j<this->iRows; j++)
    for(int i=0; i<this->iColumns; i++)
        this->daTempArray[j][i] =  buffer[e++];

  // Jos tallennus -> Lähetetään taulukko fileSaverille
  if (this->bRecordStatus == true)
  {
    fileSaver->writeFrameHeader();
    fileSaver->writeFrameData(this->daTempArray);
  }

  // Pikselien asetus oikeille paikoille keilojen lukumäärän mukaisesti
  switch(this->iColumns)
  {
    case 32:
    {
      Pixel32();
      break;
    }
    case 64:
    {
      Pixel64();
      break;
    }
    case 96:
    {
      Pixel96();
      break;
    }
    default:
    {
      break;
    }
  }
```

```
  int iX = 0;
  int iY = 0;
  int e = 0;

  for(int j=0; j<this->iRows; j++)
  {
    int n = 0;
    for(int i=0; i<this->i96Beams96Columns; i++)
    {
      if (this->iColumns == 32 || this->iColumns == 64)
      {
        if (i > 15 && i < 79)
        {
          // Haetaan koordinaattien arvot
          if (this->iColumns == 32)
          {
            iX = this->adXValues32[j][n];
            iY = this->adYValues32[j][n];
          }
          // Haetaan koordinaattien arvot
          if (this->iColumns == 64)
          {
            iX = this->adXValues64[j][n];
            iY = this->adYValues64[j][n];
          }
          n++;
          // Kopioidaan kuvadata kuvataulukkoon lasketuille paikoille
          for (int k=0; k<7; k++)
          {
            this->daImageData[this->iImageRows-1-iY][(this-
>iImageColumns/2)+(iX+k)] = this->daData[j][i];
          }
        }
      }
      // 96 keilan konfiguraatiolla suoritetaan vastaavat operaatiot
      else
      {
        iX = this->adXValues96[j][i];
        iY = this->adYValues96[j][i];

        for (int k=0; k<7; k++)
        {
          this->daImageData[this->iImageRows-1-iY][(this-
>iImageColumns/2)+(iX+k)] = this->daData[j][i];
        }
      }
      e = 0;
      for(int j=0; j<this->iImageRows; j++)
        for(int i=0; i<this->iImageColumns; i++)
```

```
            imageBuffer[e++] = this->daImageData[j][i];
          }
        }
      }
```

LIITE 4: Python-skriptin kutsuminen

```
void getPythonReady::startPython()
{
  printf("startPython\n");
  pArgs = PyTuple_New(10);

  PyObject *testPyFreq = PyFloat_FromDouble(this->defaultParams->testFreq);
  PyObject *testPyInterpolation = PyInt_FromLong(this->defaultParams->testInterpolation);
  PyObject *testPyDecimation = PyInt_FromLong(this->defaultParams->testDecimation);
  PyObject *testPyAFEPower = PyInt_FromLong(this->defaultParams->testAFEPower);
  PyObject *testPyAFESamples = PyInt_FromLong(this->defaultParams->testAFESamples);
  PyObject *testPyRxPga = PyFloat_FromDouble(this->defaultParams->testRxPga);
  PyObject *testPyTxPga = PyFloat_FromDouble(this->defaultParams->testTxPga);
  PyObject *testPyAfeCount = PyInt_FromLong(this->defaultParams->testAfeCount);

  pName = PyUnicode_FromString(this->pyTarget);
  // Asetetaan Python-skriptin moduulin nimi
  pModule = PyImport_Import(pName);
  unsigned long trickBuffer = (unsigned long)this->cBufferAddress;
  pBuff = PyLong_FromUnsignedLong(trickBuffer);
  unsigned long trickTxHeader = (unsigned long)this->txDataToPy;
  pTx = PyLong_FromUnsignedLong(trickTxHeader);
  // Asetetaan funktion nimi
  pFunc = PyObject_GetAttrString(pModule, this->pyFunctionName);
  pLen = PyInt_FromLong(this->iActualLength);
  // Asetetaan parametrit
  PyTuple_SetItem(pArgs, 0, pBuff);
  PyTuple_SetItem(pArgs, 1, pTx);
  PyTuple_SetItem(pArgs, 2, testPyFreq);
  PyTuple_SetItem(pArgs, 3, testPyInterpolation);
  PyTuple_SetItem(pArgs, 4, testPyDecimation);
  PyTuple_SetItem(pArgs, 5, testPyAFEPower);
  PyTuple_SetItem(pArgs, 6, testPyAFESamples);
  PyTuple_SetItem(pArgs, 7, testPyRxPga);
  PyTuple_SetItem(pArgs, 8, testPyTxPga);
```

```
    PyTuple_SetItem(pArgs, 9, testPyAfeCount);
    // Kutsutaan Python-funktiota funktion nimellä ja argumenteilla
    pValue = PyObject_CallObject(pFunc, pArgs);

    if (pValue == NULL)
        printf("null pValue\n");
    Py_DECREF(testPyFreq);
    Py_DECREF(testPyInterpolation);
    Py_DECREF(testPyDecimation);
    Py_DECREF(testPyAFEPower);
    Py_DECREF(testPyAFESamples);
    Py_DECREF(testPyRxPga);
    Py_DECREF(testPyTxPga);
    Py_DECREF(testPyAfeCount);
    Py_DECREF(pName);
    Py_DECREF(pModule);
    Py_DECREF(pFunc);
    Py_DECREF(pArgs);
    Py_DECREF(pTx);
    Py_DECREF(pBuff);
    Py_DECREF(pValue);
}
```