OULUN AMMATTIKORKEAKOULU

Matti Rekis

DOPPLER-SUUNTIMA-ALGORITMI

DOPPLER-SUUNTIMA-ALGORITMI

Matti Rekis
Opinnäytetyö
Kevät 2014
Tietotekniikan koulutusohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, langattomat laitteet

Tekijä: Matti Rekis
Opinnäytetyön nimi: Doppler-suuntima-algoritmi
Työn ohjaaja: Veijo Korhonen
Työn valmistumislukukausi ja -vuosi: Kevät 2014
Sivumäärä: 69 + 7 liitettä

Opinnäytetyön tavoitteena oli suunnitella ja kehittää suuntima-algoritmi, jolla voitaisiin laskea doppler-ilmiön perusteella radiolähettimen suunta eli kulma, jossa lähetin sijaitsee vastaanottimen kulkusuuntaan nähden, sekä valmistaa osoitinnäyttö, joka osoittaa lähettimen suunnan. Algoritmi laskee lähettimen sijainnin selvittämällä ensin, liikkuuko vastaanotin sitä kohti vai siitä poispäin, ja sitten vertaamalla vastaanottimen liikkeessä saadun taajuusarvon doppler-vääristymää laskettuun teoreettiseen doppler-vääristymään, jossa vastaanotin kulkisi suoraan lähetintä kohti. Algoritmi laskee kulman lähettimen suunnalle vain silloin, kun kuljetaan lähetintä kohti. Muuten se ilmoittaa vain, että kuljetaan väärään suuntaan. Suuntima-algoritmin ohjelmointi on toteutettu C-kielellä Arduino Due -mikrokontrollerikortille.

Doppler-suuntima-algoritmiohjelmasta on kaksi eri versioita. Toinen on spesifinen superheterodyne-vastaanottimelle ja toinen on teoreettisempi, määrittelemättömälle vastaanottimelle suunniteltu versio.

Doppler-suuntima-algoritmi ja suunnan osoitinnäyttö testattiin sekä ohjelmallisesti algoritmin muuttujille arvoja asettamalla että Oulun ammattikorkeakoulun laboratoriossa funktiogeneraattorilla, joka simuloi radiolähettimen signaalia. Laboratiotestauksessa ilmeni, että algoritmin taajuuslaskuri on sen verran epätarkka ja heittelevä, että funktiogeneraattorilla saaduissa mittaustuloksissa ilmeni jonkin verran vaihtelua ja epätarkkuutta. Algoritmin superheterodyne-versio antoi kuitenkin melko tarkkoja tuloksia myös funktiogeneraattorilla testattaessa. Ohjelmallisesti testattaessa algoritmin toiminta ja mittaustulokset saatiin käytyä tarkasti läpi. Doppler-suuntima-algoritmi ja osoitinnäyttö toimivat odotetulla tavalla.

Asiasanat: doppler-ilmiö, suuntima-algoritmi, taajuus, mittaus, paikannus, superheterodyne-vastaanotin, C-kieli

ABSTRACT

Oulu University of Applied Sciences
Information Technology, wireless devices

Author: Matti Rekis
Title of thesis: Doppler bearing algorithm
Supervisor: Veijo Korhonen
Term and year when the thesis was submitted: Spring 2014
Pages: 69 + 7 appendices

The goal of this final thesis was to design and develope a bearing algorithm that can be used to calculate the direction of a radio transmitter by utilizing the doppler effect and to make a pointer display that shows the direction. The algorithm calculates the angle between the receivers moving direction and the transmitter, by firstly resolving if the receiver is moving towards or backwards from the transmitters position and then compairing the doppler shift of the frequency value of the receiver when its moving, to a theoretical doppler shift where the receiver would be moving straight towards the transmitter. The algorithm calculates the angle only when the receiver is moving towards the transmitter, otherwise it just reports that the receiver is moving to the wrong direction. The bearing algorithm was programmed using C language to a Arduino Due board.

There is two versions of the doppler bearing algorithm program. One is specific for a superheterodyne receiver and the other one is more theoretical and designed to unspecified receiver, that can detect the doppler shift from the original transmitted signal without converting it to intermediate frequency.

The doppler bearing algorithm and the pointer display was tested programmatically by giving values to the attributes of the program and in a laboratory of Oulu University of Applied Sciences with a function generator that simulated the transmitters signal. The laboratory testing showed that the algorithms frequency counter is somewhat imprecise and there were some variation in the results of the frequency measurements with the function generator. However, the programmatically performed tests gave very accurate measurement results of the operation of the algorithm. The superheterodyne version of the algoritm gave pretty accurate results and worked quite well also with function generator tests. The doppler bearing algorithm and the pointer display worked as planned.

Keywords: doppler effect, bearing algorithm, frequency, measurement, positioning, superheterodyne receiver, C language

SISÄLLYS

SANASTO

AD-muunnin	Analogia-digitaalimuunnin
ARM	Advanced RISC Machines, 32-bittinen mikroprosessoriarkkitehtuuri
BGA	Ball Grid Array, piirin kotelotyyppi
CAN	Controller Area Network, automaatioväylä, jota käytetään mm. ajoneuvoissa
DA-muunnin	Digitaali-analogiamuunnin
$\overline{G}$	Output Enable Input, lähtöjen sallinta
Gerber-formaatti	Tiedostomuoto, jota käytetään piirilevysuunnitteluohjelmistoissa kuvaamaan piirilevyä
GND	Ground (0 V), maajohdin
HS USB	High Speed Universal Serial Bus, sarjaväyläarkkitehtuuri oheislaitteille
Hz	Hertsi, 1/s
I2C	Inter-Integrated Circuit, kaksisuuntainen ohjaus- ja tiedonsiirtoväylä
I/O	Input/Output, tulo/lähtö
ISM-taajuusalue	Industrial, Scientific and Medical, maailmanlaajuinen radiotaajuuskaista, jonka käyttö ei vaadi erillistä lupaa
JTAG	Joint Test Action Group, testaus- ja debuggausliittymä
LED	Light-Emitting Diode, ledi
MMC	MultiMediaCard, Flash-muistikorttityyppi

OSI	Open Systems Interconnection model, malli, joka kuvaa tiedonsiirtoprotokollien yhdistelmän seitsemässä kerroksessa
PC	Personal Computer, henkilökohtainen tietokone
PHY	Physical Layer, OSI-mallin fyysinen kerros
PSRAM	Pseudo-Static Dynamic Random Access Memory, muistityyppi, joka tallentaa jokaisen bitin omaan erilliseen kondesaattoriin
PWM	Pulse-Width Modulation, pulssinleveysmodulaatio
QFP	Quad Flat Package, piirin kotelotyyppi
RCK	Storage Register Clock Input, siirtorekisteripiirin varastorekisterin kellotulo
RDF	Radio Direction Finder, radiosuuntimalaite
RTC	Real-Time Clock, kello, joka pitää tietokoneen oikeassa ajassa
SCK	Shift Register Clock Input, siirtorekisterin kellotulo
SD	Secure Digital, muistikorttityyppi
SDIO	Secure Digital Input Output, muistikorttityyppi
SER	Serial Data Input, sarjamuotoisen datan sisääntulo
Serial Monitor	Arduino-ohjelmistoon integroitu ikkuna, jonka avulla Arduino-kortti voi kommunikoida mm. PC:n kanssa USB-liitännän avulla
SI	Serial Input, sarjamuotoisen datan sisääntulo
SPI	Serial Peripheral Interface, sarjamuotoinen oheislaiteväylä

SRAM	Static Random Access Memory, muistityyppi, jonka muistisolu koostuu kiikkupiiristä
TWI	Two Wire Interface, kaksisuuntainen ohjaus- ja tiedonsiirtoväylä
UART	Universal Asynchronous Receiver Transmitter, sarjaliikennepiiri, joka muuntaa rinnakkaismuotoista tietoa sarjamuotoiseksi ja päinvastoin
UHF	Ultra High Frequency, 0,3–3 GHz:n taajuusalue
USART	Universal Synchronous/Asynchronous Receiver Transmitter, sarjaliikenteen lähetys- ja vastaanottopiiri, joka muuntaa rinnakkaisdatan sarjadataksi
USB	Universal Serial Bus, tietokoneen sarjaväylätyyppi
VCC	Positive Supply Voltage, käyttöjännite
VHF	Very High Frequency, 30–300 MHz:n taajuusalue

1 JOHDANTO

Doppler-ilmiön selitti ensimmäisen kerran Itävaltalainen fyysikko ja matemaatikko Johann Christian Andreas Doppler (29.11.1803–17.03.1853) vuonna 1842. Doppler-ilmiö muodostuu aaltojen lähteen ja havaitsijan välisestä suhteellisesta liikkeestä, joka aiheuttaa aaltoliikkeen taajuudessa, aallonpituudessa ja vaiheessa näennäistä muutosta. (2.)

Havaitsijan liikkuessa aaltolähdettä kohti siihen törmää aaltoja useammin kuin paikalla ollessaan ja havaitsijan etääntyessä lähteestä saapuvat aallot harvemmin – taajuus siis kasvaa mentäessä kohti ja alenee siirryttäessä kauemmas lähteestä. Kummassakin tapauksessa havaitsija havaitsee signaalin eri taajuudella kuin mikä on lähteen todellinen lähetystaajuus. Doppler-ilmiö voidaan todeta mm. äänen, valon, radion tai vaikkapa veden aalloista, eli toisin sanoen yhtälöt ilmiön laskemiseen ovat samat missä tahansa rajapinnassa, jossa tapahtuu suhteellista liikettä havaitsijan ja aaltojen lähteen välillä. (2.)

Doppler-ilmiön aiheuttaman siirtymän suuruuteen ja suuntaan vaikuttaa se, kasvaako vai pieneneekö havaitsijan ja lähteen välinen etäisyys sekä niiden välinen suhteellinen nopeus. Ilmiötä voidaan hyödyntää esimerkiksi radiolähettimen paikantamiseen radiovastaanottimen ollessa liikkeessä.

Tämän opinnäytetyön aiheena on suunnitella ja toteuttaa suuntima-algoritmi sekä suunnan osoitinnäyttö ”Doppler-kompassille”, jolla voidaan liikkeessä paikantaa radiolähetin. Doppler-kompasilla tässä opinnäytetyössä tarkoitetaan kokonaisuutta, joka sisältää doppler-suuntima-algoritmiohjelman ja osoitinnäytön lisäksi vastaanottimen, jolla voidaan langattomasti havaita doppler-ilmiön aiheuttama taajuuden muutos esimerkiksi kännyköiden hyödyntämästä UHF-taajuusalueesta. Työssä on tehty myös oma versio suuntima-algoritmiohjelmasta spesifisesti 2,4 GHz:n taajuutta 100 Hz:n välitaajuudelle muuntavalle superheterodyne-vastaanottimelle. Tässä opinnäytetyössä ei oteta huomioon laitteen vastaanottimen vaatimuksia.

2 DOPPLER-ILMIÖ JA PAIKANNUS

Tässä luvussa tutustutaan doppler-ilmiöön sekä sen hyödyntämiseen radiolähettimen suunnan laskentaan. Doppler-suuntima-algoritmi hyödyntää kaavoja 2 ja 4. Kaava 2 laskee algoritmissa doppler-siirtymän arvon, kun liikutaan täsmälleen lähetintä kohti, ja kaava 4 laskee kulman lähettimen suunnalle vastaanottimen liikesuunnassa. Doppler-ilmiö muodostuu lähettimen ja vastaanottimen välisestä suhteellisesta nopeudesta. Näin ollen ei ole väliä, liikkuuko lähetin vai vastaanotin vai molemmat, mutta työssä suunnitellussa suuntima-algoritmissa lähettimen on oletettu pysyvän paikallaan vastaanottimen liikkuessa. Luvun lopussa myös hieman historiaa, lähinnä englantilaisten näkökulmasta, radiosuuntiman ja radiopaikannuksen alkuvaiheista sekä niiden hyödyntämisestä erilaisiin tarkoituksiin.

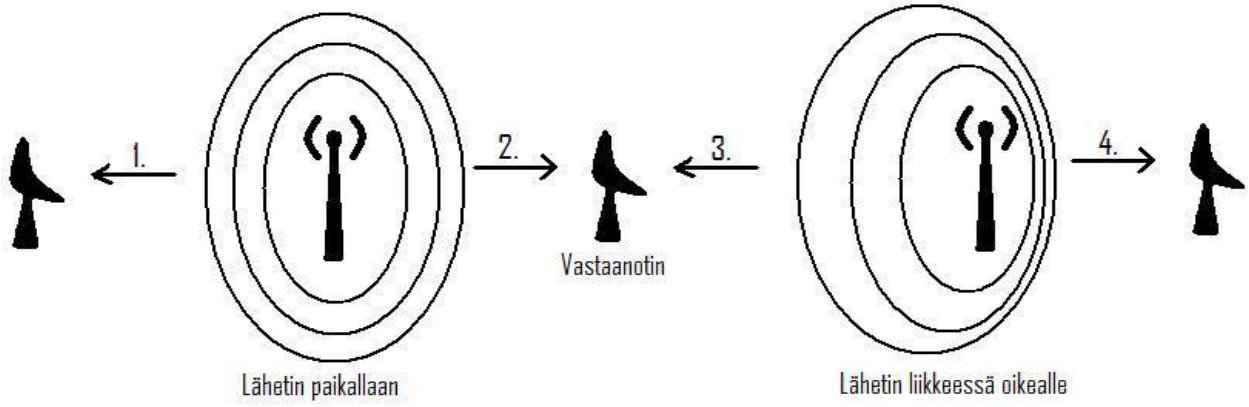
2.1 Doppler-ilmiö ja suunnan laskenta

Doppler-ilmiö tarkoittaa taajuuden näennäistä muuttumista, kun säteilyn lähde ja havaitsija, tai esimerkiksi radioliikenteessä lähetin ja vastaanotin, liikkuvat toistensa suhteen. Ihmiselle ehkä tutuin havainto ilmiöstä on hälytysajoneuvon sireeni. Kun auto lähestyy, äänen aallonpituus lyhenee eli taajuus kasvaa ja sireenin ääni kuulostaa todellista korkeammalta. Auton ohittaessa havaitsijan voi helposti kuulla, kuinka ääni muuttuu matalammaksi taajuuden pienentyessä. Samaan tapaan myös sähkömagneettisen säteilyn taajuus muuttuu, jos säteilylähde liikkuu havaitsijan suhteen. Sähkömagneettinen säteily (näkyvä valo, radioaallot jne.) etenee aina valon nopeudella riippumatta lähteen tai havaitsijan liikkeestä. Tästä seuraa, että kaava 1 pätee riippumatta siitä, liikkuuko lähde vai havaitsija. Äänen tapauksessa tilanne on hieman monimutkaisempi. (1.)

$$\frac{f_d - f_0}{f_0} = \frac{v}{c}$$ *KAAVA 1*

f_d = havaittu taajuus / vastaanotettu taajuus
f_0 = lepotaajuus / alkuperäinen lähetetty taajuus
c = valonnopeus
v = kohteen tai havaitsijan nopeus

Doppler-ilmiötä voidaan havainnollistaa esimerkiksi radiolähettimen ja radiovastaanottimen avulla. Oletetaan, että kuvan lähetin lähettää radioaaltoja tasaisesti 2,4 GHz:n taajuudella vastaanottimen pysyessä aina paikallaan. Kuvassa vasemmalla lähetin on paikallaan ja oikealla se on liikkeessä oikealle päin nopeudella 50 km/h. Kuvan lähetin ja vastaanotin ovat aina kohtisuorassa toisiinsa. (Kuva 1.)



KUVA 1. Vastaanottimen havaitsema doppler-siirtymä lähettimen liikkuessa (mukaillen 3, s. 8.)

Kuvan 1 kohdissa 1 ja 2 vastaanotin ja lähetin ovat molemmat paikoillaan. Vastaanotin havaitsee lähettimen signaalin 2,4 GHz:n taajuudella eli sen muuttumattomana arvona, sellaisena kuin se on lähettimestä lähtenytkin. Kohdassa 3 vastaanotin havaitsee taajuuden noin 111 Hz todellista lähetystaajuutta matalampana kaavan 3 mukaan, sillä lähetin liikkuu nopeudella 50 km/h vastaanottimesta poispäin. Kohdan 4 tilanteessa lähettimen liikkuessa vastaanotinta kohti on vastaanotettu taajuus noin 111 Hz korkeampi kuin lähetetty taajuus, kaavan 2 mukaisesti. Doppler-siirtymän magnitudi riippuu radiolähettimen signaalin taajuudesta ja nopeudesta sekä siitä tuleeko se kohti vai poispäin vastaanottimesta. Kuvassa 1 näkyvät ympyrät kuvaavat radioaaltoja.

$$f_d = f_0 * \left(\frac{c+v}{c}\right) \qquad \textit{KAAVA 2}$$

$f_d =$ vastaanotettu taajuus
$f_0 =$ lähetetty taajuus

$c =$ valonnopeus
$v =$ lähettimen nopeus (vastaanotin pysyy paikoillaan)

Kaavasta 2 saadaan laskettua suoraan kohti ja kaavasta 3 suoraan poispäin liikkuvan radiolähettimen teoreettinen doppler-siirtymä. Kaavat toimivat riippumatta siitä, onko liikkuva komponentti lähetin vai vastaanotin, kunhan toinen pysyy paikallaan. Kaavat 2 ja 3 on muokattu doppler-ilmiön perusyhtälöstä. (2.)

$$f_d = f_0 * \left(\frac{c-v}{c}\right) \qquad \textit{KAAVA 3}$$

$f_d =$ vastaanotettu taajuus
$f_0 =$ lähetetty taajuus
$c =$ valonnopeus
$v =$ lähettimen nopeus (vastaanotin pysyy paikoillaan)

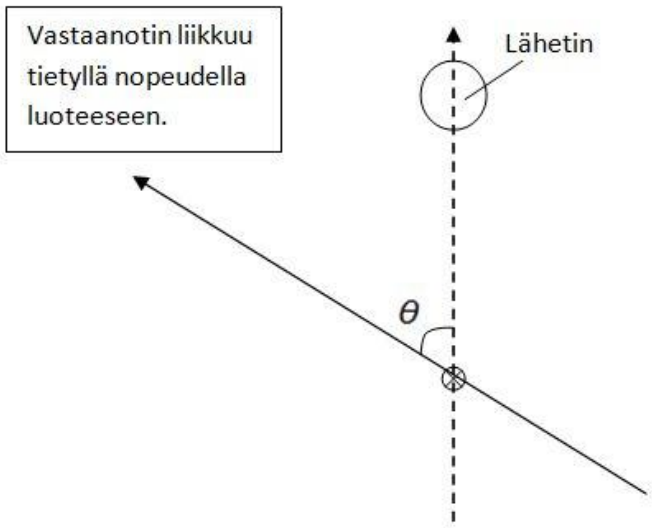
Kaavat 2 ja 3 toimivat ainoastaan silloin, kun lähettimen ja vastaanottimen välinen liike on kohtisuoraista. Jos esimerkiksi vastaanotin liikkuu kulmassa lähetintä kohti, lähettimen ollessa paikoillaan, pienenee doppler-taajuus $\cos(\theta)$ verran. θ on kulma vastaanottimen kulkusuunnan ja lähettimen suunnan välillä. Jos vastaanottimen kulkunopeus on tiedossa, voidaan doppler-taajuutta hyödyntäen laskea kulma, missä lähetin sijaitsee vastaanottimen kulkusuuntaan nähden. Kulma θ voidaan laskea kaavalla 4 tai kaavalla 5. Lähettimen suunta selviää, kun lasketaan teoreettinen doppler-ilmiön aiheuttama kulkunopeus sille, jos vastaanotin menisi suoraan lähetintä kohti, ja jaetaan se vastaanottimen todellisella kulkunopeudella tai vaihtoehtoisesti lasketaan teoreettinen doppler-taajuus, kun vastaanotin kulkee suoraan lähetintä kohti, ja jaetaan sillä vastaanottimen kulkusuunnan todellinen doppler-taajuus. (3, s. 11.) (Kuva 2.)

$$\theta = \arccos\left(\frac{\Delta f_m}{\Delta f_t}\right) \qquad \textit{KAAVA 4}$$

$\theta =$ kulma vastaanottimen kulkusuunnan ja lähettimen välillä
$\Delta f_m =$ lähettimen taajuuden doppler-vääristymä vastaanottimen liikkuessa
$\Delta f_t =$ teoreettinen lähettimen taajuuden doppler-vääristymä, mikäli vastaanotin kulkisi tietyllä nopeudella suoraan lähetintä kohti

$$\theta = \arccos\left(\frac{v_d}{v}\right)$$ *KAAVA 5*

$\theta =$ kulma vastaanottimen kulkusuunnan ja lähettimen välillä
$v_d =$ doppler-ilmiön aiheuttamasta taajuudenmuutoksesta laskettu teoreettinen kulkunopeus, jos vastaanotin kulkisi suoraan lähetintä kohti
$v =$ vastaanottimen kulkunopeus



KUVA 2. Lähettimen suunnan paikantaminen taajuuspoikkeaman kulman avulla

2.2 Radiosuuntiman historiaa ja paikannusteknologioita

Radiosuuntimateknologia (RDF) (engl. radio direction finding) tarkoittaa vastaanotetun signaalin saapumissuunnan mittausta. Tällä voidaan tarkoittaa radiotekniikkaa tai jotain muuta langatonta tiedonsiirtotekniikkaa. Yhdistämällä suuntainformaatio kahdesta tai useammasta vataanottimesta, jotka on asetettu sopiviin paikkoihin, voidaan signaalin lähde paikantaa kolmiomittauksen avulla. Paikallaan olevan lähettimen sijainti voidaan myös selvittää yksittäisellä liikkuvalla vastaanottimella, kun tiedetään vastaanottimen kulkunopeus, sillä kolmen tunnetun pisteen doppler-taajuuksista voidaan laskea nopeus-, suunta- ja paikkatiedot. Radiosuuntimateknologioita hyödynnetään mm. laivojen ja lentokoneiden navigaatiossa, hätäsignaalien ja -lähetinten paikantamisessa, villieläinten seurannassa ja laittomien tai häiritsevien lähetinten paikantamisessa. (4.)

RDF-järjestelmiä voidaan käyttää mihin tahansa radiolähteeseen. Vastaanottimen antennin koko toimii funktiona signaalin aallonpituudelle – todella pitkät aallonpituudet eli hyvin matalat taajuudet vaativat erittäin suuria antenneja. Näitä matalan taajuuden kookkaita antenneja käytetään yleisesti maalla olevissa järjestelmissä, mutta ne ovat myös erittäin hyödyllisiä laivojen navigoinnissa, koska matalat taajuudet voivat kulkea pitkiä matkoja horisonttiin ja näkyvyys merellä saattaa olla vain muutamia kymmeniä kilometrejä. Ilmailukäytössä, missä näkyvyys voi ulottua satoihin kilometreihin, voidaan käyttää korkeampia taajuuksia, mikä puolestaan mahdollistaa paljon pienempien antennien käytön. (4.)

Sotateollisuudessa RDF toimii avainkomponenttina signaalitiedustelussa ja metologiassa. Vihollisen lähettimen paikantaminen on ensimmäisestä maailmansodasta lähtien ollut korvaamatonta sodankäynnissä ja esimerkiksi toisen maailmansodan Atlantin sotatantereella radiopaikantaminen toimi avaintekijänä meritaisteluiden lopputuloksissa. On arvioitu, että Britannian huff-duff-järjestelmät, jotka olivat toisen maailmansodan kehittyneimpiä RDF-järjestelmiä, olivat suorasti tai epäsuorasti vastuussa 24 %:sta kaikista upotetuista sukellusveneistä sodan aikana. Nykyaikaisissa järjestelmissä käytetään yleisesti vaiheohjattuja antenniryhmiä, jotka mahdollistavat antennin keilan kääntämisen ja muokkaamisen elektronisesti ilman kuluvia osia ja mittaavat erittäin tarkkoja tuloksia nopeasti. Vaiheohjatut antenniryhmät ovat tänä päivänä osa suurempaa elektronisen sodankäynnin repertuaaria. (4.)

Elektroniikan kehittymisen mukana on ajan kuluessa muodostunut useita toisistaan erottuvia RDF-järjestelmien sukupolvia. Varhaiset järjestelmät hyödynsivät mekaanisesti kiertäviä antenneja, jotka vertailivat signaalien vahvuuksia antennien ollessa tietyissä kohdissa kiertoliikkeessä. Myöhemmin tästä samasta konseptista seurasi useita elektronisia versioita. Modernit järjestelmät käyttävät vaiheenvertailu- tai doppler-tekniikoita, jotka ovat yleensä helpompia automatisoida. Aikaiset englantilaiset tutkajärjestelmät, Chain Home -järjestelmät, käyttivät hyväkseen suuria RDF-vastaanottimia suuntien määrittämiseen. Myöhemmät tutkat tavallisesti käyttivät yksittäistä antennia

lähettämiseen sekä vastaanottamiseen ja suunta määritettiin siitä, mihin suuntaan antenni osoitti. (4.)

Toisessa maailmansodassa käytettiin huomattavia resursseja ja ponnisteluita salaisten radiolähetinten tunnistamiseen Britannian alueelta radiosuuntiman avulla. RSS (Radio Security Service), toisella nimellään MI8 (Military Intelligence, Section 8), suoritti lähetinten tunnistamisoperaation. Aluksi asetettiin kolme U Adcock HF DF -asemaa vuonna 1939, joiden pystyttämisestä vastasi Englannin posti. Sodan julistamisen myötä MI5 (Military Intelligence, Section 5) sekä RSS kehittivät niistä suuremman verkon. (4.)

Vuoteen 1941 mennessä vain muutamia laittomia lähettimiä oli tunnistettu Britannian alueelta. Nämä olivat saksalaisten agenttien asettamia ja ne lähettivät salaa signaalia MI5-viraston alla. Miehitetyistä ja neutraaleista valtioista ympäri Eurooppaa kirjattiin monia saksalaisilta agenteilta peräisin olevia salaisia lähetyksiä. Radioliikenteestä tuli arvokas tiedustelun lähde, joten RSS siirrettiin MI6-viraston (Military Intelligence, Section 6) hallintaan, joka oli vastuussa Britannian ulkopuolisesta salaisesta tiedustelusta. Radiosuuntiman hyödyntämisen ja lähetysten sieppaamisen volyymi ja merkitys kasvoi vuoteen 1945 asti. Toisen maailmansodan jälkeen useat RSS:n radiosuuntima-asemat jatkoivat toimintaansa kylmään sotaan GCHQ:n hallinnassa (The British SIGINT organisation). (4.)

Nykyään suurin osa Britannian radiosuuntimapaikannuksesta kohdistuu luvattomien kaupallisella radiotaajuusalueella (87,5–108,0 MHz) toimivien lähetysten jäljittämiseen, joihin käytetään pääasiassa kauko-ohjattuja VHF-paikantimien verkkoja, jotka sijaitsevat suurten kaupunkien ympärillä. (4.)

3 MIKROKONTROLLERIKORTTI

Tässä luvussa kerrotaan Arduino Due -mikrokontrollerikortista sekä SAM3X8E-mikrokontrollerista, joita opinnäytetyössä on käytetty. Työssä tarvittiin nopeaa mikrokontrolleria, jotta saadaan nopeasti ja tarkasti laskettua sekä mitattua lähettimen suunta suuntima-algoritmiohjelman avulla.

Algoritmi hyödyntää 64-bittisiä muuttujia 32-bittisten sijaan. 32-bittisillä muuttujilla laskutoimituksien ja mittauksien tulokset pyöristyvät liikaa, eikä näin ollen voida huomata lähettimen lähetystaajuuden ja liikkeessä mitatun doppler-taajuuden eroa hyvin pienissä doppler-siirtymissä. Jos käytössä olisi superheterodyne-vastaanotin, jolla lähettimen taajuus muutetaan matalammaksi välitaajuudeksi, 32-bittiset tai jopa 16-bittiset muuttujat luultavasti riittäisivät välitaajuuksille.

3.1 Arduino Due -kortti

Arduino Due on Atmelin SAM3X8E-mikrokontrolleriin perustuva mikrokontrollerikortti. Se on ensimmäinen Arduino-kortti, joka perustuu ARM-ytimiseen 32-bittiseen mikrokontrolleriin. Arduino Due sisältää 54 digitaalista I/O-pinniä (joista 12:ta voi käyttää PWM-lähtöinä), 12 analogista tuloa, 4 UART-piiriä, 84 MHz:n kellotaajuudella toimivan mikroprosessorin, mikro-USB-yhteyden, 2 DA-muunninta, 2 TWI-rajapintaa, virtaliittimen, SPI-väylän, JTAG-portin, sekä reset- ja erase-painikkeet. Toisin kuin muiden tämän hetken Arduino-korttien, Arduino Due -kortin käyttöjännite on 3,3 V. Tällöin sitä suurempaa jännitettä ei voi syöttää kortin I/O-pinneihin. Arduino-kortti sisältää kaikki komponentit, joita mikrokontrolleri tarvitsee toimiakseen. (5.)

Doppler-suuntima-algoritmi sisältää taajuuslaskurin, jonka testaus voidaan suorittaa joko funktiogeneraattorilla tai Arduino Due -kortin PWM-pinnejä hyödyntämällä. PWM-pinneistä pinni 9 on valittu PWM-signaalin lähdöksi suuntima-algoritmiohjelmassa. Jotta PWM-pinnistä saadaan taajuus, täytyy se ensin asettaa lähdöksi komennolla ”pinMode(9,OUTPUT);”, jonka jälkeen siihen voidaan asettaa noin 1 kHz:n taajuus komennolla ”analogWrite(9, 127);”. analogWrite-käskyn sulkujen sisällä olevalla vasemmanpuoleisella numerolla

valitaan haluttu pinni ja oikeanpuoleisella asetetaan pulssisuhde. 127 tarkoittaa 50 %:n pulssisuhdetta, jossa sekä puolijaksot ”ykkönen” että ”nolla” ovat yhtä kauan aktiivisia jaksonajassa.

Pulssisuhde on yleisesti sähkötekniikan käsite, joka ilmaisee jaksollisten aaltomuotojen puolijaksojen suhteen jaksoon. Pulssisuhdetta käytetään yleensä puhuttaessa pulssileveysmodulaatiosta ja kanttiaallosta, mutta sitä voidaan soveltaa myös muihin aaltomuotoihin, mikäli puolijaksot voidaan erottaa toisistaan esimerkiksi negatiiviseksi ja positiiviseksi puolijaksoksi. (8.)

3.2 AT91SAM3X8E-mikrokontrolleri

Atmelin SAM3X8E-mikrokontrolleri perustuu ARM® Cortex™-M3 -mikroprosessoriin. Mikrokontrollerin ominaisuudet: kellotaajuus 84 MHz, 512 kilotavua Flash-muistia ja 100 kilotavua SRAM-muistia. SAM3X8E sisältää pitkälle integroituja oheislaitteita yhteyksiin ja kommunikaatioon. Oheislaitteet ovat Ethernet-väylä, CAN-väylä, HS USB -väylä ja PHY-piiri, nopeat SD-/SDIO-/MMC-korttipaikat, useita USART-, SPI-, TWI-väyliä ja yksi I2C-väylä. Lisäksi mikrokontrolleri sisältää 12-bittiset AD- ja DA-muuntimet, 32-bittiset ajastimet, lämpötilansensorin, PWM-ajastimen ja RTC:n (real-time clock). 16-bittinen ulkoinen väylärajapinta tukee SRAM- ja PSRAM-muisteja sekä NOR-/NAND-Flash-muisteja virheen korjauksella. Atmelin QTouch® Library -kirjasto on käytettävissä SAM3X8E:lle. Kirjastolla voidaan helposti liittää painikkeita, liukusäätimiä ja pyöriä. Laite toimii 1,62–3,6 V:n käyttöjännitteellä ja sitä on saatavilla 144 pinnin QFP- ja BGA-koteloilla. (9.)

4 OSOITINNÄYTÖN SUUNNITTELU JA VALMISTUS

Tässä luvussa käydään läpi Doppler-kompassin osoitinnäytön suunnittelu ja valmistus aina alkuvaiheen määrittelyistä piiri- ja piirilevykaavioon sekä koekytkentään ja siitä valmiiseen kokonaisuuteen.

4.1 Suunnittelu

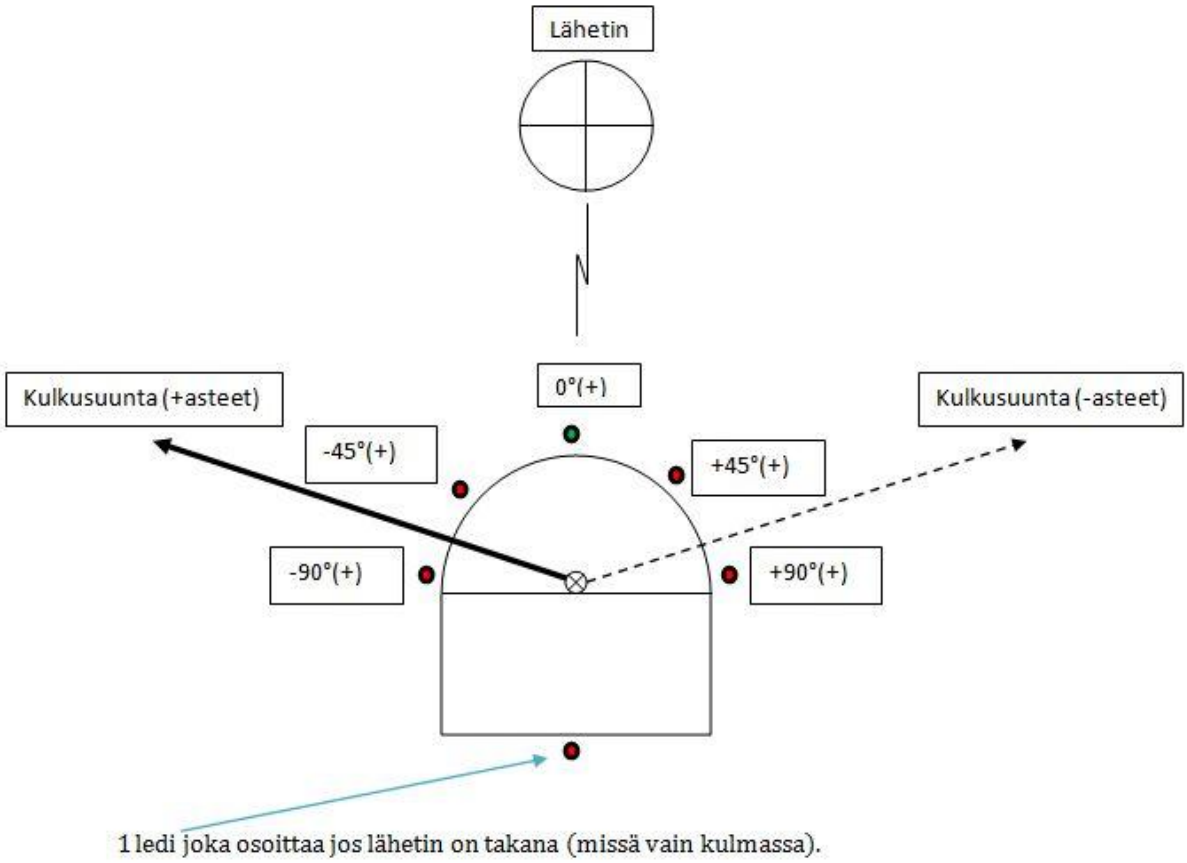
Tavoitteena osoitinnäytölle oli olla mahdollisimman selkeä sekä tarpeeksi pienikokoinen ja kevyt kuljetettavaksi ihmisen kämmenellä. Alkuperäisessä suunnitelmassa näytön oli tarkoitus koostua ympyrän muotoisesta LED-asetelmasta, jossa ledit sijaitsisivat 11,25°:n päässä toisistaan eli olisi tarvittu 32 lediä. Doppler-suuntima-algoritmin toiminnan vuoksi päädyin tekemään puoliympyrän muotoisen 18 ledistä koostuvan osoitinnäytön, jossa 17 lediä muodostaa puoliympyrän ja yksi ledi sijaitsee sen takana. Puoliympyrä näyttää 180° eteenpäin, eli kun ollaan kulkemassa kohti lähetintä –90°–+90°:n kulmassa, syttyy jokin puoliympyrän ledeistä. Puoliympyrän takana sijaitseva ledi syttyy, kun ollaan kulkemassa lähettimestä poispäin –90°–+90°:n kulmissa.

Kuvassa 3 on havainnollistettu osoitinnäyttön rakennetta, mutta kuvasta poiketen todellisessa näytössä on puoliympyrässä 17 lediä. Silloin, kun ollaan kulkemassa niin, että lähetin sijaitsee käyttäjän (vastaanottimen) oikealla puolella, syttyvät +90°–+5,625°:n kulmissa sijaitsevat ledit. Kun lähetin sijaitsee käyttäjän vasemmalla puolella, syttyvät –90°– –5,625°:n kulmissa sijaitsevat ledit. Asteelle nolla eli kuljettaessa suoraan kohti lähetintä sekä takana sijaitsevalle ledille on omat kontrollointilinjat mikrokontrollerista. Muita ledejä ohjaa kaksi 8-bittistä siirtorekisteripiiriä (74HC595). (Kuva 3.)

Syy puoliympyrään päätymisessä oli suuntima-algoritmin tapa laskea ja mitata asteet lähettimen suunnalle. Kun ollaan kävelemässä lähettimestä poispäin, joutuisi algoritmi laskemaan uudelleen kulman, jossa lähetin sijaitsee, moneen kertaan silloinkin, kun käyttäjä on menossa väärään suuntaan. On mielekkäämpää käyttäjän sekä ohjelman koodin ja mikrokontrollerin nopeuden kannalta, että algoritmi aloittaa kulman laskennan suunnalle sekä mahdolliset

korjaustoimet vasta, kun käyttäjä on kulkemassa lähetintä kohti. Näin vältytään turhalta mikrokontrollerin kuormitukselta.

Algoritmi korjaa kulkusuuntaa vain silloin, kun ollaan kulkemassa niin, että lähetin sijaitsee käyttäjän vasemmalla puolella ja sytyttää korjausta vaastaavia miinusasteiden (–90°– –0°) ledejä. Kuvan 3 katkoviivanuoli havainnollistaa kulkusuuntaa, milloin miinusasteet syttyvät. Kuljettaessa niin, että lähetin sijaitsee käyttäjän oikealla puolella, ei suunnan korjaustoimia tarvita, sillä tämä suunta on määritelty oletuskulkusuunnaksi algoritmissa. Oletuskulkusuunta näkyy paksumpana mustana nuolena kuvassa 3.



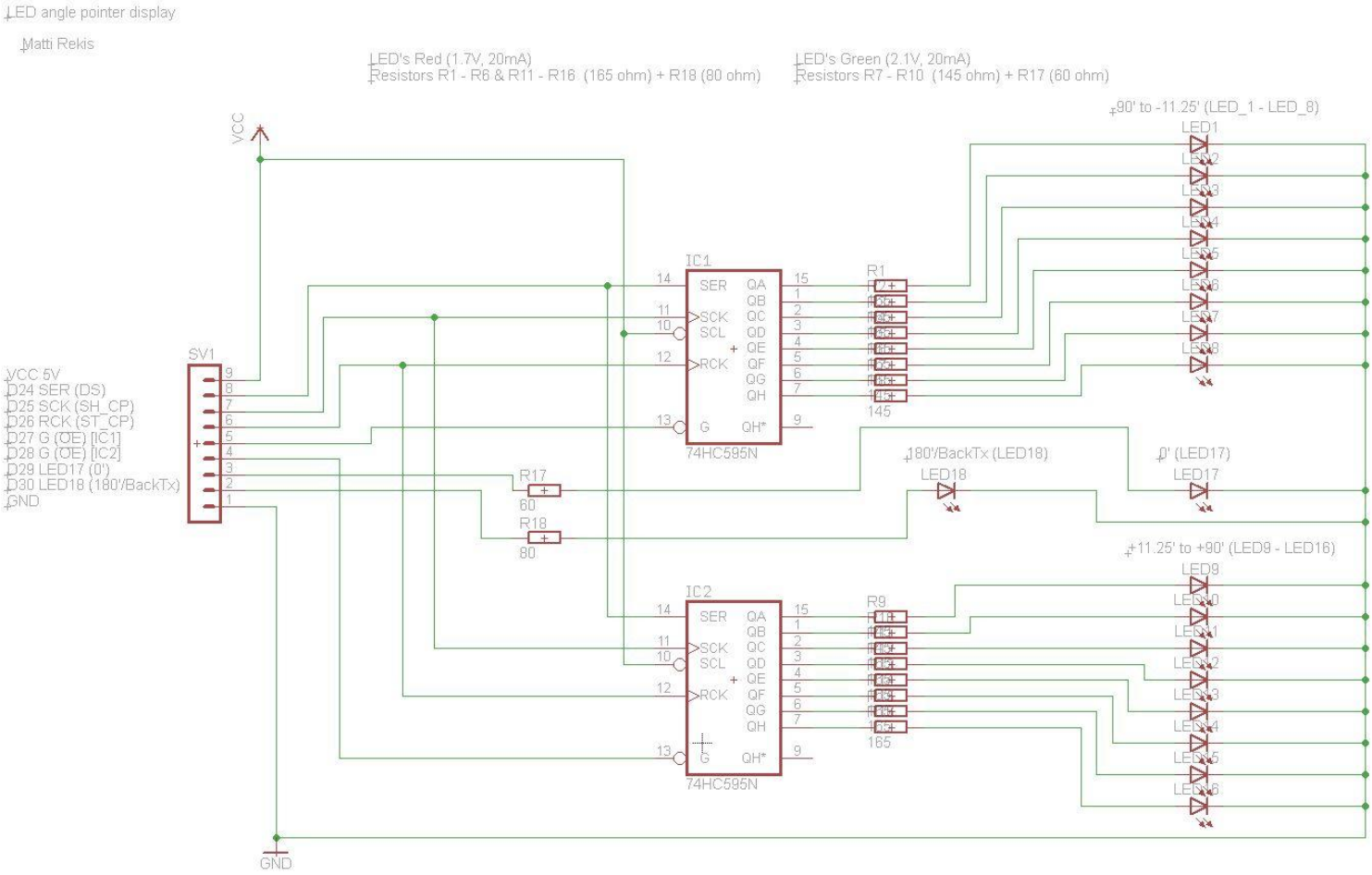
KUVA 3. Havainnollistava malli osoitinnäytöstä (+-merkit sulkujen sisällä tarkoittavat kulkusuuntaa lähetintä kohti)

Osoitinnäyttökytkennässä on päädytty hyödyntämään kahta siirtorekisteripiiriä, jotta voidaan säästää mikrokontrollerin pinnejä sekä vähentää mikrokontrollerin ja näytön välisen johdotuksen tarvetta. 74HC595-piirit ohjaavat +90°–+5,625°:n ja –90°– –5,625°:n kulmia. Missä tahansa kulmassa lähettimestä poispäin

kuljettaessa syttyy puoliympyrän takana sijaitseva yksittäinen ledi, jota kontrolloi Arduino Due -piirikortista oma ohjauslinja. Samoin asteelle nolla eli kulmille, jotka ovat suurempia kuin –5,625° ja pienempiä kuin +5,625°, löytyy piirikortista oma ohjauslinjansa.

Osoitinnäytön piirikaavion ja -levyn suunnittelu tapahtui käyttäen CadSoftin EAGLE-piirisuunnitteluohjelmiston LITE-versiota (v.6.5.0). Vihreät ledit ilmoittavat suunnan kulmille, jotka ovat suurempia kuin –28,125° ja pienempiä kuin +28,125°, ja punaiset ledit osoittavat kaikki muut kulmat. Vihreiden ledien tarkoituksena on havainnollistaa, milloin ollaan oikeassa suunnassa lähettimeen päin.

Kuvassa 4 on esitetty piirikaavio osoitinnäyttökytkennästä. Piirikaaviosta nähdään, että samat data- ja ohjaussignaalit (SER (SI), SCK, RCK), pois lukien lähdönsallinta signaalit ($\overline{\mathrm{G}}$), annetaan kummallekin siirtorekisteripiirille. Jotta saadaan haluttu ledi syttymään piirien saadessa samat datasignaalit, täytyy lähdönsallintalinjoja ($\overline{\mathrm{G}}$) kontrolloida kummallekin piirille erikseen. Tämä on suoritettu ohjelmakoodissa. 74HC595-piirit näkyvät kuvassa nimillä IC1 ja IC2. (Kuva 4.)

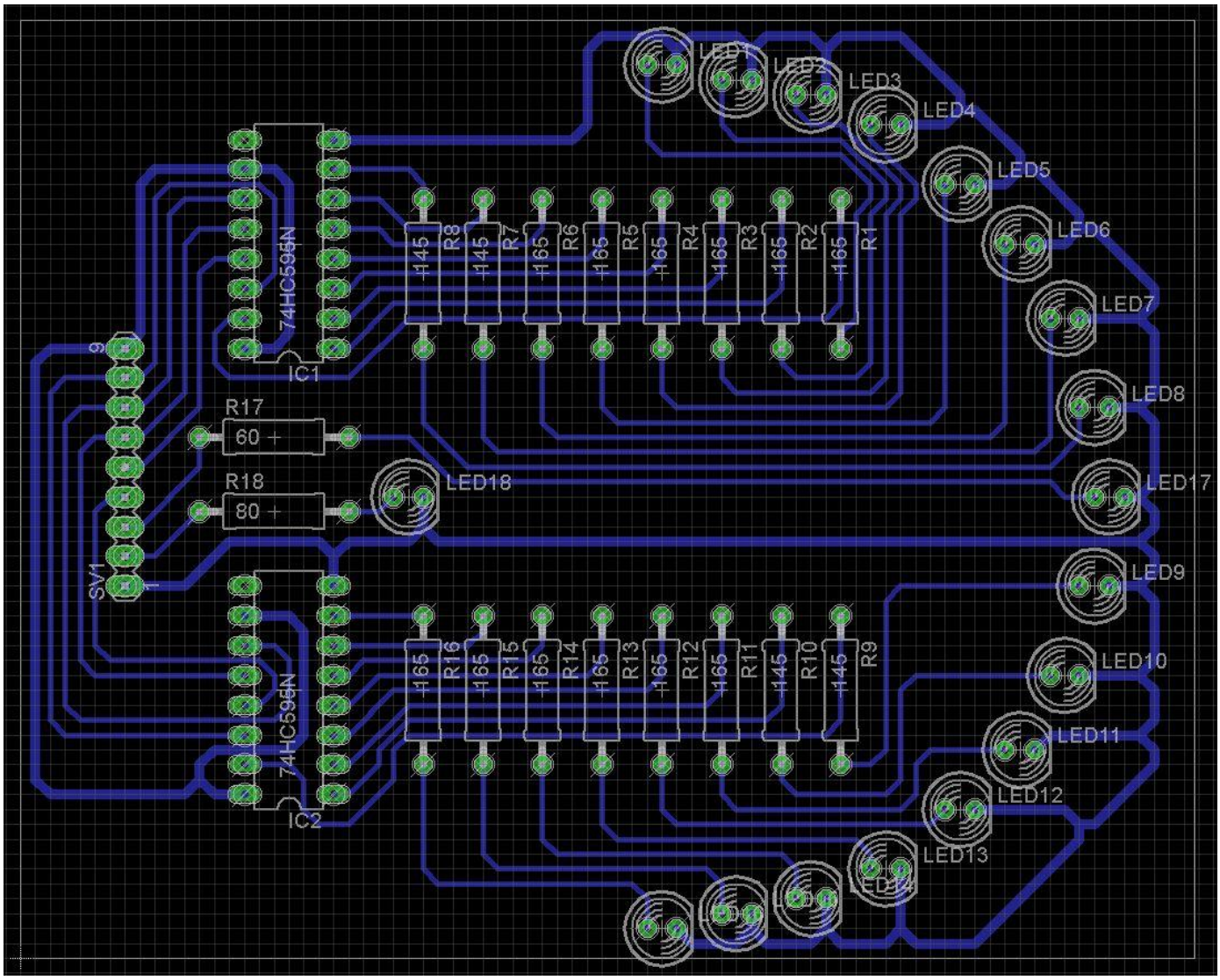


KUVA 4. Osoitinnäytön piirikaavio

Kuvassa 5 nähdään näytön piirilevykaavio, jossa ledit 1–17 on asetettu puoliympyrään osoittamaan suuntaa kuljettaessa kohti lähetintä ja yksi ledi osoittamaan suuntaa kuljettaessa siitä poispäin. Ledit 7–10 sekä 17 ovat vihreitä ja loput punaisia. Ledien etuvastusten lukuarvot saadaan laskettua Ohmin lain avulla (kaava 6). Punaisen ledin kynnysjännite on 1,7 V ja vihreän 2,1 V, sekä kummankin virrankulutus on 20 mA. Ledien 1–8 ja 9–16 kytkentöihin syötetään 5 V:n jännite ja ledeille 17–18 3,3 V:n jännite. Piirilevy on suunniteltu yksipuoleiseksi. (Kuva 5.)

$$R_{etu} = \frac{U_{in} - U_{out}}{I_{led}}$$ *KAAVA 6*

R_{etu} = ledin etuvastuksen resistanssi
U_{out} = haluttu ulostulojännite (ledin kynnysjännite)
U_{in} = kytkentään syötetty jännite
I_{led} = ledin virrankulutus

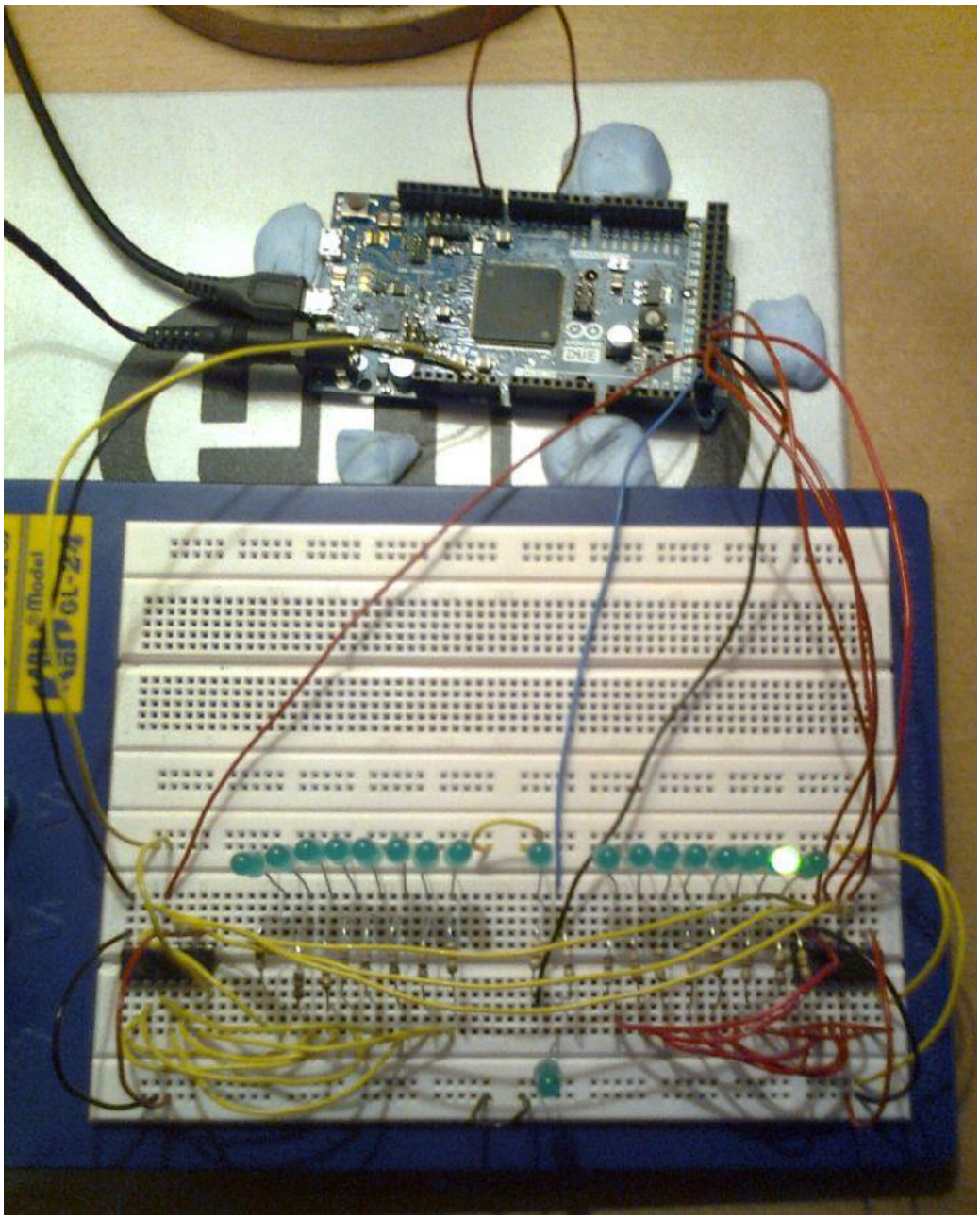


KUVA 5. Osoitinnäytön piirilevykaavio

4.2 Valmistus

Kytkennän testaaminen suoritettiin ensin koekytkentäalustalla, jonka jälkeen muodostettiin piirilevykaavioista Gerber-tiedostot ja jyrsittiin piirilevy. Juottamisen jälkeen piirilevyltä löytyi oikosulku maajohtimen ja pohjakuparin välillä. Pienen korjausoperaation myötä oikosulku poistui ja osoitinnäyttöä voitiin testata Arduino Due -piirikortin kanssa, jossa se toimi hyvin.

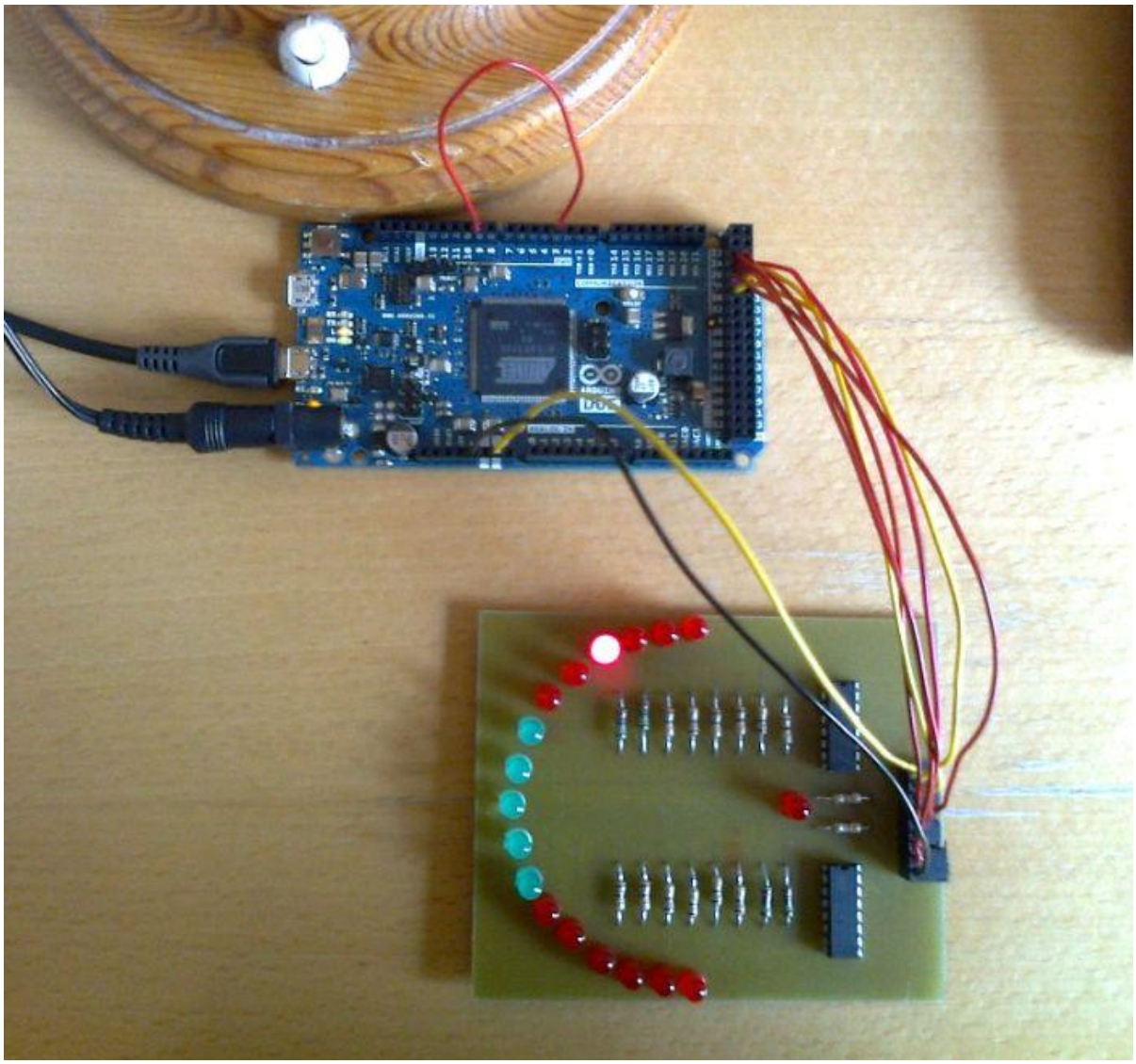
Kuvassa 6 on osoitinnäytön piirikaavio koottuna koekytkentäalustalle. Kytkentäalustalla ledit eivät ole vielä puoliympyrässä, mutta syttynyt ledi osoittaa ohjelmassa simuloidun lähettimen suunnan olevan noin +75°:n kulmassa vastaanottimen kulkusuunnasta.



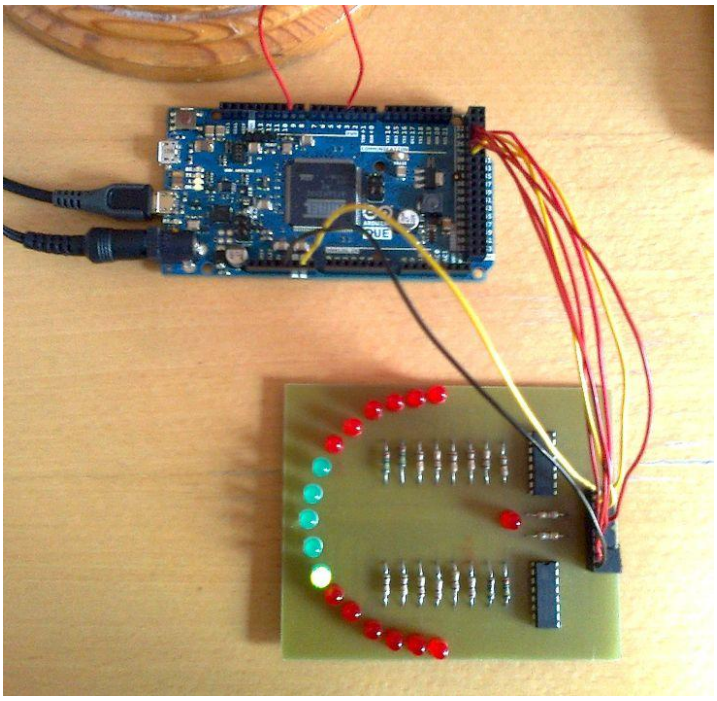
KUVA 6. Osoitinnäytön koekytkentä

Kuvassa 7 on piirilevykaavion mukaan suunniteltu ja valmistettu osoitinnäyttö toiminnassa. Puoliympyrästä huomataan, että syttynyt ledi osoittaa ohjelmakoodissa simuloidun lähettimen sijainnin olevan noin +55°:n kulmassa vastaanottimen liikesuunnasta.

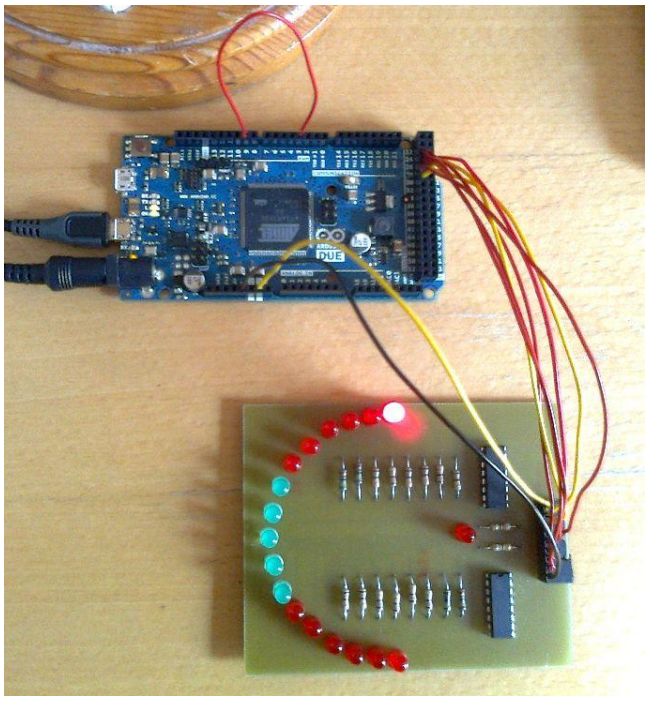
Sivulla 25 on lisää kuvia valmiista osoitinnäytöstä algoritmin toimiessa. Kuva 8 näyttää simuloidun lähettimen olevan noin –25°:n kulmassa, kuva 9 noin +90°:n kulmassa, kuva 10 noin –55°:n kulmassa ja kuva 11 osoittaa, että lähetin sijaitsee jossain vastaanottimen takana.



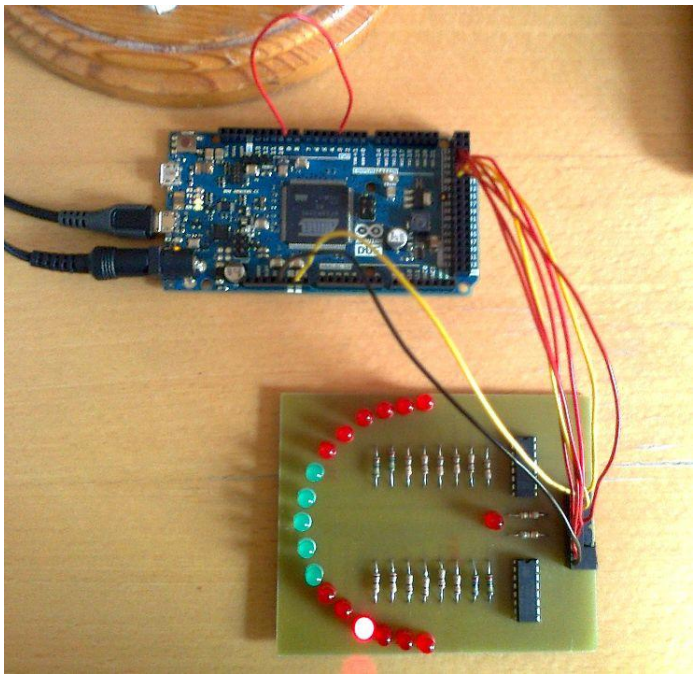
KUVA 7. Valmis osoitinnäyttö



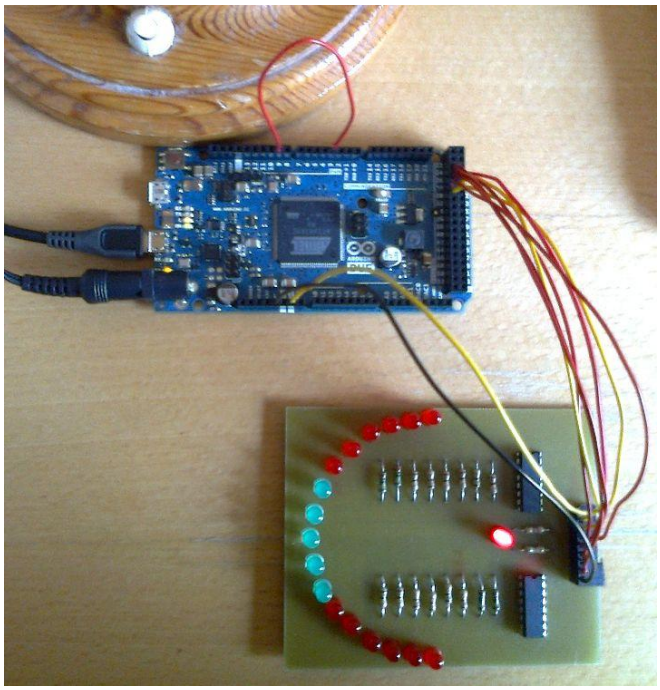
KUVA 8. Lähettimen suunta n. –25°



KUVA 9. Lähettimen suunta n. +90°



KUVA 10. Lähettimen suunta n. –55°



KUVA 11. Lähetin sijaitsee jossain kulkusuunnan takana

5 SUUNTIMA-ALGORITMIN SUUNNITTELU

Tämän opinnäytetyön päätarkoituksena on Doppler-kompassin suuntima-algoritmin toteuttaminen. Lopullisen Doppler-kompassin tulisi sisältää tarkka taajuuden mittausyksikkö, jotta radiolähetin voidaan paikantaa, sillä doppler-vääristymä suurissakin taajuuksissa (>MHz) on vain muutamien hertsien luokkaa, riippuen tietenkin vastaanottimen nopeudesta. Doppler-taajuuden mittaukseen saadaan helpotusta, jos käytetään superheterodyne-vastaanotinta. Superheterodyne-vastaanottimella saadaan doppler-vääristymä siirrettyä lähettimestä vastaanotetusta signaalista suoraan matalammalle välitaajuudelle, jolloin sen mittaaminen ja jatkokäsittely on huomattavasti vaivattomampaa. Tässä luvussa käydään läpi suuntima-algoritmin suunnittelun vaiheet sekä algoritmissa tarvittavat toiminnot ja yhtälöt.

5.1 Alustavaa

Doppler-suuntima-algoritmin tarkoitus on laskea radiolähettimen suunta vastaanottimen eli Doppler-kompassin liikkeen perusteella muodostuvasta doppler-siirtymästä lähettimen taajuudessa. Lähettimen oletetaan aina pysyvän paikoillaan. Algoritmiin syötetään ensin lähettimen taajuus ilman suhteellista liikettä vastaanottimen kanssa. Sitten tätä lähettimen taajuuarvoa, joka ei sisällä doppler-siirtymää, hyödyntäen lasketaan se taajuus, joka olisi, jos kuljettaisiin suoraan lähetintä kohti tietyllä nopeudella. Kutsutaan tätä taajuusarvoa teoreettiseksi taajuudeksi. Kun teoreettisen taajuuden doppler-vääristymä on laskettu, voidaan tätä vääristymää hyväksi käyttäen selvittää lähettimen suunta vertaamalla sitä todellisen mitatun taajuuden doppler-vääristymään vastaanottimen ollessa liikkeessä. Kaavassa 7 on esitetty teoreettisen taajuuden laskemiseen vaadittava yhtälö sekä tarvittavat muuttujat ja vakiot.

$$f_t = f_{\text{gen}} * \frac{c+v}{c} \qquad \textit{KAAVA 7}$$

$f_t =$ teoreettinen taajuus, jos kuljettaisiin suoraan lähetintä kohti tietyllä nopeudella
$f_{gen} =$ lähettimen taajuus vastaanottimen ja lähettimen ollessa paikallaan
$c =$ valonnopeus

$v =$ vastaanottimelle määritetty liikenopeus

Jotta saadaan laskettua kulma, missä lähetin sijaitsee vastaanottimen kulkusuuntaan nähden, pitää tietyssä nopeudessa mitatun lähettimen taajuuden doppler-vääristymää verrata teoreettisen taajuuden doppler-vääristymään, jossa vastaanotin kulkisi suoraan lähetintä kohti. Yhtälö on esitetty kaavassa 8.

$$\theta = \arccos\left(\frac{f_{m-fgen}}{f_{t-fgen}}\right) = \arccos\left(\frac{\Delta f_m}{\Delta f_t}\right) \qquad \textit{KAAVA 8}$$

$\theta =$ kulma vastaanottimen kulkusuunnan ja lähettimen välillä
$f_{gen} =$ lähettimen taajuus vastaanottimen ja lähettimen ollessa paikallaan
$f_m =$ todellinen (mitattu tai simuloitu) taajuus vastaanottimen liikkuessa tietyllä nopeudella
$f_t =$ teoreettinen taajuus, mikäli vastaanotin kulkisi suoraan lähetintä kohti tietyllä nopeudella
$\Delta f_m =$ lähettimen taajuuden doppler-vääristymä vastaanottimen liikkuessa tietyllä nopeudella
$\Delta f_t =$ teoreettinen lähettimen taajuuden doppler-vääristymä, jos vastaanotin liikkuisi tietyllä nopeudella suoraan lähetintä kohti

Algoritmi laskee kulman vain silloin, kun ollaan kulkemassa radiolähetintä kohti. Jos kuljetaan lähettimestä poispäin, algoritmi ilmoittaa vain, että suunta on väärä. Tieto, onko vastaanotin menossa kohti lähetintä vai siitä poispäin, saadaan selville vertaamalla lähettimen todellista lähetystaajuutta eli ilman liikettä ja doppler-ilmiötä mitattua taajuutta f_{gen} liikkeessä mitattuun taajuuteen f_m eli taajuuteen, joka sisältää doppler-vääristymän. Lähetintä kohti liikkuessa ehto on $f_m > f_{gen}$ ja siitä poispäin liikkuessa $f_m < f_{gen}$. Kun lähetin ja vastaanotin ovat paikallaan $f_m = f_{gen}$, tällöin myöskään doppler-ilmiötä ei synny.

5.2 Suunnankorjaus

Doppler-suuntima-algoritmi laskee lähettimen kulman vastaanottimeen nähden ainoastaan, jos kuljetaan lähetintä kohti. Muuten ilmoitetaan vain, että lähetin sijaitsee käyttäjän takana. Tämä säästää mikroprosessoria turhalta laskennalta, koska suunnan uudelleenlaskenta täytyisi tehdä joka tapauksessa, kun käännytään lähetintä kohti.

Jotta saadaan selvitettyä lähettimen todellinen kulma, siis se kummalla puolella lähetin oikeasti sijaitsee kulkusuunnassa, +0°–+90° vai –0°– –90°, täytyy liikkeessä saatua doppler-vääristymää Δf_m verrata aiempaan liikkeessä saatuun doppler-vääristymään Δf_{m2}. Jos $\Delta f_m < \Delta f_{m2}$, tiedetään, että algoritmin täytyy korjata kulmaa plusasteista miinusasteisiin. Tämä vaihto plusasteisista kulmista (+0°–+90°) miinusasteisin kulmiin (–0° – –90°) aktivoidaan vain silloin, kun korjattava kulma on plusasteinen ja pienempi tai yhtä suuri kuin 45°.

Suurempien kuin 45°:n kulmien korjaus ilmoittaisi lähettimen olevan jossain vastaanottimen kulkusuunnan takana sijaitsevassa kulmassa. Tämä ei voi pitää paikkaansa, sillä vastaanottimen käyttäjä on menossa lähetintä kohti (muuten osoitinnäytön puoliympyrän ledeistä mikään ei olisi syttynyt alun perinkään).

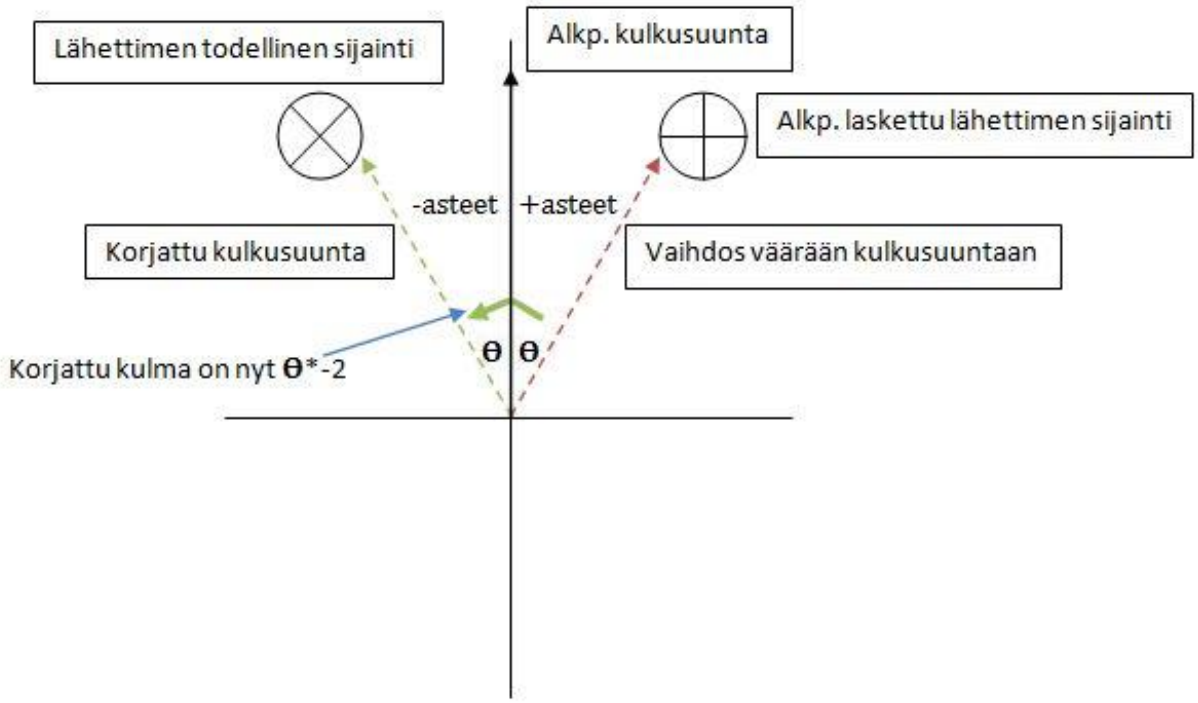
Jos alkuperäinen korjattava kulma olisi suurempi kuin 45°, esimerkiksi 70°, ei asteen korjaustoimintoa tarvittaisi, koska käyttäjä kääntyisi joka tapauksessa ensin 140°:n kulmaan lähettimestä poispäin, joka olisi osoitinnäytön puoliympyrän (+0–+90° ja –0– –90°) ulkopuolella. Käännöksen jälkeen algorimi ilmoittaisi, että lähetin sijaitsee jossain kulkusuunnan takana sytyttämällä suunnan osoitinnäytöstä taaksepäin osoittavan ledin. Suunnankorjaus aktivoituu vasta, kun kuljetaan vähän matkaa väärän (korjattavan) kulman mukaan.

Suuntima-algoritmi korjaa kulkusuuntaa vain silloin, kun lähetin sijaitsee vastaanottimen kulkusuunnan vasemmalla puolella. Algoritmi olettaa aina ensiksi vastaanottimen liikkuvan niin, että lähetin sijaitsee käyttäjän oikealla puolella ja laskee sekä sytyttää plusasteisen kulman LED-osoitinnäytöstä. Tämän jälkeen algoritmi alkaa selvittää onko liikesuunta väärä. Osoitinnäyttöä ja sen toimintaa on havainnoitu luvun 4 kuvassa 3. Korjaustoiminnon yhtälö on esitetty kaavassa 9.

$$\theta_- = \theta_+ * -2 \qquad \textit{KAAVA 9}$$

θ_- = miinusasteinen (korjattu) kulma vastaanottimen kulkusuunnan ja lähettimen välillä
θ_+ = plusasteinen kulma vastaanottimen kulkusuunnan ja lähettimen välillä

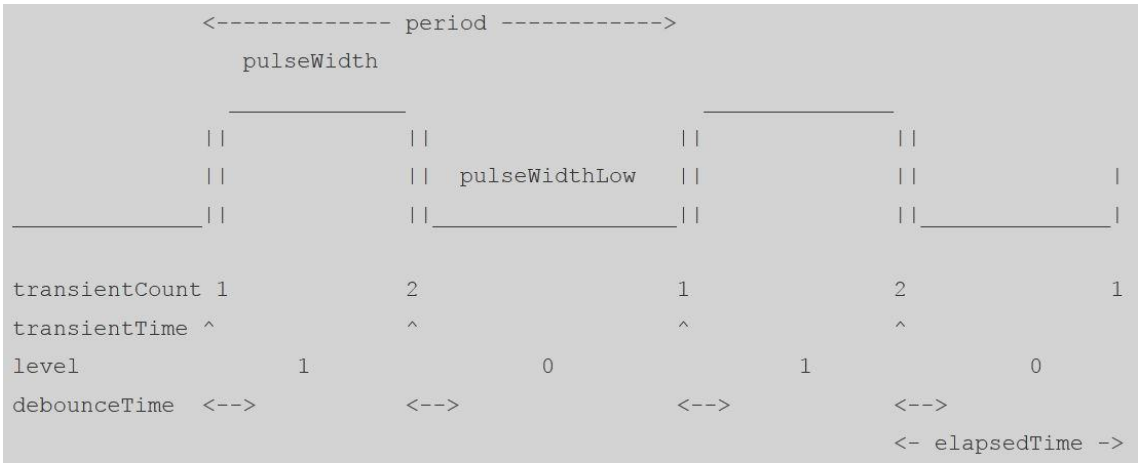
Käyttäjän kulkiessa väärän kulman suuntaan täytyy huomioida, että lähetin sijaitsee täysin vastakkaisella puolella kuin ennen korjaustoimia oletettiin, joten korjattava kulma θ_+ on kerrottava kahdella. Negatiivinen etumerkki 2:n edessä kertoo algoritmille, että sytytettävä ledi sijaitsee osoitinnäytön miinusasteiden puolella. Korjaustoiminnon suunnanmuutosta on havainnollistettu kuvassa 12.



KUVA 12. Suunnankorjaustoiminnon havainnollistus

5.3 Taajuuslaskuri

Suuntima-algoritmi sisältää taajuuslaskurin, joka toimii ohjelmallisesti mikrokontrollerin keskeytystä hyödyntäen. Laskuri laskee taajuuden havaitsemalla mitattavan signaalin HIGH- ja LOW-tilojen (”1” ja ”0”) muutoksia. Aina, kun mitattavassa signaalissa vaihtuu tila ”ykkösestä” ”nollaan” tai ”nollasta” ”ykköseen”, tapahtuu mikrokontrollerissa keskeytys. Taajuus f voidaan laskea mittaamalla jaksonaikaan T eli mitattavan signaalin HIGH-LOW-periodiin kuluva aika, sillä $f = 1/T$. Jaksonaika voidaan yhtä hyvin myös määrittää LOW-HIGH-periodista. Jaksonaika näkyy kuvassa 13 nimellä ”period”. (6, alaotsikko 9 Frequency counter library.)



KUVA 13. Jaksonajan mittaus taajuuden määrittämiseksi (6, alaotsikko 9 Frequency counter library)

Taajuuslaskurifunktion tarkkuus ei ole sellaisenaan riittävä doppler-siirtymän mittaamiseen, ainakaan ellei käytössä ole superheterodyne-vastaanotinta. Mittaustuloksissa jopa matalalla noin 100 Hz:n taajuudella ilmenee epätarkkuutta ja mitattu taajuusarvo heittelee ±0,5 Hz eri mittauskerroilla, vaikka taajuus pidettäisiin funktiogeneraattorissa vakiona. Kun taajuutta kasvatetaan, myös mittausepätarkkuus kasvaa, eikä taajuuslaskurifunktio kykene mittaamaan korkeampia kuin 2 kHz:n taajuuksia ollenkaan. Syy laskurin rajoittuneeseen kykyyn laskea korkeampia taajuuksia saattaa olla siinä, ettei funktioita ole suoraan suunniteltu Arduino Duen ARM-mikrokontrollerille, jolla suuntima-algoritmi toimii.

Taajuuslaskurin epätarkkuuden takia tarkempi algoritmin testaus on toteutettu antamalla muuttujille f_{gen} (lähettimen taajuus ilman doppler-vääristymää) ja Δf_m (lähettimen taajuuden doppler-vääristymä) arvot ohjelmallisesti. Myös vastaanottimen nopeudelle annetaan arvo suoraan ohjelmassa.

6 SUUNTIMA-ALGORITMIN OHJELMOINTI

Ohjelmointi suoritettiin C-kielellä Arduino-ohjelmistolla. Tässä ja seuraavissa luvuissa käytetyt muuttujat, vakiot, funktiot ja pinnit on määritelty luvussa 6.1. (Huom! Luvussa 8 muuttujat *ft*, *dft*, *fm* ja *fgen* sekä funktio *dtow(x,y,z)* on määritelty välitaajuusvastaanottimelle sopivaksi.)

C-kielinen doppler-suuntima-algoritmiohjelma on kokonaisuudessaan erilaisina testausversioina liitteissä 2, 3, 4, 5, 6 ja 7. Liitteet 5, 6 ja 7 sisältävät suuntima-algoritmiohjelman superheterodyne-versiot.

6.1 Muuttujat, vakiot ja funktiot

Arduino Due -mikrokontrollerikortissa muuttujan tietotyyppi double on 64-bittinen ja float on 32-bittinen liukuluku. Suuntima-algoritmiohjelma käyttää 64-bittisiä lukuja taajuuksille ja suunnan laskentaan, koska 32-bittiset luvut eivät ole tarpeeksi tarkkoja tiettyihin operaatioihin, joita tarvitaan. Esimerkiksi doppler-siirtymän sisältämän taajuuden vertaaminen ilman doppler-siirtymää sisältävään taajuuteen ei onnistu 32-bittisillä luvuilla, jos käytössä ei ole välitaajuusvastaanotinta (superheterodyne-vastaanotin). Algoritmin superheterodyne-version muuttujille (*fm*, *fgen*, *dfm*...), jossa välitaajuudelle sisällytetään doppler-vääristymä, riittää matalampikin tarkkuus (esim. 32- tai 16-bittiä).

Ohjelman muuttujien, vakioiden ja funktioiden tietotyypit

- double: 64-bittinen liukuluku.
- const double: 64-bittinen liukuluku, jonka arvo on vakio.
- int: 32-bittinen kokonaisluku.
- byte: 8-bittinen etumerkitön kokonaisluku.
- const byte: 8-bittinen etumerkitön kokonaisluku, jonka arvo on vakio.
- bool: Looginen tietotyyppi, joka voi sisältää vain totuusarvon ”tosi” (”1”) tai ”epätosi” (”0”).

Suuntima-algoritmin sisältämät muuttujat, vakiot ja funktiot, joita tarvitaan lähettimen suunnan laskennassa

Muuttujat:

- double *forig*: Ohjelmallisesti asetettu lähettimen taajuus ilman doppler-siirtymää eli silloin, kun lähetin sekä vastaanotin ovat paikallaan. Tätä muuttujaa käytetään ainoastaan suuntima-algoritmiohjelman superheterodyne-versiossa.
- double *fgen*: Mitattu (tai ohjelmallisesti asetettu) lähettimen taajuus ilman doppler-siirtymää eli silloin, kun lähetin sekä vastaanotin ovat paikallaan. Algoritmin superheterodyne-versiossa *fgen* sisältää superheterodyne-vastaanottimesta saadun välitaajusarvon ilman doppler-vääristymää.
- double *fm*: Mitattu (tai ohjelmallisesti asetettu) lähettimen taajuus, kun vastaanotin on liikkeessä. Algoritmin superheterodyne-versiossa *fm* sisältää superheterodyne-vastaanottimesta saadun välitaajusarvon, kun vastaanotin on liikkeessä.
- double *ft*: Laskettu teoreettinen taajuusarvo sille, jos vastaanotin liikkuisi tietyllä nopeudella suoraan lähetintä kohti.
- double *dfm*: Lähettimen taajuuden tai välitaajuuden (jos käytetään superheterodyne-vastaanotinta) doppler-vääristymä vastaanottimen liikkuessa (*dfm=fm-fgen*).
- double *dft*: Teoreettinen lähettimen taajuuden doppler-vääristymä, mikäli vastaanotin liikkuisi tietyllä nopeudella suoraan lähetintä kohti (*dft=ft-fgen*). Suuntima-algoritmin superheterdyne-versiossa *dft*:n laskentatapa muuttuu (*dft=ft-forig*).
- double *vdf*: Mitatun (tai ohjelmallisesti asetetun) lähettimen taajuuden doppler-vääristymän suhde laskettuun teoreettiseen lähettimen taajuuden doppler-vääristymään, jos vastaanotin liikkuisi suoraan lähetintä kohti (*vdf=dfm*/*dft*).
- double *facos*: Kulma vastaanottimen kulkusuunnan ja lähettimen välillä. Laskettu hyödyntäen arcuskosinia (*facos*=acos(*vdf*)).
- double *dfm2*: Käytetään havaitsemaan, onko laskettu kulma (*facos*) oikein vai tarvitseeko aloittaa suunnan korjaustoimet. *dfm* ei saisi

pienentyä, mikäli liikesuunta on laskettu oikein ja etenemme lähemmäs lähetintä.

Vakiot:

- const double *pi*=3.14159265359: Piin likiarvo.
- const double *c*=299792458: Valonnopeuden likiarvo [m/s].

Funktiot:

- double *dtow*(double *x,* double *y,* double *z*): Aliohjelma doppler-vääristymän sisältämän teoreettisen taajuusarvon *ft* laskemiseksi, joka olisi, mikäli kuljettaisiin suoraan lähetintä kohti tietyllä nopeudella. Parametrien sisällöt ovat: *x* = *fgen*, *y* = *c* ja *z* = *v*. Funktion paluuarvona on double-tietotyyppinen arvo. Suuntima-algoritmin superheterodyne-versiossa parametri *x* saa muuttujan *forig* arvon.

Kirjastot, muuttujat, funktiot ja pinnit, joita tarvitaan muissa ohjelman toiminnoissa

Kirjastot:

- #include <*Streaming.h*>: Kirjasto informaation tulostamista varten mm. Serial Monitor -ikkunaan.
- #include <*FreqPeriodCounter.h*>: FreqPeriodCounter-funktion ohjelmakoodit. (6, alaotsikko 9 Frequency counter library.)

Muuttujat:

- double *rcircled*: Tallennetaan kierroslaskurin taajuuskierroksien lukumäärä uuden *vdf*-arvon laskemista varten *vdf*:n ollessa suurempi kuin yksi. Kokonaisten taajuuskierrosten lukumäärä muodostuu *vdf*:n arvon kokonaisluvun perusteella.
- int *rcircle*: Poistetaan *vdf*-arvosta desimaalit, jotta saadaan kierroksien määrästä tasaluku.
- int *rcc*: Muuttujaan *rcc* kopiodaan *rcircle*:n arvo, jotta voidaan huomata kokonaisen taajuuskierroksen vaihtuminen. Kierroksen vaihtuminen

tapahtuu silloin, kun *rcircle*:n arvo on erisuuri kuin *rcc*:n arvo seuraavan kerran, kun siirrytään kierroslaskurin koodiin.

- byte *i*: Muuttujan *i* avulla päätetään, kuinka monesta taajuustuloksesta otetaan keskiarvo taajuuslaskuri-koodissa.
- const byte *counterInterrupt*: Päätetään, että FreqPeriodCounter-funktio käyttää keskeytysperiaatetta taajuuden mittaamiseen. Aina, kun tulosignaali muuttaa tilaansa (”1” tai ”0”), tapahtuu keskeytys, ja keskeytysten välisen ajan perusteella lasketaan taajuus. (6, alaotsikko 9 Frequency counter library.)
- bool *reset*: Muuttuja *reset* saa arvon ”tosi” aina, kun doppler-suuntimaohjelman suorittaminen aloitetaan alusta, ja ”epätosi”, kun muuttujan *fgen* taajuusarvo on mitattu. *reset*:n avulla tiedetään, kun käyttäjä on painanut Arduino Due -mikrokontrollerikortin reset-painiketta eli kun tahdotaan laskea uusi lähettimen tai välitaajuuden (superheterodyne) taajuusarvo ilman doppler-siirtymää (*fgen*).
- bool *rcflag*: Muuttujan *rcflag* tila vaihtuu aina, kun huomataan kierroslaskuri-koodissa, että kokonainen taajuuskierros on käyty läpi. *rcflag*:n tilan vaihdos vaihtaa *vdf*-arvon laskentatapaa (kierroslaskuri-koodiin siirrytään vain silloin, kun *vdf* on saanut arvon, joka on suurempi kuin yksi).

Pinnit:

- const byte *counterPin*: Arduino Due -kortin pinni 3. Käytetään taajuuden mittaamiseen FreqPeriodCounter-funktiossa. (6, alaotsikko 9 Frequency counter library.)
- const byte *dataPin*: 75HC595-piirin sarjamuotoinen datalinja (SI) Arduino Due -kortin pinnistä 24. (7.) (Kuva 4.)
- const byte *clockPin*: 75HC595-piirin kellotulo (SCK) Arduino Due -kortin pinnistä 25. (7.) (Kuva 4.)
- const byte *latchPin*: Linjalla (RCK) estetään 74HC595-piirin lähtöjen tilat silloin, kun piirin rekisteriin syötetään tietoa. Ohjaus Arduino Due -kortin pinnistä 26. (7.) (Kuva 4.)

- const byte *oePin1*: Piirin IC1 lähtöjen sallintalinja ($\overline{G}$). Ohjaus Arduino Due -kortin pinnistä 27. (7.) (Kuva 4.)
- const byte *oePin2*: Piirin IC2 lähtöjen sallintalinja ($\overline{G}$). Ohjaus Arduino Due -kortin pinnistä 28. (7.) (Kuva 4.)
- const byte *led17Pin*: Astetta nolla osoittavan ledin ohjauslinja (LED17). (Kuva 4.)
- const byte *led18Pin*: Taaksepäin osoittavan ledin ohjauslinja (LED18). (Kuva 4.)

Funktiot:

- void *ledscr*(double *x*): Funktio osoitinnäytön ohjaamiseen. Parametri *x* sisältää sytytettävän ledin kulman.
- void *ledsoff*(): Aliohjelma, joka sammuttaa kaikki ledit osoitinnäytöstä.

6.2 Suunnan laskennan koodit

Ohjelmallinen toteutus lähettimen kulman laskemiselle Doppler-kompassin kulkusuuntaan nähden on esitetty kuvassa 14. Osa ohjelmakoodin riveistä on asetettu kommenteiksi, koska kyseisiä käskyjä käytetään vain, kun tahdotaan testata algoritmia ohjelmallisesti eli antaa tiettyjä arvoja mm. muuttujille *fgen* ja *dfm*.

```
if(fm>fgen) //moving towards to Tx
{
Serial << "Heading to Tx.\n";
//dfm=dfm-0.000001; //angle repair test [-angles] (programmatically simulated dfm)
facos=acos(vdf)*180/pi;
Serial << "[Original angle] vdf: " << vdf << ", Acos: " << facos << " [+Tx_0]\n"; //shows the original angle in the serial monitor
if(dfm<dfm2&&vdf>=cos(45*pi/180)) //if dfm<dfm2 the Rx is going to the wrong way and needs to repair its direction, only angles that are <=45 degrees is repaired
{
facos=-facos*2; //ex. +44 degrees -> after direction repair -88 degrees
Serial << "[Repaired angle] vdf: " << vdf << ", Acos: " << facos << " [+Tx_1]\n"; //shows the repaired angle in the serial monitor
}
ledscr(facos); //function for LED screen
//dfm=dfm+0.1; //increases the value of dfm, for testing the algorithm programmatically
//if(dfm>=(dft*4)) //for testing the cycle calculation code when vdf>1
//dfm=-0.001;
}
else if(fm<fgen) ////moving away from Tx. 1 led at the back of the led screen shows when moving away, so no need to calculate the angle of the Tx
{
Serial << "Heading away Tx.\n";
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, HIGH); //disables IC2 outputs
digitalWrite(led18Pin, HIGH); //set 180 degrees/backTx led on
digitalWrite(led17Pin, LOW); //set 0 degree led off
//facos=acos(vdf)*180/pi;
//Serial << "vdf: " << vdf << ", Acos: " << facos << " [-Tx_0]\n";
//dfm=dfm-1; //decreases the value of dfm, for testing the algorithm programmatically
//if(dfm<=(dft*-2))
//dfm=0.001;
}
else
{
Serial << "No doppler-frequency.\n";
ledsoff(); //turns off all leds
}
```

KUVA 14. Koodi lähettimen kulman laskemiseksi

Jotta voidaan laskea lähettimen suunta, täytyy ensiksi saada muuttujille *fgen*, *fm* ja *ft* arvot. Näistä arvoista voidaan selvittää *dfm* (*dfm=fm-fgen*), *dft* (*dft=ft-fgen*) ja *vdf* (*vdf=dfm/dft*). Suuntima-algoritmin superheterodyne-versiossa *dft*:n ja *ft*:n laskentatavat muuttuvat (*dft=ft-forig*; *ft=dtow(forig,c,v)*). *fgen*:lle ja *fm*:lle saadaan arvot joko ohjelmallisesti asettamalla tai mittaamalla ulkoisesta taajuuslähteestä, esimerkiksi funktiogeneraattorista. Algoritmin sisältämä taajuuslaskuri on liian epätarkka havaitsemaan doppler-vääristymää mitattavasta lähettimen taajuudesta, ellei sitä madalleta välitaajuudelle esimerkiksi 100 Hz:iin superheterodyne-vastaanottimen avulla, joten ohjelman toiminnan tarkempi testaus on toteutettu ohjelmallisesti. Jos annetaan arvo muuttujalle *dfm*, saadaan myös *fm* laskettua (*fm=fgen+dfm*). Muuttujien kuvaukset ovat luvussa 6.1.

ft:n arvo saadaan laskettua kuvan 15 aliohjelman mukaisesti. Funktio laskee teoreettisen taajuusarvon sille, mikäli kuljettaisiin suoraan lähetintä kohti tietyllä nopeudella. Funktion parametrit ovat: *x* = *fgen* (*x* = *forig*, jos kyseessä on ohjelman superheterodyne-versio), *y* = *c*, *z* = *v*. Paluuarvona (*vft*) aliohjelma antaa taajuusarvon muuttujalle *ft*. (Kuva 15.)

```
double dtow(double x, double y, double z) //function for calculating theoretical value of frequency where Rx would be moving straight towards to Tx at certain speed
{
double vft=0;
vft=x*((y+z)/y);
return(vft);
}
```

KUVA 15. Funktio ft-taajuuden laskemista varten

vdf:n arvo saadaan, kun jaetaan lähettimen taajuuden doppler-siirtymä teoreettisella doppler-siirtymällä, missä vastaanotin liikkuisi tietyllä nopeudella suoraan lähetintä kohti. Ohjelman superheterodyne-versiossa *dft*:n laskentatapa muuttuu kuvasta poiketen (*dft=ft-forig*). (Kuva 16.)

```
vdf=dfm/dft; //measured(or simulated) doppler shift in fm [dfm=fm-fgen] / calculated theoretical doppler shift in ft [dft=ft-fgen]
```

KUVA 16. Doppler-vääristymien suhde

Kuvan 14 ohjelmakoodin toiminta on seuraavanlainen:

Jos muuttuja *fm* on suurempi kuin *fgen*, ollaan menossa lähetintä kohti, jolloin täytyy laskea kulma (*facos*), missä lähetin sijaitsee. Tämä onnistuu ottamalla arcuskosinifunktio muuttujasta *vdf* (*facos*=acos(*vdf*)*180/*pi*)). Silloin, kun algoritmi huomaa, että suunnalle laskettu kulma on väärä (*dfm*<*dfm2*) ja pienempi tai yhtä suuri kuin 45°, suoritetaan suunnankorjaustoiminto, jossa alkuperäinen väärin laskettu kulma kerrotaan –2:lla. Kun kulma *facos* on laskettu, siirretään sen lukuarvo aliohjelmaan *ledscr(x)*, joka ohjaa osoitinnäyttöä.

Jos *fm* on pienempi kuin *fgen*, liikutaan lähettimestä poispäin. Tällöin ei lasketa kulmaa ollenkaan, vaan käyttäjälle osoitetaan, että lähetin sijaitsee vastakkaisessa kulkusuunnassa. Tämä tapahtuu laittamalla osoitinnäytön siirtotekisteripiirien lähdöt korkeaimpedanssiseen tilaan ja sammuttamalla nollaa astetta osoittava ledi (LED17) sekä sytyttämällä taaksepäin osoittava ledi (LED18). Luvussa 4.1 on esitetty ledien nimet ja paikat.

Jos *fm* on yhtä suuri kuin *fgen,* ovat lähetin ja vastaanotin paikallaan eikä doppler-ilmiötä synny. Tällöin kirjoitetaan Arduinon Serial Monitor -väylään ”No doppler-frequency.” ja sammutetaan osoitinnäytöstä kaikki ledit.

6.3 Taajuuslaskuri

Doppler-kompassissa radiolähettimen taajuuden mittaamista varten tarvitaan tarkka vastaanotin. Tässä opinnäytetyössä taajuusarvo lähettimelle tai superheterodyne-vastaanottimen muodostamalle välitaajuudelle saadaan joko ohjelmallisesti simuloimalla tai mittaamalla funktiogeneraattorista. Funktiogeneraattorista saatu taajuus voidaan laskea taajuuslaskurin avulla, joka mittaa ajan, joka kuluu yhdessä signaalin jaksossa. Jaksonaika T tarkoittaa periodia, missä esimerkiksi kanttiaaltoinen signaali käy läpi tilanvaihdoksen ”ykkösestä” ”nollaan” tai ”nollasta” ”ykköseen”. Taajuus saadaan laskettua jaksonajasta kaavalla 10. (10.)

$$f = 1/T \qquad \textit{KAAVA 10}$$

$f =$ taajuus
$T =$ jaksonaika

Taajuuslaskurin hyödyntämän funktion nimi on FreqPeriodCounter ja sen on suunnitellut ja tehnyt Albert van Dalen. Aliohjelma, joka laskee taajuusarvon, näkyy kuvassa 17 ja 18 nimellä *counter.hertz()*. Kuvien 17 ja 18 ohjelmat keskiarvoistavat *counter.hertz()*-funktiosta saatuja tuloksia tarkentaakseen taajuuden mittausta. *counter.hertz()*-funktion mittaustarkkuus on teoriassa millihertsi-luokkaa, mutta käytännössä funktiogeneraattorilla testattuna kävi ilmi, että mittaustarkkuudessa on epätarkkuutta ja taajuustulokset heittelivät epämääräisesti. Taajuustulosten vaihtelua ilmeni varsinkin korkeammilla taajuuksilla (> 0,5 kHz). Kuvan 17 ohjelma mittaa taajuuden *fgen* (lähettimen alkuperäinen taajuus tai siitä muodostettu välitaajuus ilman doppler-vääristymää), jota käytetään referenssiarvona, kun selvitetään lähettimen suuntaa, ja kuvan 18 ohjelma mittaa taajuuden *fm* (lähettimen taajuus tai siitä muodostettu välitaajuus doppler-vääristymällä). (6, alaotsikko 9 Frequency counter library)

```
if(reset==true) //reset -> Tx [fgen] frequency measurement and averanging when still (without doppler shift)
{
while(i<10)//averages the frequency measurement [accuracy 10 frequency results]
{
if(counter.ready())
{
fgen=fgen+(double)counter.hertz(1000)/1000; //frequency accuracy in mHz .000 Hz
i++;
Serial << "Freq_value [fgen] " << i << ": " << fgen << "\n"; //shows the freq sums in serial monitor (while loop won't work without this command for some strange reason...)
}
else if(counter.elapsedTime > 50000) //detects if there is no frequency signal for 50ms
{
Serial << "No signal.\n";
ledsoff(); //turns off all leds
i=(byte)0;
fgen=0;
}
}
fgen=fgen/(double)i;
i=(byte)0;
reset=false;
//fgen=fgen*1000000; //converts to higher frequency [ex. MHz] to see the doppler shift with Due pwm signal or low frequency with function generator (BETA)
ft=dtow(fgen,c,v); //calculates the theoretical value of doppler frequency where Rx would be moving straight towards to Tx
dft=ft-fgen;
Serial << "-----------------------------------------------------------------------------------------------------------------------------------------\n";
}
```

KUVA 17. fgen-taajuuden mittaus ja keskiarvoistus

```
while(i<10) //frequency measurements and averaging when moving [accuracy 10 frequency results] (this code section is used with external frequency source)
{
if(counter.ready())
{
fm=fm+(double)counter.hertz(1000)/1000; //frequency accuracy in mHz .000 Hz
i++;
Serial << "Freq_value [fm] " << i << ": " << fm << "\n"; //shows the freq sums in serial monitor (while loop won't work without this command for some strange reason...)
}
else if(counter.elapsedTime > 50000) //detects if there is no frequency signal for 50ms
{
Serial << "No signal.\n";
ledsoff(); //turns off all leds
i=(byte)0;
fm=0;
}
}
fm=fm/(double)i;
//fm=fm*1000000; //convert to higher frequency [ex. MHz] to see the doppler shift with Due pwm signal or low frequency with function generator (BETA)
dfm=fm-fgen;
i=(byte)0;
Serial << "-----------------------------------------------------------------------------------------------------------------------------------------\n";
```

KUVA 18. fm-taajuuden mittaus ja keskiarvoistus

fgen-taajuuden mittaus suoritetaan, kun sekä lähetin että vastaanotin ovat paikallaan, ja tällöin myös lasketaan arvo muuttujalle *ft* (teoreettinen doppler-siirtymä, vastaanottimen liikkuessa suoraan lähetintä kohti) aliohjelmalla *dtow(fgen,c,v)*, jonka jälkeen mitataan vain arvoa *fm*. Suuntima-algoritmin superheterodyne-versiossa muuttujalle *ft* lasketaan arvo ennen pääohjelmaan siirtymistä muuttujan *forig* mukaan (*dtow(forig,c,v)*). *fgen*-taajuuden mittaus on asetettu ehtolauseen ”if(reset==true)” sisään, jotta käyttäjä voi päättää, milloin *fgen*-taajuuden arvo mitataan. Arduino Due -mikrokontrollerikortissa on vain yksi painike, jota painamalla koko ohjelmakoodin suoritus aloitetaan alusta, ja tällöin muuttuja *reset* saa arvon ”true”. Kun *fgen*:lle on mitattu arvo, annetaan muuttujalle *reset* arvo ”false”, jolloin pääsy *fgen*-taajuuden mittaukseen estyy ennen seuraavaa resetiä. (Kuva 17.)

”while(i<10)”-silmukka määrittää, kuinka monta yksittäistä tulosta mitataan eli kuinka tarkka taajuuden keskiarvoistamisesta tahdotaan. *fgen*- ja *fm*-taajuuksien tarkkuudeksi on asetettu tässä esimerkissä 10 tuloksen keskiarvo. Jostain kummallisesta syystä while-silmukka ei toiminut ja ohjelma jäi jumiin ilman käskyä, jossa kirjoitetaan jotain sarjaväylään. Siksi koodissa on käsky ”Serial << ”Freq_value” << i << ”: ” << fm << ”\n””, joka näyttää taajuusarvon välitulokset Serial Monitor -ikkunassa. Arduino Due -kortin pinni 3 on valittu taajuuden mittaamiseen ulkoisesta taajuuslähteestä doppler-suuntima-algoritmi-ohjelmassa. (Kuva 17 ja 18.)

6.4 Kierroslaskuri

vdf-muuttujan lukuarvo ei saa olla suurempi kuin yksi, jotta siitä voidaan ottaa arcuskosinifunktio kulman *facos* laskemista varten. Kierroslaskurikoodin tarkoituksena on muokata *vdf*:n arvoa sen ylittäessä luvun yksi. Laskuri vähentää alkuperäisestä *vdf*-arvosta ”taajuuskierrosten” lukumäärän *rcircled*:n silloin, kun *rcflag*-muuttuja sisältää arvon ”false” (*vdf*=*vdf*-*rcircled*). *rcflag*:n ollessa ”true” vähennetään arvosta 1 *vdf*:n ja *rcircled*:n erotus (*vdf*=1-(*vdf*-*rcircled*)). Yksi ”taajuuskierros” tarkoittaa sitä, kun *dfm*-muuttujan lukuarvo on suurempi kuin *dft*-muuttujan arvo, mutta *dfm* on silti pienempi kuin *dft* * 2. Kaksi ”taajuuskierrosta” tarkoittaa sitä, että *dfm*:n arvo on kaksinkertainen tai suurempi kuin *dft*:n arvo, mutta pienempi kuin *dft* * 3. Samalla logiikalla muodostuvat kaikki suuremmatkin kierrosmäärät. *rcflag*:n tilat ”true” ja ”false” vaihtelevat sen mukaan, kuinka monta kokonaista ”taajuuskierrosta” on tapahtunut. Ensimmäisen kierrosvaihdoksen jälkeen joka toisella kierrosmäärän vaihdoksella vaihdetaan tapaa, millä *vdf*:n uusi lukuarvo lasketaan. Kokonaisten kierrosvaihdoksien vaihtumisen havaitseminen on toteutettu siten, että kopioidaan koodin lopussa muuttujaan *rcc* muuttujan *rcircle* arvo (*rcircle* on *vdf*:n arvo ilman desimaleja). Tämän jälkeen katsotaan, onko *rcircle*:n arvo muuttunut käskyllä ”if(rcc!=rcircle&&rcircle>2)”, kun on palattu uudelleen kierroslaskurikoodiin. (Kuva 19.)

```
vdf=dfm/dft; //measured (or simulated) doppler shift in fm [dfm=fm-fgen] / calculated theoretical doppler shift in ft [dft=ft-fgen]
Serial << "dfm: " << dfm << " Hz, v: " << v << " m/s, fgen: " << fgen << " Hz, fm: " << fm << " Hz, ft: " << ft << " Hz, vdf: " << vdf << "\n";

if(vdf>1) //converts vdf-value when vdf>1 (ex. if (old)vdf=1.01 -> (new)vdf=0.01||0.99, this code changes the way how vdf-values are calculated)
{
Serial << "vdf_old: " << vdf << "\n";
rcircle=(int)vdf; //cycle count from double to int (how many full frequency cycles have happened (ex. dfm=100 & dft=50 -> two cycles)
rcircled=(double)rcircle; //cycle count from int to double (converts to full cycles only, no decimal numbers)
if(rcc!=rcircle&&rcircle>=2) //alternates the vdf-values calculation parametres on every second change when a full cycle happens (first cycle is not included)
rcflag=!rcflag;
if(rcflag==true)
vdf=1-(vdf-rcircled);
else
vdf=vdf-rcircled;
Serial << "rcircle: " << rcircle << ", rcircled: " << rcircled << ", vdf_new: " << vdf << "\n";
}

rcc=rcircle; //copies the value of rcircle to rcc, so that a full cycle change can be seen when rcircles value changes
```

KUVA 19. Kierroslaskurin koodi

Kierroslaskurikoodin tarpeellisuus ja oikeellisuus doppler-siirtymän ja lähettimen todellisen kulman laskemisessa on kyseenalainen ja se varsinaisesti perustuu vain itse muodostettuun ajatukseen. Luultavasti muuttujan *vdf* arvo ei voi fysikaalisista syistä nousta suuremmaksi kuin yksi, sillä *ft*:n eli lähettimen taajuuden teoreettinen suuruus, kun ollaan kulkemassa tarkalleen kohti lähetintä, on suurin mahdollinen taajuus doppler-ilmiössä, eikä näin ollen *fm* voi nousta sitä korkeammaksi. Toisaalta mittausvirhe tai jokin muu ohjelmallinen epätarkkuus saattaisi nostaa *fm*:n arvoa sen fysikaalista maksimia *ft*:tä suuremmaksi. Tämän esimerkin *fm*-muuttujan taajuus ei tule välitaajuusvastaanottimesta (superheterodyne), vaan tarkoittaa lähettimestä vastaanotettua todellista taajuutta liikkeessä. Tässä luvussa käytettyjen muuttujien määritykset löytyvät luvusta 6.1. (Kuva 19.)

Suunnanlaskennan koodiin on lisätty tiettyihin liitteistä löytyviin ohjelmaversioihin (liite 5, liite 6) ehtolause ”if(fm>fgen&&vdf<=1&&vdf>=0)”, jos tahdotaan esimerkiksi olla käyttämättä kierroslaskurikoodia. Tällöin ei sallita muuttujan *fm* arvon menevän yhtään fysikaalisen maksimin *ft*:n yli.

6.5 Osoitinnäyttö

Osoitinnäytön kontrolloinnin aliohjelma on esitetty kuvassa 20. Se hyödyntää mm. digitalWrite- ja shiftOut-komentoja. digitalWrite-käskyllä voidaan antaa halutulle pinnille tietty tila, HIGH tai LOW eli ”ykkönen” tai ”nolla”. shiftOut-käskyllä siirretään haluttu tavun kokoinen informaatio siirtorekisteripiireille (74HC595), jotta saadaan syttymään haluttu ledi väleillä –90°– –5,625° ja +90°–+5,625°.

```
void ledscr(double x) //fuction for controlling the led pointer display
{
if(x>-5.625&&x<5.625) //0' led17 on
{
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, HIGH); //disables IC2 outputs
digitalWrite(led17Pin, HIGH); //set 0' led on
digitalWrite(led18Pin, LOW); //set 180'/backTx led off
}
else
{
digitalWrite(led17Pin, LOW); //set 0' led off
digitalWrite(led18Pin, LOW); //set 180'/backTx led off
if(x>0&&x>=5.625) //IC2 enable
{
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, LOW); //enables IC2 outputs
}
else if(x<0&&x<=-5.625)//IC1 enable
{
digitalWrite(oePin2, HIGH); //disables IC2 outputs
digitalWrite(oePin1, LOW); //enables IC1 outputs
}
else
{
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, HIGH); //disables IC2 outputs
}
digitalWrite(latchPin, LOW); //latch pin low so led's don't change while sending bits
if(x<0)
{
if(x>=-90.000&&x<=-84.375) //LED1 -90' [leds 1-8 negat.']
shiftOut(dataPin, clockPin, MSBFIRST, 0x01);
else if(x>-84.375&&x<=-73.125) //LED2 -78.75'
shiftOut(dataPin, clockPin, MSBFIRST, 0x02);
else if(x>-73.125&&x<=-61.875) //LED3 -67.5'
shiftOut(dataPin, clockPin, MSBFIRST, 0x04);
else if(x>-61.875&&x<=-50.625) //LED4 -56.25'
shiftOut(dataPin, clockPin, MSBFIRST, 0x08);
else if(x>-50.652&&x<=-39.375) //LED5 -45'
shiftOut(dataPin, clockPin, MSBFIRST, 0x10);
else if(x>-39.375&&x<=-28.125) //LED6 -33.75'
shiftOut(dataPin, clockPin, MSBFIRST, 0x20);
else if(x>-28.125&&x<=-16.875) //LED7 -22.5'
shiftOut(dataPin, clockPin, MSBFIRST, 0x40);
else if(x>-16.875&&x<=-5.625) //LED8 -11.25'
shiftOut(dataPin, clockPin, MSBFIRST, 0x80);
}
else
{
if(x<16.875&&x>=5.625) //LED9 +11.25' [leds 9-16 posit.']
shiftOut(dataPin, clockPin, MSBFIRST, 0x01);
else if(x<28.125&&x>=16.875) //LED10 +22.5'
shiftOut(dataPin, clockPin, MSBFIRST, 0x02);
else if(x<39.375&&x>=28.125) //LED11 +33.75'
shiftOut(dataPin, clockPin, MSBFIRST, 0x04);
else if(x<50.652&&x>=39.375) //LED12 +45'
shiftOut(dataPin, clockPin, MSBFIRST, 0x08);
else if(x<61.875&&x>=50.625) //LED13 +56.25'
shiftOut(dataPin, clockPin, MSBFIRST, 0x10);
else if(x<73.125&&x>=61.875) //LED14 +67.5'
shiftOut(dataPin, clockPin, MSBFIRST, 0x20);
else if(x<84.375&&x>=73.125) //LED15 +78.75'
shiftOut(dataPin, clockPin, MSBFIRST, 0x40);
else if(x<=90.000&&x>=84.375) //LED16 +90'
shiftOut(dataPin, clockPin, MSBFIRST, 0x80);
}
digitalWrite(latchPin, HIGH); //latch pin high so leds will light up
}
}
```

KUVA 20. Funktio osoitinnäytön ohjaamiseen

ledscr(x)-funktio saa parametrina *facos*-muuttujan eli lähettimen kulman informaation pääohjelmasta. Jos kulma on pienempi kuin +5,625° ja suurempi kuin

–5,625°, sytytetään astetta 0 osoittava ledi (LED17), jolloin estetään kummankin siirtorekisteripiirin lähdöt sekä sammutetaan taaksepäin osoittava ledi (LED18). Taaksepäin osoittavan ledin sytyttäminen tapahtuu pääohjelmassa, kun *fm* on pienempi kuin *fgen.* (Kuva 20.)

Kummallekin 74HC595-piirille siirretään aina sama data, mutta pinneillä *oePin1* sekä *oePin2* voidaan määrätä, kumman piirin lähdöt sallitaan, sen mukaan, onko kyseessä negatiivinen vai positiivinen aste. *oePin1* hallitsee negatiivisia ja *oePin2* positiivisia kulmia. Kun tahdotaan sytyttää jokin piirien ohjaamista ledeistä, ensiksi estetään sen piirin lähdöt, jonka kontrolloimia ledejä ei tahdota sytyttää, ja sallitaan sen piirin lähdöt, jossa sytytettävä ledi sijaitsee. *dataPin*- ja *clockPin*-pinnien kautta siirretään ja ajoitetaan haluttu data siirtorekisteripiireille. *latchPin*-pinnillä määrätään, milloin piireille siirretty informaatio sytyttää tietyn ledin, jotta ledit eivät vilkkuisi datan siirtämisen aikana. (Kuva 20.)

Kun tahdotaan sammuttaa osoitinnäytöstä kaikki ledit, hypätään aliohjelmaan *ledsoff()*, joka estää kummankin siirtorekisteripiirin lähdöt sekä asettaa pinnit *led17Pin* ja *led18Pin* nollaksi. (Kuva 21.)

```
void ledsoff() //function to turn all leds off
{
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, HIGH); //disables IC2 outputs
digitalWrite(led17Pin, LOW); //set 0' led off
digitalWrite(led18Pin, LOW); //set 180'/backTx led off
}
```

KUVA 21. Funktio kaikkien ledien sammuttamiseen

7 SUUNTIMA-ALGORITMIN TESTAUS

Tässä luvussa käydään läpi doppler-suuntima-algoritmiohjelman määrittelemättömälle vastaanottimelle suunnitellun version testaus. Muuttujat *fgen* ja *fm* mittaavat (tai saavat ohjelmallisesti) suoraan simuloidun lähettimen taajuutta eikä esimerkiksi superheterodyne-vastaanottimen välitaajuutta. Suuntima-algoritmin testaus on suoritettu sekä funktiogeneraattorilla että ohjelmallisesti muuttujille (*dfm*, *fgen*) arvoja sijoittamalla. Nopeudelle *v* annetaan ohjelmallisesti arvo kummassakin testausmuodossa. Mitatut taajuusarvot ja muut muuttujien arvot on kerätty käyttäen Serial Monitor -ikkunaa Arduino-ohjelmistossa. Arduino Due -piirikortin pinni 3 toimii ulkoisen taajuuslähteen mittaamisen tulopinninä. Tässä luvussa käytetyt muuttujat on esitetty luvussa 6.1. Liitteet 2, 3 ja 4 sisältävät ohjelman eri testausversiot.

7.1 Testaus funktiogeneraattorilla

Ensiksi testattiin taajuuslaskurikoodin tarkkuus syöttämällä funktiogeneraattorilla Arduino Due -mikrokontrollerikortin pinniin 3 kanttiaaltoista signaalia taajuusarvoilla 100 Hz, 200 Hz, 300 Hz, 500 HZ, 1000 Hz, 2000 Hz, 3000 Hz ja 4000 Hz sekä tarkkailemalla taajuuslaskurin muodostamaa kymmentä eri taajuustulosta näistä vakiotaajuuksista. Jokaiselle funktiogeneraattorilla annetulle taajuudelle on vastaavat muuttujat taajuuslaskurin mittaustuloksille taulukossa 1 (*fm 1*, *fm 2*, *fm 3*, *fm 4*, *fm 5*, *fm 6*, *fm 7*, *fm 8*). Funktiogeneraattorin syöttämä taajuus on mitattu oskilloskoopilla ja se pysyy vakiona eikä siinä tapahdu heittelyä. (Taulukko 1.)

Toisena testausvuorossa on lähettimen suunnan paikannustesti, jossa säädetään funktiogeneraattorilla taajuutta *fm* siten, että saadaan lähettimen suunnan positiiviset kulmat käytyä läpi mahdollisimman tarkasti. Jotta taajuuslaskurin tarkkuus doppler-siirtymän mittauksessa olisi tarkin mahdollinen, pitää testaus suorittaa matalalla noin 100 Hz:n taajuusalueella. Koska testaustaajuus eli lähettimen taajuus ilman doppler-siirtymää *fgen* on hyvin matala (*fgen* = 99,837 Hz), täytyy nopeus *v* asettaa erittäin suureksi (5555555,556 m/s = 20 000 000 km/h), jotta voidaan huomata riittävän tarkasti

doppler-vääristymä taajuudesta *fm*. Taajuuslaskuri keskiarvoistaa jokaisen mitatun taajuustuloksen 50 välituloksella. (Taulukko 2.)

TAULUKKO 1. Taajuuslaskurin tarkkuustestin mittaustulokset

Taajuuslaskurin tarkkuustesti (keskiarvoistettu 50 välituloksella) [kanttiaaltoinen signaali]								
Arvot funktiogeneraattorista	**100 Hz**	**200 Hz**	**300 Hz**	**500 Hz**	**1000 Hz**	**2000 Hz**	**3000 Hz**	**4000 Hz**
Muuttujat taajuuslaskurin mittaustuloksille	***fm 1 [Hz]***	***fm 2 [Hz]***	***fm 3 [Hz]***	***fm 4 [Hz]***	***fm 5 [Hz]***	***fm 6 [Hz]***	***fm 7 [Hz]***	***fm 8 [Hz]***
Tulos 1	99,604	202,383	300,794	503,461	1031,904	1930,555	2396,789	2146,526
Tulos 2	99,888	201,425	299,791	503,726	1031,501	1791,520	2170,587	2143,996
Tulos 3	99,664	202,387	300,031	501,101	1034,246	1881,104	2413,426	2158,653
Tulos 4	99,791	201,429	298,842	503,402	1046,055	1933,080	2033,861	2196,372
Tulos 5	99,915	202,147	298,770	502,864	985,690	1949,105	2286,067	2223,754
Tulos 6	100,015	201,271	299,270	498,421	1039,671	1955,127	2442,424	2128,287
Tulos 7	99,823	201,471	299,258	508,418	1013,998	1867,215	2165,939	2210,316
Tulos 8	99,834	201,187	298,769	497,401	1015,807	1860,125	2421,656	1917,512
Tulos 9	99,708	200,712	300,140	505,005	1023,044	1870,420	2473,431	2322,956
Tulos 10	99,766	200,643	301,352	508,585	1004,992	1898,732	2367,698	2115,696

Taulukosta 1 voidaan huomata, että vaikka tiettyä taajuusarvoa pidetään vakiona funktiogeneraattorilla, tapahtuu taajuuslaskurin laskemana siinä suurta heittelyä, varsinkin korkeammissa taajuuksissa (> 0,5 kHz). Suurin laskurin mittaustarkkuus saadaan matalilla taajuuksilla. 100 Hz:n signaali heittelee välillä 99,604–100,015 Hz taajuuslaskurilla mitattuna (*fm 1*). Mitä korkeammaksi taajuus funktiogeneraattorista asetetaan (100–4000 Hz), sitä epätarkemmaksi ja heittelevämmäksi taajuuslaskurin laskemat taajuusarvot (*fm 1 – fm 8*) menevät. 2000 Hz:n eli *fm 6* -muuttujan arvojen kohdalla huomataan, että taajuuslaskuri on jo täysin epätarkka sekä sen antamat taajuusarvot heittelevät suuresti. 3000 Hz:stä eteenpäin laskurin tarkkuus on aivan käyttökelvotonta sen antaessa tuloksia väliltä 2165,939–2473,431 Hz (*fm 7*).

TAULUKKO 2. Lähettimen suunnan paikannustestin mittaustulokset

Lähettimen suunnan paikannustesti funktiogeneraattorilla [positiiviset asteet]				
- keskiarvoistettu 50 välituloksella - kanttiaaltoinen signaali				
***fgen*:** 99,837 Hz ***v*:** 5555555,556 m/s = 20 000 000 km/h ***ft*:** 101,687 Hz ***dft*:** 1,850 Hz				
	***fm* [Hz]**	***dfm* [Hz]**	***vdf***	***facos* [°]**
Tulos 1	99,904	0,066	0,035	87,955
Tulos 2	99,972	0,134	0,072	85,846
Tulos 3	100,075	0,237	0,128	82,639
Tulos 4	100,160	0,322	0,174	79,976
Tulos 5	100,221	0,384	0,207	78,020
Tulos 6	100,318	0,480	0,259	74,962
Tulos 7	100,327	0,489	0,264	74,673
Tulos 8	100,456	0,618	0,334	70,484
Tulos 9	100,499	0,662	0,357	69,032
Tulos 10	100,542	0,705	0,381	67,599
Tulos 11	100,568	0,730	0,394	66,759
Tulos 12	100,603	0,765	0,413	65,574
Tulos 13	100,700	0,863	0,466	62,193
Tulos 14	100,760	0,922	0,498	60,107
Tulos 15	100,818	0,980	0,529	58,012
Tulos 16	100,851	1,013	0,547	56,800
Tulos 17	100,855	1,017	0,549	56,651
Tulos 18	100,921	1,083	0,585	54,168
Tulos 19	100,964	1,126	0,608	52,508
Tulos 20	101,007	1,169	0,631	50,810
Tulos 21	101,050	1,212	0,655	49,070
Tulos 22	101,117	1,279	0,691	46,262
Tulos 23	101,145	1,307	0,706	45,050
Tulos 24	101,193	1,355	0,732	42,909
Tulos 25	101,233	1,395	0,754	41,057
Tulos 26	101,284	1,446	0,781	38,590
Tulos 27	101,317	1,479	0,799	36,921
Tulos 28	101,339	1,501	0,811	35,771
Tulos 29	101,358	1,520	0,821	34,752
Tulos 30	101,405	1,568	0,847	32,051
Tulos 31	101,452	1,614	0,872	29,257
Tulos 32	101,481	1,643	0,888	27,363
Tulos 33	101,494	1,656	0,895	26,474

taulukko 2 jatkuu

taulukko 2 jatkuu

Tulos 34	101,520	1,683	0,909	24,531
Tulos 35	101,540	1,705	0,921	22,835
Tulos 36	101,568	1,730	0,935	20,750
Tulos 37	101,569	1,732	0,936	20,574
Tulos 38	101,591	1,753	0,947	18,636
Tulos 39	101,601	1,763	0,952	17,641
Tulos 40	101,609	1,772	0,957	16,696
Tulos 41	101,632	1,794	0,969	14,133
Tulos 42	101,634	1,796	0,970	13,877
Tulos 43	101,642	1,804	0,975	12,803
Tulos 44	101,656	1,818	0,982	10,672
Tulos 45	101,662	1,824	0,985	9,617
Tulos 46	101,669	1,832	0,990	7,999
Tulos 47	101,671	1,833	0,990	7,773
Tulos 48	101,679	1,841	0,995	5,653

Taulukosta 2 huomataan, että joitain asteita *facos* jää 90°–0°:n välistä. Tämä johtuu siitä, että taajuuslaskurin laskema taajuusarvo *fm* heittelehti, vaikka taajuutta pidettiin funktiogeneraattorissa vakiona. Mittaustulokset ovat muodostuneet siten, että funktiogeneraattorista on varovaisesti nostettu taajuutta ja odotettu sitten jonkin aikaa, että saadaan halutun kulman muuttujien tiedot ylös. Odottelutuokio aiheutuu edellä mainitusta laskurin epätarkkuudesta ja heittelystä, jolloin taajuuden *fm* ja täten myös kulman *facos* (*facos* = *arccos* (*dfm*/*dft*)) arvot eivät pysy vakaana funktiogeneraattorin taajuuden tavoin. Muuttujat *ft* sekä *dft* on laskettu ensimmäiseksi mitatun taajuuden *fgen* ja ohjelmallisesti asetetun nopeuden *v* perusteella. Negatiiviset kulmat on jätetty mittaamatta, sillä se olisi ollut lähes mahdotonta käsin funktiogeneraattorilla. Negatiiviset kulmat aktivoituvat, jos suuntima-algoritmin seuraavan laskentakierroksen *dfm*-muuttujan arvo on pienempi kuin edellisellä kierroksella sekä kulma *facos* on pienempi kuin 45°. Negatiivisten kulmien testaus käydään läpi ohjelmallisesti seuraavassa luvussa.

7.2 Testaus ohjelmallisesti

Ohjelmallisesti testattaessa mikrokontrollerille ei syötetä ollenkaan signaalia ulkopuolisesta lähteestä, vaan kaikki muuttujien arvot muodostuvat ohjelmassa itsessään, tai on annettu käsin alkuasetuksissa. Muuttujat, joille täytyy asettaa arvot, jotta saadaan laskettua lähettimelle suunta, ovat lähettimen alkuperäinen

taajuus ilman doppler-vääristymää *fgen*, vastaanottimen liikesuunnassa mitatun (simuloidun) taajuuden doppler-vääristymä *dfm* sekä vastaanottimen liikenopeus *v*. Liikesuunnassa mitattu taajuus *fm* voidaan laskea muuttujan *dfm* perusteella (*fm=fgen+dfm*). Algoritmi ei laske taaksepäin osoittavia kulmia, vaan ilmoittaa käyttäjälle väärän suunnan silloin, kun *fm* on pienempi kuin *fgen*.

Ensiksi testataan positiiviset kulmat +90°–+0° inkrementoimalla muuttujan *dfm* lukuarvoa 5 mHz joka laskentakierroksella. *dfm*:n alkuarvo on 1 mHz. Kulmien *facos* laskeminen aloitetaan noin +90°:sta ja päätetään noin 0°:seen, siten, että mittaustuloksiin on merkitty vain ne kulmat, jolloin asteen kokonaisluku vaihtuu seuraavaan. Esimerkiksi jos aloitetaan 89,xxx°:sta, seuraava mittaustuloksiin otettu kulma on 88,xxx°, eli kaikki osa-asteet ennen vaihdosta 89°–88° on jätetty merkitsemättä tuloksiin. Testin parametrit ovat *v* = 2,778 m/s = 10 km/h, *fgen* = 900 MHz, *ft* = 900000008,339 Hz, *dft* = 8,339 Hz. (Taulukko 3.)

Negatiiviset kulmat –90°– –0° testataan inkrementoimalla *dfm*:n lukuarvoa 5 mHz ja dekrementoimalla sitä 1 µHz joka laskentakierroksella. *dfm*:n alkuarvo on 1 mHz. *fm*:n taajuusarvo muuttuu lineaarisesti *dfm*:n mukana. Negatiivisia kulmia **facos* aletaan laskemaan vasta, kun *facos* alittaa 45°:n kulman ja muuttuja *dfm* alenee edellisen laskentakierroksen arvostaan. *dfm*:n lukuarvoa täytyy dekrementoida ennen uuden kulman laskentaoperaatiota ja inkrementoida vasta sen jälkeen, jotta negatiiviset kulmat saadaan ilmentymään. Vähentämisen suuruus (1 µHz) on pieni suhteessa kasvattamiseen (5 mHz), jotta asteet laskisivat ja kaikki tulokset saataisiin ohjelmassa käytyä läpi. Testin parametrit ovat *v* = 2,778 m/s = 10 km/h, *fgen* = 900 MHz, *ft* = 900000008,339 Hz, *dft* = 8,339 Hz. (Taulukko 4.)

TAULUKKO 3. Positiivisten kulmien paikannustestin mittaustulokset

Ohjelmallinen lähettimen suunnan paikannustesti [positiiviset asteet]				
- positiivisten asteiden asetus: *dfm*:n arvoa inkrementoidaan 5 mHz joka laskentakierroksella - otettu huomioon vain tulokset, kun kulman *facos* kokonaisluku vaihtuu				
***fgen*:** 900 MHz ***v*:** 2,778 m/s = 10 km/h ***ft*:** 900000008,339 Hz ***dft*:** 8,339 Hz				
	***fm* [Hz]**	***dfm* [Hz]**	***vdf***	***facos* [°]**
Tulos 1	900000000,001	0,001	0,000	89,993
Tulos 2	900000000,146	0,146	0,017	88,996
Tulos 3	900000000,296	0,296	0,035	87,965
Tulos 4	900000000,440	0,441	0,052	86,968
Tulos 5	900000000,585	0,586	0,070	85,970
Tulos 6	900000000,730	0,731	0,087	84,971
Tulos 7	900000000,876	0,876	0,105	83,970
Tulos 8	900000001,021	1,021	0,122	82,967
Tulos 9	900000001,161	1,160	0,139	81,997
Tulos 10	900000001,305	1,305	0,156	80,990
Tulos 11	900000001,450	1,450	0,173	79,980
Tulos 12	900000001,595	1,595	0,191	78,967
Tulos 13	900000001,735	1,735	0,208	77,985
Tulos 14	900000001,881	1,880	0,225	76,965
Tulos 15	900000002,021	2,020	0,242	75,975
Tulos 16	900000002,161	2,160	0,259	74,982
Tulos 17	900000002,300	2,300	0,275	73,983
Tulos 18	900000002,440	2,440	0,292	72,980
Tulos 19	900000002,580	2,580	0,309	71,972
Tulos 20	900000002,715	2,715	0,325	70,993
Tulos 21	900000002,855	2,855	0,342	69,973
Tulos 22	900000002,991	2,990	0,358	68,983
Tulos 23	900000003,126	3,125	0,374	67,986
Tulos 24	900000003,261	3,260	0,391	66,982
Tulos 25	900000003,396	3,395	0,407	65,970
Tulos 26	900000003,526	3,525	0,422	64,988
Tulos 27	900000003,656	3,655	0,438	63,999
Tulos 28	900000003,791	3,790	0,454	62,962
Tulos 29	900000003,916	3,915	0,469	61,994
Tulos 30	900000004,046	4,045	0,485	60,978
Tulos 31	900000004,171	4,170	0,500	59,991
Tulos 32	900000004,296	4,295	0,515	58,994
Tulos 33	900000004,421	4,420	0,530	57,987
Tulos 34	900000004,546	4,545	0,545	56,968

taulukko 3 jatkuu

taulukko 3 jatkuu

Tulos 35	900000004,666	4,665	0,559	55,979
Tulos 36	900000004,786	4,785	0,573	54,978
Tulos 37	900000004,906	4,905	0,588	53,965
Tulos 38	900000005,021	5,020	0,602	52,982
Tulos 39	900000005,136	5,135	0,615	51,986
Tulos 40	900000005,251	5,250	0,629	50,976
Tulos 41	900000005,360	5,360	0,642	49,997
Tulos 42	900000005,475	5,475	0,656	48,957
Tulos 43	900000005,580	5,580	0,669	47,994
Tulos 44	900000005,690	5,690	0,682	46,969
Tulos 45	900000005,796	5,795	0,694	45,974
Tulos 46	900000005,901	5,900	0,707	44,962
Tulos 47	900000006,001	6,000	0,719	43,981
Tulos 48	900000006,100	6,100	0,731	42,983
Tulos 49	900000006,200	6,200	0,743	41,965
Tulos 50	900000006,296	6,295	0,754	40,980
Tulos 51	900000006,391	6,390	0,766	39,974
Tulos 52	900000006,485	6,485	0,777	38,947
Tulos 53	900000006,575	6,575	0,788	37,953
Tulos 54	900000006,661	6,660	0,798	36,993
Tulos 55	900000006,751	6,750	0,809	35,953
Tulos 56	900000006,835	6,835	0,819	34,946
Tulos 57	900000006,916	6,915	0,829	33,975
Tulos 58	900000006,996	6,995	0,838	32,978
Tulos 59	900000007,075	7,075	0,848	31,954
Tulos 60	900000007,151	7,150	0,857	30,967
Tulos 61	900000007,225	7,225	0,866	29,951
Tulos 62	900000007,296	7,295	0,874	28,973
Tulos 63	900000007,366	7,365	0,883	27,964
Tulos 64	900000007,430	7,430	0,891	26,996
Tulos 65	900000007,496	7,495	0,898	25,995
Tulos 66	900000007,560	7,560	0,906	24,957
Tulos 67	900000007,621	7,620	0,913	23,962
Tulos 68	900000007,680	7,680	0,921	22,926
Tulos 69	900000007,735	7,735	0,927	21,935
Tulos 70	900000007,786	7,785	0,933	20,997
Tulos 71	900000007,840	7,840	0,940	19,915
Tulos 72	900000007,886	7,885	0,945	18,987
Tulos 73	900000007,935	7,935	0,951	17,901
Tulos 74	900000007,975	7,975	0,956	16,984
Tulos 75	900000008,021	8,020	0,961	15,892
Tulos 76	900000008,055	8,055	0,965	14,989
Tulos 77	900000008,095	8,095	0,970	13,887
Tulos 78	900000008,131	8,130	0,974	12,847

taulukko 3 jatkuu

taulukko 3 jatkuu

Tulos 79	900000008,161	8,160	0,978	11,884
Tulos 80	900000008,190	8,190	0,982	10,838
Tulos 81	900000008,215	8,215	0,985	9,883
Tulos 82	900000008,241	8,240	0,988	8,826
Tulos 83	900000008,261	8,260	0,990	7,881
Tulos 84	900000008,281	8,280	0,992	6,805
Tulos 85	900000008,296	8,295	0,994	5,872
Tulos 86	900000008,310	8,310	0,996	4,760
Tulos 87	900000008,320	8,320	0,997	3,844
Tulos 88	900000008,330	8,330	0,998	2,627
Tulos 89	900000008,335	8,335	0,999	1,722

Taulukossa 3 on mittaustulokset positiivisille kulmille (*facos*) kolmen desimaalin tarkkuudella. Muuttujan *vdf* arvo määrää suoraan kulman asteluvun (*facos* = arccos (*dfm*/*dft*) = arccos (*vdf*)). Taulukosta huomataan, että lähettimen lähetystaajuuden *fgen* ollessa 900 MHz ja vastaanottimen nopeuden *v* ollessa 10 km/h tarvitaan suuremmissa kulmissa yhden asteen *facos* eroon noin 150 mHz:n suuruinen taajuus *fm* (tulos 1 – tulos 25). Esimerkiksi kulman 89,993° (tulos 1) *fm*:n arvo on 900000000,001 Hz ja kulman 88,996° (tulos 2) *fm*:n arvo on 900000000,146 Hz. Näiden erotus on noin $|150|$ mHz. Kulmien (*facos*) pienentyessä kohti arvoa 0° pienenee myös *fm*:n arvojen erotuksen suuruus aina seuraavaan kulmaan nähden. Viimeisistä tuloksista (tulos 88 – tulos 89) nähdään, että noin asteen muutos kulmassa *facos* muodostuu vain 5 mHz:n erosta *fm*:n taajuudessa.

TAULUKKO 4. Negatiivisten kulmien paikannustestin mittaustulokset

Ohjelmallinen lähettimen suunnan paikannustesti [negatiiviset asteet]					
- negatiivisten asteiden asetus: *dfm*:n arvoa inkrementoidaan 5 mHz ja dekrementoidaan 1 µHz joka laskentakierroksella - otettu huomioon vain tulokset, kun kulman **facos* kokonaisluku vaihtuu					
***fgen*:** 900 MHz ***ft*:** 900000008,339 Hz ***dft*:** 8,339 Hz ***v*:** 2,778 m/s = 10 km/h					
	***fm* [Hz]**	***dfm* [Hz]**	***vdf***	***facos* [°]**	***facos* [°]**
Tulos 1	900000005,899	5,899	0,707	44,973	–89,947
Tulos 2	900000005,949	5,949	0,713	44,485	–88,971
Tulos 3	900000005,999	5,999	0,719	43,993	–87,986
Tulos 4	900000006,049	6,049	0,725	43,496	–86,993
Tulos 5	900000006,099	6,099	0,731	42,995	–85,990
Tulos 6	900000006,149	6,149	0,737	42,489	–84,978
Tulos 7	900000006,199	6,199	0,743	41,978	–83,956
Tulos 8	900000006,249	6,249	0,749	41,462	–82,924
Tulos 9	900000006,294	6,294	0,754	40,993	–81,986
Tulos 10	900000006,344	6,344	0,760	40,466	–80,933
Tulos 11	900000006,389	6,389	0,766	39,988	–79,976
Tulos 12	900000006,439	6,439	0,772	39,450	–78,901
Tulos 13	900000006,484	6,484	0,777	38,961	–77,923
Tulos 14	900000006,529	6,529	0,782	38,467	–76,935
Tulos 15	900000006,574	6,574	0,788	37,968	–75,936
Tulos 16	900000006,619	6,619	0,793	37,462	–74,925
Tulos 17	900000006,664	6,664	0,799	36,951	–73,903
Tulos 18	900000006,704	6,704	0,803	36,492	–72,984
Tulos 19	900000006,749	6,749	0,809	35,969	–71,938
Tulos 20	900000006,789	6,789	0,814	35,498	–70,997
Tulos 21	900000006,834	6,834	0,819	34,962	–69,925
Tulos 22	900000006,874	6,874	0,824	34,480	–68,961
Tulos 23	900000006,914	6,914	0,829	33,992	–67,984
Tulos 24	900000006,954	6,954	0,833	33,497	–66,995
Tulos 25	900000006,994	6,994	0,838	32,996	–65,992
Tulos 26	900000007,034	7,034	0,843	32,488	–64,976
Tulos 27	900000007,074	7,074	0,848	31,973	–63,946
Tulos 28	900000007,114	7,114	0,853	31,450	–62,901
Tulos 29	900000007,149	7,149	0,857	30,986	–61,973
Tulos 30	900000007,189	7,189	0,862	30,448	–60,897
Tulos 31	900000007,224	7,224	0,866	29,971	–59,942
Tulos 32	900000007,259	7,259	0,870	29,486	–58,972
Tulos 33	900000007,294	7,294	0,874	28,994	–57,988
Tulos 34	900000007,329	7,329	0,878	28,494	–56,988

taulukko 4 jatkuu

taulukko 4 jatkuu

Tulos 35	900000007,364	7,364	0,883	27,986	–55,972
Tulos 36	900000007,399	7,399	0,887	27,469	–54,938
Tulos 37	900000007,434	7,434	0,891	26,943	–53,887
Tulos 38	900000007,464	7,464	0,895	26,485	–52,970
Tulos 39	900000007,499	7,499	0,899	25,940	–51,881
Tulos 40	900000007,529	7,529	0,902	25,465	–50,931
Tulos 41	900000007,559	7,559	0,906	24,982	–49,964
Tulos 42	900000007,589	7,589	0,910	24,489	–48,979
Tulos 43	900000007,619	7,619	0,913	23,987	–47,975
Tulos 44	900000007,649	7,649	0,917	23,475	–46,951
Tulos 45	900000007,679	7,679	0,920	22,953	–45,906
Tulos 46	900000007,709	7,709	0,924	22,418	–44,837
Tulos 47	900000007,734	7,734	0,927	21,964	–43,928
Tulos 48	900000007,764	7,764	0,931	21,406	–42,812
Tulos 49	900000007,789	7,789	0,934	20,930	–41,861
Tulos 50	900000007,814	7,814	0,937	20,444	–40,889
Tulos 51	900000007,839	7,839	0,940	19,947	–39,894
Tulos 52	900000007,864	7,864	0,943	19,437	–38,875
Tulos 53	900000007,889	7,889	0,946	18,915	–37,830
Tulos 54	900000007,909	7,909	0,948	18,486	–36,973
Tulos 55	900000007,934	7,934	0,951	17,937	–35,874
Tulos 56	900000007,954	7,954	0,953	17,485	–34,971
Tulos 57	900000007,979	7,979	0,956	16,904	–33,809
Tulos 58	900000007,999	7,999	0,959	16,425	–32,851
Tulos 59	900000008,019	8,019	0,961	15,932	–31,865
Tulos 60	900000008,039	8,039	0,963	15,424	–30,848
Tulos 61	900000008,059	8,059	0,966	14,899	–29,798
Tulos 62	900000008,074	8,074	0,968	14,492	–28,985
Tulos 63	900000008,094	8,094	0,970	13,933	–27,866
Tulos 64	900000008,109	8,109	0,972	13,498	–26,997
Tulos 65	900000008,129	8,129	0,974	12,897	–25,794
Tulos 66	900000008,144	8,144	0,976	12,427	–24,854
Tulos 67	900000008,159	8,159	0,978	11,938	–23,877
Tulos 68	900000008,174	8,174	0,980	11,430	–22,860
Tulos 69	900000008,189	8,189	0,981	10,898	–21,796
Tulos 70	900000008,204	8,204	0,983	10,338	–20,677
Tulos 71	900000008,214	8,214	0,984	9,948	–19,897
Tulos 72	900000008,229	8,229	0,986	9,333	–18,667
Tulos 73	900000008,239	8,239	0,987	8,900	–17,800
Tulos 74	900000008,249	8,249	0,989	8,444	–16,889
Tulos 75	900000008,259	8,259	0,990	7,963	–15,926
Tulos 76	900000008,269	8,269	0,991	7,451	–14,902
Tulos 77	900000008,279	8,279	0,992	6,901	–13,802
Tulos 78	900000008,289	8,289	0,993	6,303	–12,607

taulukko 4 jatkuu

taulukko 4 jatkuu

Tulos 79	900000008,294	8,294	0,994	5,983	–11,966
Tulos 80	900000008,304	8,304	0,995	5,283	–10,566
Tulos 81	900000008,309	8,309	0,996	4,896	–9,792
Tulos 82	900000008,314	8,314	0,996	4,475	–8,951
Tulos 83	900000008,324	8,324	0,998	3,486	–6,972
Tulos 84	900000008,329	8,329	0,998	2,866	–5,733
Tulos 85	900000008,334	8,334	0,999	2,068	–4,137
Tulos 86	900000008,339	8,339	0,999	0,586	–1,173

Taulukossa 4 on mittaustulokset negatiivisille kulmille (**facos*) kolmen desimaalin tarkkuudella. Negatiiviset kulmat muodostuvat, kun ensin laskettu positiivinen kulma *facos* kerrotaan –2:lla. Tämä korjaustoiminto tapahtuu sen vuoksi, että suuntima-algoritmi huomaa alkuperäisen lasketun kulkusuunnan olevan väärä, koska *dfm*:n arvo on pienentynyt edellisestä laskentakierroksesta. *dfm* ei voi laskea, jos vastaanotin etenee kohti lähetintä. Taulukosta huomataan, että lähettimen lähetystaajuuden *fgen* ollessa 900 MHz ja vastaanottimen nopeuden *v* ollessa 10 km/h tarvitaan suuremmissa kulmissa yhden asteen **facos* eroon noin 50 mHz:n suuruinen taajuus *fm* (tulos 1 – tulos 25). Esimerkiksi kulman –89,947° (tulos 1) *fm*:n arvo on 900000005,899 Hz ja kulman –88,971° (tulos 2) *fm*:n arvo on 900000005,949 Hz. Näiden erotus on noin $|50|$ mHz. Negatiivisten kulmien **facos* lähestyessä arvoa 0° pienenee myös *fm*:n arvojen erotuksen suuruus aina seuraavaan kulmaan nähden. Viimeisistä tuloksista (tulos 85 – tulos 86) nähdään, että noin kolmen asteen muutos kulmassa **facos* muodostuu vain 5 mHz:n erosta *fm*:n taajuudessa. Negatiivisten kulmien laskemiseen käytettiin samoja muuttujien arvoja kuin positiivisten kulmien laskemiseen. Tämän vuoksi kulmien välisten taajuuksien (*fm*) ero saa aikaan suuremman muutoksen kulmassa **facos* kuin *facos* (**facos=facos**-2).

8 SUUNTIMA-ALGORITMIN SUPERHETERODYNE-VERSIO

Tässä luvussa käydään läpi superheterodyne-versio suuntima-algoritmi-ohjelmasta. Välitaajuusvastaanotin vähentää runsaasti algoritmin taajuuslaskurin ja Doppler-kompassin vastaanottimen tarkkuusvaatimuksia. Tämän vuoksi ohjelman superheterodyne-versio antaa hyvän näkökohdan laitteen hyödyntämiseen käytännössä. Liitteet 5, 6 ja 7 sisältävät suuntima-algoritmin superheterodyne-vastaanotinversion erilaiset testausversiot.

8.1 Muutoksia algoritmissa

Superheterodyne-vastaanotin madaltaa lähettimestä saatavan taajuuden, jotta sitä olisi helpompi jatkokäsitellä. Doppler-vääristymä siirtyy välitaajuuteen samansuuruisena, kuin jos se mitattaisiin suoraan lähettimen taajuudesta.

Doppler-suuntima-algoritmiin on lisätty yksi double-tietotyyppinen muuttuja *forig*. Muuttuja toimii lähettimen taajuusarvona ilman doppler-siirtymää (esim. 2,4 GHz). Tätä muuttujaa (*forig*) hyödyntäen lasketaan *ft* eli teoreettinen doppler-siirtymä sille, jos vastaanotin kulkisi suoraan lähetintä kohti, aliohjelmalla *dtow*(double *x,* double *y,* double *z*)*.*

Funktio *dtow(x,y,z)* toimii kuten kappaleessa 6.2 on selitetty. Muutoksena määrittelemättömälle vastaanottimelle suunniteltuun suuntima-algoritmiin on se, että superheterodyne-versiossa muuttuja *forig* sijoitetaan parametrin *x* paikalle *fgen*-muuttujan sijaan. Muuttujan *dft*-arvo lasketaan tässä ohjelmaversiossa *forig*-muuttujaa hyödyntäen (*dft=ft-forig*). *dft* sisältää muuttujan *ft* doppler-siirtymän arvon.

Muuttuja *fgen* toimii tässä ohjelman superheterodyne-versiossa superheterodyne-vastaanottimesta saatavana lähettimen välitaajuutena silloin, kun sekä lähetin että vastaanotin ovat paikoillaan eli ilman doppler-vääristymää.

Muuttujan *fm* tehtävänä on ottaa vastaan superheterodyne-vastaanottimesta saatava välitaajuus silloin, kun vastaanotin on liikkeessä. Tällöin

välitaajuudessa on mukana doppler-siirtymä, jonka arvo sijaitsee muuttujassa *dfm* (*dfm=fm-fgen*).

8.2 Algoritmin superheterodyne-version testaus

Suuntima-algoritmiohjelman superheterodyne-version testaus on suoritettu sekä ohjelmallisesti että funktiogeneraattorilla. Ohjelmallisessa testauksessa ei ole mukana ulkopuolista taajuuslähdettä, vaan muuttujille *fgen* ja *dfm* (*fm=fgen+dfm*) asetetaan arvot ohjelmassa itsessään. Funktiogeneraattoritestauksessa annetaan taajuusarvot funktiogeneraattorilla muuttujille *fgen* ja *fm* Arduino Due -kortin pinnin 3 kautta, jota algoritmin taajuuslaskuri käyttää. Molemmissa testausmuodoissa annetaan ohjelmassa arvot muuttujille *forig* ja *v*. Mittauksien tulokset ovat 3 desimaalin tarkkuudella.

8.2.1 Testaus ohjelmallisesti

Ohjelmallisessa paikannustestissä käydään lähettimen suunnan positiiviset (*facos*) ja negatiiviset (**facos*) asteet läpi noin +90°:n kulmasta –90°:n kulmaan noin viiden asteen välein. Nopeudelle *v* annetaan ihmisen kävelynopeutta vastaava arvo (1,338 m/s = 5 km/h), lähettimelle *forig* annetaan taajuusarvo 2,4 GHz, joka sijaitsee ISM-taajuusalueella, ja superheterodyne-vastaanottimen välitaajuudeksi *fgen* valitaan 100 Hz. (Taulukko 5 ja 6.)

Taulukoista huomataan, että 2,4 GHz:n taajuudella doppler-vääristymä täsmälleen lähettimen suuntaan liikkuessa nopeudella 5 km/h on 11,118 Hz. Välitaajuuden *fgen* ollessa 100 Hz tarkoittaa se noin 11 %:n lisäystä siihen, joten kun taajuus *fm* saa arvon noin 111 Hz, kuljetaan suoraan kohti lähetintä. (Taulukko 5 ja 6.)

TAULUKKO 5. Posiivisten kulmien paikannustestin mittaustulokset (superheterodyne)

Lähettimen suunnan paikannustesti ohjelmallisesti (superheterodyne) [positiiviset asteet]				
- positiivisten asteiden asetus: *dfm*:n arvoa inkrementoidaan 5 mHz joka laskentakierroksella - kulman *facos* tulokset on taulukoitu n. 5 asteen hypyin				
forig: 2,4 GHz ***fgen*****:** 100 Hz ***ft*****:** 2400000011,118 Hz ***dft*****:** 11,118 Hz ***v*****:** 1,388 m/s = 5 km/h				
	***fm* [Hz]**	***dfm* [Hz]**	***vdf***	***facos* [°]**
Tulos 1	100,001	0,001	0,000	89,994
Tulos 2	100,775	0,776	0,069	85,998
Tulos 3	101,740	1,740	0,156	80,991
Tulos 4	102,690	2,690	0,242	75,994
Tulos 5	103,620	3,620	0,325	70,994
Tulos 6	104,525	4,525	0,407	65,980
Tulos 7	105,390	5,390	0,484	60,997
Tulos 8	106,220	6,220	0,559	55,978
Tulos 9	107,000	7,000	0,629	50,975
Tulos 10	107,725	7,725	0,694	45,984
Tulos 11	108,395	8,395	0,755	40,964
Tulos 12	108,996	8,996	0,809	35,994
Tulos 13	109,531	9,531	0,857	30,997
Tulos 14	109,996	9,996	0,899	25,971
Tulos 15	110,381	10,381	0,933	20,991
Tulos 16	110,691	10,691	0,961	15,946
Tulos 17	110,916	10,916	0,981	10,962
Tulos 18	111,061	11,061	0,994	5,849
Tulos 19	111,116	11,116	0,999	1,306

Taulukosta 5 huomataan, että *fm* ja *fgen* välitaajuuksien ero on selvästi nähtävissä, joten doppler-vääristymän havaitseminen on helppoa. Doppler-vääristymän suurin arvo (*dft*) on 11,118 Hz:n taajuudella 2,4 GHz (*forig*) ja nopeudella 5 km/h (*v*). Liikkeessä vastaanotetusta 111,116 Hz:n välitaajuudesta *fm* eli 11,116 Hz:n doppler-vääristymästä *dfm* (*dfm=fm-fgen*) saadaan suora kulkusuunta lähettimeen, kulman *facos* saadessa arvon 1,306° (*facos* = arccos (*vdf*) = arccos (*dfm*/*dft*) = arcos (11,116/11,118)) (tulos 19).

TAULUKKO 6. Negatiivisten kulmien paikannustestin mittaustulokset (superheterodyne)

Lähettimen suunnan paikannustesti ohjelmallisesti (superheterodyne) [negatiiviset asteet]					
- negatiivisten asteiden asetus: *dfm*:n arvoa inkrementoidaan 5 mHz ja dekrementoidaan 1 µHz joka laskentakierroksella - kulman **facos* tulokset on taulukoitu n. 5 asteen hypyin					
***forig*:** 2,4 GHz ***fgen*:** 100 Hz ***ft*:** 2400000011,118 Hz ***dft*:** 11,118 Hz ***v*:** 1,388 m/s = 5 km/h					
	***fm* [Hz]**	***dfm* [Hz]**	***vdf***	***facos* [°]**	***facos* [°]**
Tulos 1	107,864	7,864	0,707	44,984	-89,968
Tulos 2	108,134	8,134	0,731	42,980	-85,961
Tulos 3	108,459	8,459	0,760	40,464	-80,929
Tulos 4	108,764	8,764	0,788	37,979	-75,959
Tulos 5	109,054	9,054	0,814	35,481	-70,962
Tulos 6	109,329	9,329	0,839	32,961	-65,923
Tulos 7	109,584	9,584	0,861	30,462	-60,924
Tulos 8	109,819	9,819	0,883	27,982	-55,964
Tulos 9	110,038	10,038	0,902	25,461	-50,922
Tulos 10	110,238	10,238	0,920	22,947	-45,895
Tulos 11	110,418	10,418	0,937	20,438	-40,877
Tulos 12	110,578	10,578	0,951	17,930	-35,860
Tulos 13	110,718	10,718	0,964	15,415	-30,831
Tulos 14	110,838	10,838	0,974	12,886	-25,773
Tulos 15	110,933	10,933	0,983	10,468	-20,937
Tulos 16	111,013	11,013	0,990	7,883	-15,767
Tulos 17	111,068	11,068	0,995	5,441	-10,882
Tulos 18	111,103	11,103	0,998	2,987	-5,975
Tulos 19	111,118	11,118	0,999	0,261	-0,522

Taulukosta 6 nähdään, että algoritmin suunnan korjaustoiminto aktivoituu, kun alkuperäinen väärin laskettu kulma *facos* alittaa 45°. Tällöin korjattu kulma **facos* saa arvon ~–90° (**facos=facos**-2) (tulos 1). Negatiivisten eli korjattujen kulmien tapauksessa otetaan huomioon vastaanottimen käyttäjän suunnan muutokset. Tämän vuoksi *facos* ja **facos* eroavat toisistaan.

8.2.2 Testaus funktiogeneraattorilla

Funktiogeneraattorilla suoritetussa paikannustestissä käydään lähettimen suunnan positiiviset asteet (*facos*) läpi mahdollisimman tarkasti noin 90°:n kulmasta noin kulmaan 0°. Nopeudelle *v* annetaan ihmisen kävelynopeutta vastaava arvo (1,338 m/s = 5 km/h), lähettimelle *forig* annetaan taajuusarvo 2,4 GHz, joka sijaitsee ISM-taajuusalueella, ja superheterodyne-vastaanottimen välitaajuudeksi *fgen* asetetaan funktiogeneraattorilla noin 100 Hz:n taajuus. Funktiogeneraattorista syötetään 2 Vpp (peak-to-peak) amplitudista kanttiaaltoista signaalia. (Taulukko 7.)

Kulmat (*facos*) käydään läpi siten, että kasvatetaan funktiogeneraattorin taajuutta noin 100 Hz:n taajuudesta 100 mHz:n hypyin aina noin 111 Hz:iin asti, jolloin vastaanotin osoittaa suoraan simuloitua lähetintä kohti. Mittaustulosten taajuudet *fm* ja *fgen* on keskiarvoistettu 50 välituloksella. (Taulukko 7.)

Tulokset heittelevät noin 1–3 astetta suuntaansa (esim. ~80° heittelee välillä ~77°–~83°) taulukon 7 tulosten alkuvaiheesta noin keskivaiheeseen asti. Suurin osa kulmien heittelystä pysyy kuitenkin ±1 asteen sisällä noin 45°:n kulmaan (*facos*) asti.

Tulosten loppupuolella heittely lisääntyy ja on noin 3–6 astetta suuntaansa. Tällöinkin suurin osa kulmien variaatiosta pysyy ±3 asteen sisällä noin 25°:n kulmaan asti. Kun ollaan lähellä kulmaa 0°, on asteiden variaatio voimakkainta. (Taulukko 7.)

TAULUKKO 7. Lähettimen suunnan paikannustestin mittaustulokset (superheterodyne)

Lähettimen suunnan paikannustesti funktiogeneraattorilla (superheterodyne) [positiiviset asteet]				
- keskiarvoistettu 50 välituloksella - kanttiaaltoinen signaali				
forig*:** 2,4 GHz ***fgen: 99,870 Hz ***ft:*** 2400000011,118 Hz ***dft:*** 11,118 Hz ***v:*** 1,388 m/s = 5 km/h				
	***fm* [Hz]**	***dfm* [Hz]**	***vdf***	***facos* [°]**
Tulos 1	99,974	0,103	0,009	89,464
Tulos 2	100,083	0,212	0,019	88,902
Tulos 3	100,345	0,475	0,042	87,550
Tulos 4	100,459	0,588	0,052	86,963
Tulos 5	100,646	0,776	0,069	85,996
Tulos 6	100,923	1,052	0,094	84,566
Tulos 7	101,097	1,227	0,110	83,662
Tulos 8	101,249	1,379	0,124	82,874
Tulos 9	101,422	1,552	0,139	81,975
Tulos 10	101,667	1,797	0,161	80,698
Tulos 11	101,830	1,959	0,176	79,848
Tulos 12	102,080	2,210	0,198	78,533
Tulos 13	102,226	2,356	0,211	77,764
Tulos 14	102,403	2,532	0,227	76,832
Tulos 15	102,651	2,780	0,250	75,516
Tulos 16	102,927	3,056	0,274	74,043
Tulos 17	102,950	3,080	0,277	73,916
Tulos 18	103,276	3,406	0,306	72,161
Tulos 19	103,334	3,464	0,311	71,844
Tulos 20	103,612	3,741	0,336	70,334
Tulos 21	103,802	3,932	0,353	69,289
Tulos 22	104,031	4,160	0,374	68,025
Tulos 23	104,036	4,165	0,375	67,997
Tulos 24	104,311	4,441	0,376	66,456
Tulos 25	104,512	4,641	0,377	65,325
Tulos 26	104,626	4,756	0,378	64,673
Tulos 27	104,915	5,045	0,379	63,013
Tulos 28	104,993	5,123	0,380	62,561
Tulos 29	105,132	5,262	0,381	61,752
Tulos 30	105,266	5,395	0,485	60,969

taulukko 7 jatkuu

taulukko 7 jatkuu

Tulos 31	105,543	5,672	0,510	59,322
Tulos 32	105,606	5,736	0,515	58,943
Tulos 33	105,797	5,927	0,533	57,784
Tulos 34	105,985	6,115	0,549	56,634
Tulos 35	106,091	6,220	0,559	55,979
Tulos 36	106,382	6,511	0,585	54,149
Tulos 37	106,449	6,579	0,591	53,721
Tulos 38	106,593	6,723	0,604	52,794
Tulos 39	106,831	6,960	0,626	51,241
Tulos 40	106,885	7,014	0,630	50,884
Tulos 41	107,093	7,222	0,649	49,489
Tulos 42	107,192	7,322	0,658	48,810
Tulos 43	107,315	7,444	0,669	47,966
Tulos 44	107,496	7,626	0,685	46,693
Tulos 45	107,694	7,823	0,703	45,279
Tulos 46	107,822	7,951	0,715	44,342
Tulos 47	107,984	8,113	0,729	43,134
Tulos 48	108,125	8,255	0,742	42,059
Tulos 49	108,142	8,272	0,743	41,930
Tulos 50	108,265	8,395	0,755	40,969
Tulos 51	108,498	8,627	0,775	39,107
Tulos 52	108,536	8,666	0,779	38,792
Tulos 53	108,745	8,874	0,798	37,043
Tulos 54	108,753	8,883	0,798	36,969
Tulos 55	108,932	9,062	0,815	35,407
Tulos 56	109,051	9,181	0,825	34,334
Tulos 57	109,151	9,280	0,834	33,418
Tulos 58	109,282	9,411	0,846	32,170
Tulos 59	109,398	9,527	0,856	31,031
Tulos 60	109,495	9,625	0,865	30,042
Tulos 61	109,573	9,703	0,872	29,228
Tulos 62	109,670	9,799	0,881	28,191
Tulos 63	109,705	9,835	0,884	27,803
Tulos 64	109,778	9,908	0,891	26,985
Tulos 65	109,899	10,028	0,901	25,583
Tulos 66	109,976	10,106	0,908	24,639
Tulos 67	110,057	10,186	0,916	23,630
Tulos 68	110,107	10,237	0,920	22,970
Tulos 69	110,243	10,372	0,932	21,108
Tulos 70	110,288	10,417	0,936	20,455
Tulos 71	110,325	10,455	0,940	19,893
Tulos 72	110,406	10,536	0,947	18,630
Tulos 73	110,487	10,617	0,954	17,280
Tulos 74	110,550	10,679	0,960	16,158

taulukko 7 jatkuu

taulukko 7 jatkuu

Tulos 75	110,596	10,726	0,964	15,271
Tulos 76	110,655	10,785	0,969	14,076
Tulos 77	110,699	10,828	0,973	13,113
Tulos 78	110,779	10,908	0,981	11,156
Tulos 79	110,860	10,990	0,988	8,715
Tulos 80	110,958	11,088	0,997	4,269
Tulos 81	110,986	11,115	0,999	1,351

Funktiogeneraattorilla tehdyt mittaukset osoittavat, että suuntima-algorimin taajuuslaskuri on riittävän tarkka (joskin variaatiota on) lähettimen suunnan löytämiseen välitaajuudesta 100 Hz (*fgen*). Välitaajuuden *fgen* taajuus 99,870 Hz eli taajuus, kun vastaanotin (ja lähetin) on paikallaan, on mitattu taajuuslaskurilla siten, että funktiogeneraattoriin on asettu täsmälleen 100 Hz:n taajuus. Kun taajuus *fgen* on asettu, mitataan ainoastaan välitaajuutta *fm* (vastaanotin liikkeessä) kasvattamalla funktiogeneraattorin taajuusarvoa 100 mHz:n hypyin aina noin 111 Hz:iin asti. (Taulukko 7.)

Viimeinen eli 1,351°:n mittaustulos (tulos 81) saatiin funktiogeneraattorista mitatulla taajuusarvolla 110,986 Hz (*fm*). Tällöin doppler-siirtymän suuruus on 11,115 Hz (*dfm* = *fm* – *fgen* = 110,986 Hz – 99,870 Hz = ~11,115 Hz). (Taulukko 7.)

Mittaukset vastaavat tilannetta, jossa 2,4 GHz:n taajuudella lähettävää lähetintä paikannetaan kävelyvauhdista. Jos nopeutta kasvatettaisiin, vähenisi myös variaatio kulmien *facos* tuloksissa. Heittely vähenisi myös, jos sallittaisiin vain sellaiset kulmien *facos* mittaustulokset, joiden arvot ovat lähimpänä toisiaan, ja jätettäisiin arvot, jotka menevät tietyn verran yli määritellystä otannasta. Näissä mittaustuloksissa on käytetty 50 välituloksen keskiarvoistusta välitaajuuksien mittaamiseen. Kun välituloksien lukumäärää nostetaan, kasvaa myös taajuuslaskurin mittaustarkkuus. (Taulukko 7.)

8.3 Yhteenvetoa algoritmin superheterodyne-versiosta

Superheterodyne-vastaanottimella saadaan doppler-vääristymä siirrettyä lähettimestä vastaanotetusta signaalista suoraan välitaajuudelle, jolloin sen mittaaminen ja jatkokäsittely on huomattavasti helpompaa.

Ainoana heikkoutena superheterodyne-vastaanottimessa on se, että sen muodostama välitaajuus on spesifinen vain tietylle lähettimen taajuudelle (esim. 2,4 GHz). Taajuuden spesifisyys on kuitenkin pieni kompromissi siihen nähden, että lähettimen paikantaminen vastaanotetusta taajuudesta vaatii paljon vähemmän tarkkuutta vastaanottimelta ja algoritmin taajuuslaskurilta.

Sellaiselta vastaanottimelta, jonka täytyisi kyetä huomaamaan doppler-vääristymä lähetetystä signaalista esim. 5 km/h tuntinopeudella liikkuessa ilman, että se siirrettäisiin matalammalle välitaajuudelle, vaadittaisiin paljon suurempaa tarkkuutta kuin doppler-siirtymän mittaamiseen välitaajuus-menetelmällä.

9 JATKOTOIMENPITEET

Jotta valmis Doppler-kompassi voitaisiin rakentaa, tarvittaisiin tarkka vastaanotin, jolla vastaanotetusta radiotaajuudesta voidaan erottaa doppler-ilmiön aiheuttama siirtymä. Nykyisillä taajuuslaskureilla ei ole kustannustehokasta laskea esimerkiksi UHF-taajuusalueella värähtelevän signaalin jokaista kellojaksoa ja yrittää havaita siitä muutamien hertsien tai muutamien kymmenien hertsien suuruisia muutoksia vastaanotetussa taajuudessa. Doppler-vääristymän havaitsemiseen sopisi mm. superheterodyne-vastaanotin. (3, s. 7.)

Superheterodyne-vastaanottimen eli välitaajuusvastaanottimen avulla kantoaallon (esim. UHF-taajuus) doppler-siirtymä saadaan pudotettua matalammalle taajuudelle, jolloin sen mittaaminen helpottuu. Doppler-siirtymä näkyy saman kokoisena välitaajuudessa. Välitaajuus voidaan laskea siten, että vähennetään vastaanotetusta taajuudesta paikallisoskillaattorin taajuus ja lisätään siihen doppler-ilmiön aiheuttama taajuuden muutos. (3, s. 12.)

Työn taajuuslaskuriosio jäi odotettua epätarkemmaksi havaitsemaan funktiogeneraattorista vastaanotettua signaalia, jotta sillä voitaisiin saada täysin tarkkoja ja heittelemättömiä tuloksia ”lähettimen” suunnasta. Tämä epätarkkuus ilmeni matalalta 100 Hz:n taajuudelta ja kasvoi aina 2 kHz:iin asti, jonka jälkeen laskurin taajuuden mittauskyky katosi täysin. Ongelma laskurin odotettua heikompaan tarkkuutteen ja kykenemättömyyteen laskea korkeampia kuin 2 kHz:n taajuuksia saattaa johtua siitä, ettei käytettyä laskurifunktiota ole aiemmin testattu Arduino Due -kortilla. Jos taajuuslaskurista saataisiin tarkempi, olisi suuntima-algoritmin testaaminen ulkoisella taajuuslähteellä tai langattomasti sujuvampaa.

Välitaajuusvastaanottimesta tulevasta matalammasta, esimerkiksi 100 Hz:n taajuudesta algoritmin taajuuslaskuri kykenee mittamaan vaikkapa kävelyvauhdista UHF-taajuusalueen kantoaallon doppler-siirtymän, sillä doppler-vääristymä säilyy välitaajuudessa yhtä suurena kuin kantoaallossa.

Doppler-suuntima-algoritmin testaus langattomasti vastaanottimen avulla olisi seuraava askel algoritmin toiminnan testauksessa. Langaton testaus antaisi tarkemman kokonaiskuvan Doppler-kompassin toiminnasta ja siitä, kuinka hyvin sitä voitaisiin hyödyntää radiolähettimen paikantamiseen kävellessä, juostessa tai vaikkapa autolla ajettaessa. Jotta algoritmista saataisiin tarkkoja lukemia Doppler-kompassin todellisessa käyttötilanteessa, tarvitsisi käyttäjän nopeus olla tiedossa. Nopeus voitaisiin mitata vaikkapa jonkin nopeusmittarin kautta tai esimerkiksi GPS-paikannusta hyödyntämällä. Toinen vaihtoehto voisi olla käyttöliittymä, josta valitaan nopeus ohjelmallisesti. Tämän opinnäytetyön puitteisiin ei langaton testaus kuulunut, ja se vaatisi esimerkiksi aiemmin mainitun superheterodyne-vastaanottimen kehittämisen, joten se olisikin sitten toisen opinnäytetyön laajuinen prosessi.

10 LOPPUSANAT

Tässä työssä suunniteltiin ja kirjoitettiin C-kielellä doppler-suuntima-algoritmi sekä kehitettiin suunnan osoitinnäyttö Doppler-kompasille. Suuntima-algoritmista on oma versio superheterodyne-vastaanottimelle sekä teoreettinen versio määrittelemättömälle vastaanottimelle, joka pystyisi mittaamaan doppler-vääristymän suoraan lähettimen taajuudesta ilman sen muokkaamista välitaajuudelle.

Laboratoriotestauksen kannalta ongelmaa aiheutti suuntima-algoritmiohjelman negatiivisten asteiden testaus, joka osoittautui sen verran vaikeaksi toteuttaa, että tulokset kyseisestä suunnankorjaustoiminnosta taulukoitiin vain ohjelmallisen testauksen kautta.

Algoritmin kierroslaskurikoodin oikeellisuus fysikaalisessa mielessä jäi kyseenalaiseksi. Luultavasti kierroslaskurin ohjelmakoodiin ei todellisessa lähettimen suunnan paikannustilanteessa tarvitse mennä, sillä periaatteessa liikkeessä mitattu lähettimen taajuus ei voi ylittää teoreettisesti laskettua taajuusarvoa, jossa vastaanotin kulkisi suoraan lähetintä kohti, edellyttäen, että vastaanottimen kulkunopeus todellisuudessa on sama tai pienempi kuin teoreettisesti lasketun taajuuden yhtälössä. Jotta algoritmista saadaan totuudenmukainen tulos lähettimen suunnalle, täytyy vastaanottimen todellisen liikenopeuden olla yhtä suuri kuin teoreettisesti lasketun taajuuden yhtälössä.

Superheterodyne-vastaanottimelle suunniteltu versio suuntima-algoritmista toimii taajuuslaskurilla funktiogeneraattorista mitattaessa melko tarkasti ja sillä voitiin paikantaa simuloitu lähetin muutaman asteen tarkkuudella, joskin variaatiota kulmissa esiintyi.

Tämän opinnäytetyön tekemiseen sisältyi runsaasti pohdintaa ja se oli mielenkiintoinen sekä opettavainen kokemus. Suuntima-algoritmin suunnittelun ja ohjelmoinnin lisäksi työn aihepiiriin kuului piirikaavio- ja piirilevysuunnittelua sekä komponenttien juottamista ja mittalaitteiden käyttöä, joten sain siitä mielestäni paljon irti.

LÄHTEET

1. Karttunen, Hannu, Ursa ja Tuorlan observatorio. Doppler-ilmiö. Saatavissa: http://www.astro.utu.fi/zubi/spectra/doppler.htm. Hakupäivä 26.2.2014.

2. Doppler_effect. 2014. Wikipedia. Vapaa tietosanakirja. Saatavissa: http://en.wikipedia.org/wiki/Doppler_effect. Hakupäivä 26.2.2014.

3. Taskila, Pekka 2011. Doppler superheterodyne -vastaanotin. Oulu: Oulun seudun ammattikorkeakoulu, tietotekniikan koulutusohjelma. Opinnäytetyö. Saatavissa: http://urn.fi/URN:NBN:fi:amk-2011121117934. Hakupäivä 27.2.2014.

4. Direction_finding. Wikipedia. Vapaa tietosanakirja. 2014. Saatavissa: http://en.wikipedia.org/wiki/Direction_finding. Hakupäivä 1.3.2014.

5. Arduino Due. 2014. Arduino. Saatavissa: http://arduino.cc/en/Main/arduinoBoardDue. Hakupäivä 4.4.2014.

6. van Dalen, Albert 2011. Frequency / period counter for the Arduino. Saatavissa: http://www.avdweb.nl/arduino/hardware-interfacing/frequency-period-counter.html. Hakupäivä 26.4.2014.

7. 74HC595. DatasheetCatalog.com. Saatavissa: http://pdf.datasheetcatalog.com/datasheet/stmicroelectronics/1989.pdf. Hakupäivä 9.5.2014.

8. Pulssisuhde. 2013. Wikipedia. Vapaa tietosanakirja. Saatavissa: http://fi.wikipedia.org/wiki/Pulssisuhde. Hakupäivä 12.5.2014.

9. SAM3X8E. Atmel. Saatavissa: http://www.atmel.com/devices/sam3x8e.aspx. Hakupäivä 12.5.2014

10. Taajuus. 2013. Wikipedia. Vapaa tietosanakirja. Saatavissa: http://fi.wikipedia.org/wiki/Taajuus. Hakupäivä 21.5.2014.

LIITTEET

Liite 1 Lähtötietomuistio

Liite 2 C-kielinen doppler-suuntima-algoritmi (ulkoisella taajuuslähteellä testaus)

Liite 3 C-kielinen doppler-suuntima-algoritmi (ohjelmallinen testaus PWM-signaalilla ja ilman negatiivisia asteita)

Liite 4 C-kielinen doppler-suuntima-algoritmi (ohjelmallinen testaus)

Liite 5 C-kielinen doppler-suuntima- algoritmi [superheterodyne] (ulkoisella taajuuslähteellä testaus)

Liite 6 C-kielinen doppler-suuntima-algoritmi [superheterodyne] (ohjelmallinen testaus ilman negatiivisia asteita)

Liite 7 C-kielinen doppler-suuntima-algoritmi [superheterodyne] (ohjelmallinen testaus)

LÄHTÖTIETOMUISTIO

Tekijä[1] Matti Rekis, [redacted]

Tilaaja[2] Oulun seudun ammattikorkeakoulu/tekniikan yksikkö

Tilaajan yhdyshenkilö ja yhteystiedot[3] Veijo Korhonen / Henry Hinkula

Työn nimi[4] Doppler-kompassi

Työn kuvaus[5] Rakentaa demolaitteisto, missä lähettimen suunta lasketaan Doppler-taajuuden perusteella ja suunta ilmaistaan esim. LED:illä.

Työn tavoitteet[6] Laitteen suunnittelu ja toteutus prototyyppiasteelle.Testaus laboratoriossa.

Tavoiteaikataulu[7] 31.05.2014

Päiväys ja allekirjoitukset[8] 20.2.2014 Matti Rekis

1. Tekijän nimi, puhelinnumero ja sähköpostiosoite.
2. Työn teettävän yrityksen virallinen nimi.
3. Sen henkilön nimi ja yhteystiedot, joka yrityksessä valvoo työnsuoritusta.
4. Työn nimi voi olla tässä vaiheessa työnimi, jota myöhemmin tarkennetaan.
5. Työ kuvataan lyhyesti. Siinä esitetään muun muassa työn tausta, lähtötilanne ja työssä ratkaistavat ongelmat.
6. Esitetään lyhyesti ja selvästi työn tavoitteet.
7. Esitetään projektin tavoiteaikataulu. Silloin, kun työllä on välitavoitteita, myös ne merkitään aikatauluun. Tavoiteaikataulun ja oppilaitoksen yleisaikataulun perusteella tekijä laatii oman aikataulunsa.
8. Lähtötietomuistio päivätään ja sen allekirjoittavat tekijä ja tilaajan yhdyshenkilö.

```
/*************************************************************************************************************************************
*************************************************************************************
Project : Doppler-bearing algorithm v.1.0 (external frequency source)
Hardware: Arduino Due [AT91SAM3X8E]
Software: Arduino software
Date    : 14.05.2014
Author  : Matti Rekis
Changes : Added code for LED screen and for averaging frequency measurements. Removed
the "moving away"-value calculation of the doppler-shift (dawa()).
Comments: Frontside leds in half circle and one led at back pointing if moving away from Tx.
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------
Using 64 bit double attributes for calculations/measurements.
Attributes  : fgen = original Tx frequency for reference [when still; no doppler shift]
              v = speed of the receiver [m/s] [Tx stays still, Rx moves]
              fm = measured (or simulated) Tx frequency when moving [doppler shift included]
              ft = calculated theoretical frequency value where Rx would be moving straight to Tx [or
straign away from Tx <- it's not necessary to calculate the "moving away" value]
              dfm = value of the doppler shift of fm [dfm=fm-fgen;fm=fgen+dfm]
              dft = value of the theoretically calculated doppler shift of ft [dft=ft-fgen]
              vdf = doppler shift in fm [dfm] / calculated theoretical doppler shift in ft [dft];
[vdf=dfm/dft]
              facos = the angle between Tx and Rx; calculated by arccos-function [facos=acos(vdf)]
              dfm2 = used to observe if the angle of Rx and Tx changes correctly when moving to-
wards to Tx [fm frequency should not decrease if moving closer to Tx]
*************************************************************************************************************************************
*************************************************************************************/
#include <Streaming.h>
#include <FreqPeriodCounter.h>

double dtow(double x, double y, double z); //functions
//double dawa(double x, double y, double z);
void ledscr(double x);
void ledsoff();

const byte counterPin = 3; // pin 3 = fgen (pin 3 is used for the measurement of the frequencies
with FreqPeriodCounter function)
const byte counterInterrupt = 1;

const byte dataPin=24, clockPin=25, latchPin=26, oePin1=27, oePin2=28; //D24=SI[DS],
D25=SCK[SH_CP], D26=RCK[ST_CP], D27=!G[!OE] for IC1, D28=!G[!OE] for IC2
const byte led17Pin=29, led18Pin=30; //D29=LED17[0 degree], D30=LED18[180 de-
grees/Txback]

FreqPeriodCounter counter(counterPin, micros, 0); //calculates the time that takes to HIGH-
LOW-period (or. vice versa) to determine T  -> f=1/T

void setup(void)
{
pinMode(dataPin, OUTPUT); //set pins to outputs to control the shift registers
pinMode(clockPin, OUTPUT);
pinMode(latchPin, OUTPUT);
pinMode(oePin1, OUTPUT); //IC1 output enable pin
pinMode(oePin2, OUTPUT); //IC2 output enable pin
pinMode(led17Pin, OUTPUT); //set pin to output for 0 degrees
pinMode(led18Pin, OUTPUT); //set pin to output for 180 degrees/Txback
Serial.begin(9600);
pinMode(9,OUTPUT); //pin 9 have to be connected to pin 3 to test with arduinos PWM-signal
analogWrite(9, 127); //pin 9 = 50% duty cycle (arduino PWM-signal in pin 9)
delay(500); //wait for stable PWM-signal
```

```
attachInterrupt(counterInterrupt, counterISR, CHANGE); //using interrupt for the frequency measurements
ledsoff(); //turns off all leds
}

void loop(void)
{
const double pi=3.14159265359, c=299792458;
double v=2.7778, fgen=0, fm=0, ft=0, dfm=0, dft=0, facos=0, vdf=0, rcircled=0, dfm2=0;
//attributes for calculating and measuring doppler shift
int rcircle=0, rcc=0;
bool reset=true, rcflag=true;
byte i=0;

if(counter.ready()) //tests that the counter function is ready
Serial << "counter_function: " << counter.hertz(1000) << "\n";

while(1)
{
fm=0; //formats the frequency result before new measurement/calculation

if(reset==true) //reset -> Tx [fgen] frequency measurement and averanging when still (without doppler shift) (used with external frequency source)
{
while(i<20)//averages the frequency measurement [accuracy 20 frequency results]
{
if(counter.ready())
{
fgen=fgen+(double)counter.hertz(1000)/1000; //frequency accuracy in mHz .000 Hz
i++;
Serial << "Freq_value [fgen] " << i << ": " << fgen << "\n"; //shows the frequency sums in serial monitor (while loop won't work without this command for some strange reason...)
}
else if(counter.elapsedTime > 50000) //detects if there is no frequency signal for 50ms
{
Serial << "No signal.\n";
ledsoff(); //turns off all leds
i=(byte)0;
fgen=0;
}
}
fgen=fgen/(double)i;
i=(byte)0;
reset=false;
//fgen=fgen*1000000; //converts to higher frequency [ex. MHz] to see the doppler shift with Due PWM-signal or low frequency with function generator (BETA)
ft=dtow(fgen,c,v); //calculates the theoretical value of doppler frequency where Rx would be moving straight towards to Tx
dft=ft-fgen;
Serial << "-------------------------------------------------------------------------------------------------------------------\n";
}

//fm=fgen+dfm; //this code is used when dfm value is set programmatically

while(i<10) //frequency measurements and averaging when moving [accuracy 10 frequency results] (this code section is used with external frequency source)
{
if(counter.ready())
{
```

```
fm=fm+(double)counter.hertz(1000)/1000; //frequency accuracy in mHz .000 Hz
i++;
Serial << "Freq_value [fm] " << i << ": " << fm << "\n"; //shows the frequency sums in Serial
Monitor (while loop won't work without this command for some strange reason...)
}
else if(counter.elapsedTime > 50000) //detects if there is no frequency signal for 50ms
{
Serial << "No signal.\n";
ledsoff(); //turns off all leds
i=(byte)0;
fm=0;
}
}
fm=fm/(double)i;
//fm=fm*1000000; //convert to higher frequency [ex. MHz] to see the doppler shift with Due pwm
signal or low frequency with function generator (BETA)
dfm=fm-fgen;
i=(byte)0;
Serial << "----------------------------------------------------------------------------------------------------------
----------\n";

/*if(fm>fgen) //this code section is not needed, because the Tx-away function (dawa()) will not
be used
ft=dtow(fgen,c,v);
else if(fm<fgen)
ft=dawa(fgen,c,v);
dft=ft-fgen;*/

vdf=dfm/dft; //measured (or simulated) doppler shift in fm [dfm=fm-fgen] / calculated theoretical
doppler shift in ft [dft=ft-fgen]
Serial << "dfm: " << dfm << " Hz, v: " << v << " m/s, fgen: " << fgen << " Hz, fm: " << fm << " Hz,
ft: " << ft << " Hz, vdf: " << vdf << "\n";

if(vdf>1) //converts vdf-value when vdf>1 (vdf is always a posit.value), [ex. if vdf=1.01 ->
(new)vdf=0.01||0.99, this code section changes the flow of full cycles]
{
Serial << "vdf_old: " << vdf << "\n";
rcircle=(int)vdf; //cycle count from double to int (how many full frequency cycles have happened
(ex. dfm=100 & dft=50 -> two cycles)
rcircled=(double)rcircle; //cycle count from int to double (converts to full cycles only, no decimal
numbers)
if(rcc!=rcircle&&rcircle>=2) //alternates the vdf-values calculation parametres of a vdf>1-state on
every second change when a full cycle happens (first cycle is not included)
rcflag=!rcflag;
if(rcflag==true)
vdf=1-(vdf-rcircled);
else
vdf=vdf-rcircled;
Serial << "rcircle: " << rcircle << ", rcircled: " << rcircled << ", vdf_new: " << vdf << "\n";
}

rcc=rcircle; //copies the value of rcircle to rcc, so that a full cycle change can be seen when
rcircles value changes

if(fm>fgen) //moving towards to Tx
{
Serial << "Heading to Tx.\n";
//dfm=dfm-0.000001; //angle repair test [-angles] (programmatically simulated dfm)
facos=acos(vdf)*180/pi;
Serial << "[Original angle] vdf: " << vdf << ", Acos: " << facos << " [+Tx_0]\n"; //shows the origi-
```

```
nal angle in the serial monitor
if(dfm<dfm2&&vdf>=cos(45*pi/180)) //if dfm<dfm2 the Rx is going to the wrong way and needs to repair its direction, only angles that are <=45 degrees is repaired
{
facos=-facos*2; //ex. +44 degrees -> after direction repair -88 degrees
Serial << "[Repaired angle] vdf: " << vdf << ", Acos: " << facos << " [+Tx_1]\n"; //shows the repaired angle in the serial monitor
}
ledscr(facos); //function for LED screen
//dfm=dfm+0.1; //increases the value of dfm, for testing the algorithm programmatically
//if(dfm>=(dft*4)) //for testing the cycle calculation code when vdf>1
//dfm=-0.001;
}
else if(fm<fgen) //moving away from Tx. 1 led at the back of the led screen shows when moving away, so no need to calculate the angle of the Tx
{
Serial << "Heading away Tx.\n";
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, HIGH); //disables IC2 outputs
digitalWrite(led18Pin, HIGH); //set 180 degrees/backTx led on
digitalWrite(led17Pin, LOW); //set 0 degree led off
//facos=acos(vdf)*180/pi;
//Serial << "vdf: " << vdf << ", Acos: " << facos << " [-Tx_0]\n";
//dfm=dfm-1; //decreases the value of dfm, for testing the algorithm programmatically
//if(dfm<=(dft*-2))
//dfm=0.001;
}
else
{
Serial << "No doppler-frequency.\n";
ledsoff(); //turns off all leds
}
//delay(100);
Serial << "-----------------------------------------------------------------------------------------------------------------------\n";
dfm2=dfm; //copies the value of dfm to dfm2 to see if the last dfm(dfm2) is smaller than the new dfm of next calculation
}
}

double dtow(double x, double y, double z) //function for calculating theoretical value of frequency where Rx would be moving straight towards to Tx at certain speed
{
double vft=0;
vft=x*((y+z)/y);
return(vft);
}

/*double dawa(double x, double y, double z) //function for calculating theoretical value of frequency where Rx would be moving straight backwards from Tx at certain speed (this function is not necessary)
{
double vft=0;
vft=x*((y-z)/y);
return(vft);
}*/

void ledscr(double x) //fuction for controlling the led pointer display
{
if(x>-5.625&&x<5.625) //0' led17 on
```

```
{
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, HIGH); //disables IC2 outputs
digitalWrite(led17Pin, HIGH); //set 0' led on
digitalWrite(led18Pin, LOW); //set 180'/backTx led off
}
else
{
digitalWrite(led17Pin, LOW); //set 0' led off
digitalWrite(led18Pin, LOW); //set 180'/backTx led off
if(x>0&&x>=5.625) //IC2 enable
{
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, LOW); //enables IC2 outputs
}
else if(x<0&&x<=-5.625)//IC1 enable
{
digitalWrite(oePin2, HIGH); //disables IC2 outputs
digitalWrite(oePin1, LOW); //enables IC1 outputs
}
else
{
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, HIGH); //disables IC2 outputs
}
digitalWrite(latchPin, LOW); //latch pin low so led's don't change while sending bits
if(x<0)
{
if(x>=-90.000&&x<=-84.375) //LED1 -90' [leds 1-8 negat.']
shiftOut(dataPin, clockPin, MSBFIRST, 0x01);
else if(x>-84.375&&x<=-73.125) //LED2 -78.75'
shiftOut(dataPin, clockPin, MSBFIRST, 0x02);
else if(x>-73.125&&x<=-61.875) //LED3 -67.5'
shiftOut(dataPin, clockPin, MSBFIRST, 0x04);
else if(x>-61.875&&x<=-50.625) //LED4 -56.25'
shiftOut(dataPin, clockPin, MSBFIRST, 0x08);
else if(x>-50.652&&x<=-39.375) //LED5 -45'
shiftOut(dataPin, clockPin, MSBFIRST, 0x10);
else if(x>-39.375&&x<=-28.125) //LED6 -33.75'
shiftOut(dataPin, clockPin, MSBFIRST, 0x20);
else if(x>-28.125&&x<=-16.875) //LED7 -22.5'
shiftOut(dataPin, clockPin, MSBFIRST, 0x40);
else if(x>-16.875&&x<=-5.625) //LED8 -11.25'
shiftOut(dataPin, clockPin, MSBFIRST, 0x80);
}
else
{
if(x<16.875&&x>=5.625) //LED9 +11.25' [leds 9-16 posit.']
shiftOut(dataPin, clockPin, MSBFIRST, 0x01);
else if(x<28.125&&x>=16.875) //LED10 +22.5'
shiftOut(dataPin, clockPin, MSBFIRST, 0x02);
else if(x<39.375&&x>=28.125) //LED11 +33.75'
shiftOut(dataPin, clockPin, MSBFIRST, 0x04);
else if(x<50.652&&x>=39.375) //LED12 +45'
shiftOut(dataPin, clockPin, MSBFIRST, 0x08);
else if(x<61.875&&x>=50.625) //LED13 +56.25'
shiftOut(dataPin, clockPin, MSBFIRST, 0x10);
else if(x<73.125&&x>=61.875) //LED14 +67.5'
shiftOut(dataPin, clockPin, MSBFIRST, 0x20);
else if(x<84.375&&x>=73.125) //LED15 +78.75'
```

```
shiftOut(dataPin, clockPin, MSBFIRST, 0x40);
else if(x<=90.000&&x>=84.375) //LED16 +90'
shiftOut(dataPin, clockPin, MSBFIRST, 0x80);
}
digitalWrite(latchPin, HIGH); //latch pin high so leds will light up
}
}

void ledsoff() //function to turn all leds off
{
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, HIGH); //disables IC2 outputs
digitalWrite(led17Pin, LOW); //set 0' led off
digitalWrite(led18Pin, LOW); //set 180'/backTx led off
}

void counterISR() //frequency counter function
{
counter.poll();
}
```

```
/*****************************************************************************************************************
*******************************************************************************
Project : Doppler-bearing algorithm v.1.0 (PWM-signal)
Hardware: Arduino Due [AT91SAM3X8E]
Software: Arduino software
Date    : 14.05.2014
Author  : Matti Rekis
Changes : Added code for LED screen and for averaging frequency measurements. Removed
the "moving away"-value calculation of the doppler-shift (dawa()).
Comments: Frontside leds in half circle and one led at back pointing if moving away from Tx.
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------
Using 64 bit double attributes for calculations/measurements.
Attributes  : fgen = original Tx frequency for reference [when still; no doppler shift]
              v = speed of the receiver [m/s] [Tx stays still, Rx moves]
              fm = measured (or simulated) Tx frequency when moving [doppler shift included]
              ft = calculated theoretical frequency value where Rx would be moving straight to Tx [or
straign away from Tx <- it's not necessary to calculate the "moving away" value]
              dfm = value of the doppler shift of fm [dfm=fm-fgen;fm=fgen+dfm]
              dft = value of the theoretically calculated doppler shift of ft [dft=ft-fgen]
              vdf = doppler shift in fm [dfm] / calculated theoretical doppler shift in ft [dft];
[vdf=dfm/dft]
              facos = the angle between Tx and Rx; calculated by arccos-function [facos=acos(vdf)]
              dfm2 = used to observe if the angle of Rx and Tx changes correctly when moving to-
wards to Tx [fm frequency should not decrease if moving closer to Tx]
*****************************************************************************************************************
*******************************************************************************/
#include <Streaming.h>
#include <FreqPeriodCounter.h>

double dtow(double x, double y, double z); //functions
//double dawa(double x, double y, double z);
void ledscr(double x);
void ledsoff();

const byte counterPin = 3; // pin 3 = fgen (pin 3 is used for the measurement of the frequencies
with FreqPeriodCounter function)
const byte counterInterrupt = 1;

const byte dataPin=24, clockPin=25, latchPin=26, oePin1=27, oePin2=28; //D24=SI[DS],
D25=SCK[SH_CP], D26=RCK[ST_CP], D27=!G[!OE] for IC1, D28=!G[!OE] for IC2
const byte led17Pin=29, led18Pin=30; //D29=LED17[0 degree], D30=LED18[180 de-
grees/Txback]

FreqPeriodCounter counter(counterPin, micros, 0); //calculates the time that takes to HIGH-
LOW-period (or. vice versa) to determine T  -> f=1/T

void setup(void)
{
pinMode(dataPin, OUTPUT); //set pins to outputs to control the shift registers
pinMode(clockPin, OUTPUT);
pinMode(latchPin, OUTPUT);
pinMode(oePin1, OUTPUT); //IC1 output enable pin
pinMode(oePin2, OUTPUT); //IC2 output enable pin
pinMode(led17Pin, OUTPUT); //set pin to output for 0 degrees
pinMode(led18Pin, OUTPUT); //set pin to output for 180 degrees/Txback
Serial.begin(9600);
pinMode(9,OUTPUT); //pin 9 have to be connected to pin 3 to test with arduinos PWM-signal
analogWrite(9, 127); //pin 9 = 50% duty cycle (arduino PWM-signal in pin 9)
delay(500); //wait for stable PWM-signal
```

```
attachInterrupt(counterInterrupt, counterISR, CHANGE); //using interrupt for the frequency measurements
ledsoff(); //turns off all leds
}

void loop(void)
{
const double pi=3.14159265359, c=299792458;
double v=2.7778, fgen=0, fm=0, ft=0, dfm=0.00001, dft=0, facos=0, vdf=0, rcircled=0, dfm2=0; //attributes for calculating and measuring doppler shift
int rcircle=0, rcc=0;
bool reset=true, rcflag=true;
byte i=0;

if(counter.ready()) //tests that the counter function is ready
Serial << "counter_function: " << counter.hertz(1000) << "\n";

while(1)
{
fm=0; //formats the frequency result before new measurement/calculation

if(reset==true) //reset -> Tx [fgen] frequency measurement and averanging when still (without doppler shift) (used with external frequency source)
{
while(i<20)//averages the frequency measurement [accuracy 20 frequency results]
{
if(counter.ready())
{
fgen=fgen+(double)counter.hertz(1000)/1000; //frequency accuracy in mHz .000 Hz
i++;
Serial << "Freq_value [fgen] " << i << ": " << fgen << "\n"; //shows the frequency sums in serial monitor (while loop won't work without this command for some strange reason...)
}
else if(counter.elapsedTime > 50000) //detects if there is no frequency signal for 50ms
{
Serial << "No signal.\n";
ledsoff(); //turns off all leds
i=(byte)0;
fgen=0;
}
}
fgen=fgen/(double)i;
i=(byte)0;
reset=false;
fgen=fgen*1000000; //converts to higher frequency [ex. MHz] to see the doppler shift with Due PWM-signal or low frequency with function generator (BETA)
ft=dtow(fgen,c,v); //calculates the theoretical value of doppler frequency where Rx would be moving straight towards to Tx
dft=ft-fgen;
Serial << "-------------------------------------------------------------------------------------------------------------------\n";
}

fm=fgen+dfm; //this code is used when dfm value is set programmatically

/*while(i<10) //frequency measurements and averaging when moving [accuracy 10 frequency results] (this code section is used with external frequency source)
{
if(counter.ready())
{
```

```
fm=fm+(double)counter.hertz(1000)/1000; //frequency accuracy in mHz .000 Hz
i++;
Serial << "Freq_value [fm] " << i << ": " << fm << "\n"; //shows the frequency sums in Serial
Monitor (while loop won't work without this command for some strange reason...)
}
else if(counter.elapsedTime > 50000) //detects if there is no frequency signal for 50ms
{
Serial << "No signal.\n";
ledsoff(); //turns off all leds
i=(byte)0;
fm=0;
}
}
fm=fm/(double)i;
//fm=fm*1000000; //convert to higher frequency [ex. MHz] to see the doppler shift with Due pwm
signal or low frequency with function generator (BETA)
dfm=fm-fgen;
i=(byte)0;
Serial << "----------------------------------------------------------------------------------------------------------
----------\n";*/

/*if(fm>fgen) //this code section is not needed, because the Tx-away function (dawa()) will not
be used
ft=dtow(fgen,c,v);
else if(fm<fgen)
ft=dawa(fgen,c,v);
dft=ft-fgen;*/

vdf=dfm/dft; //measured (or simulated) doppler shift in fm [dfm=fm-fgen] / calculated theoretical
doppler shift in ft [dft=ft-fgen]
Serial << "dfm: " << dfm << " Hz, v: " << v << " m/s, fgen: " << fgen << " Hz, fm: " << fm << " Hz,
ft: " << ft << " Hz, vdf: " << vdf << "\n";

if(vdf>1) //converts vdf-value when vdf>1 (vdf is always a posit.value), [ex. if vdf=1.01 ->
(new)vdf=0.01||0.99, this code section changes the flow of full cycles]
{
Serial << "vdf_old: " << vdf << "\n";
rcircle=(int)vdf; //cycle count from double to int (how many full frequency cycles have happened
(ex. dfm=100 & dft=50 -> two cycles)
rcircled=(double)rcircle; //cycle count from int to double (converts to full cycles only, no decimal
numbers)
if(rcc!=rcircle&&rcircle>=2) //alternates the vdf-values calculation parametres of a vdf>1-state on
every second change when a full cycle happens (first cycle is not included)
rcflag=!rcflag;
if(rcflag==true)
vdf=1-(vdf-rcircled);
else
vdf=vdf-rcircled;
Serial << "rcircle: " << rcircle << ", rcircled: " << rcircled << ", vdf_new: " << vdf << "\n";
}

rcc=rcircle; //copies the value of rcircle to rcc, so that a full cycle change can be seen when
rcircles value changes

if(fm>fgen) //moving towards to Tx
{
Serial << "Heading to Tx.\n";
//dfm=dfm-0.000001; //angle repair test [-angles] (programmatically simulated dfm)
facos=acos(vdf)*180/pi;
```

```
Serial << "[Original angle] vdf: " << vdf << ", Acos: " << facos << " [+Tx_0]\n"; //shows the original angle in the serial monitor
if(dfm<dfm2&&vdf>=cos(45*pi/180)) //if dfm<dfm2 the Rx is going to the wrong way and needs to repair its direction, only angles that are <=45 degrees is repaired
{
facos=-facos*2; //ex. +44 degrees -> after direction repair -88 degrees
Serial << "[Repaired angle] vdf: " << vdf << ", Acos: " << facos << " [+Tx_1]\n"; //shows the repaired angle in the serial monitor
}
ledscr(facos); //function for LED screen
dfm=dfm+0.05; //increases the value of dfm, for testing the algorithm programmatically
if(dfm>=dft)
dfm=-0.001;
}
else if(fm<fgen) ////moving away from Tx. 1 led at the back of the led screen shows when moving away, so no need to calculate the angle of the Tx
{
Serial << "Heading away Tx.\n";
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, HIGH); //disables IC2 outputs
digitalWrite(led18Pin, HIGH); //set 180 degrees/backTx led on
digitalWrite(led17Pin, LOW); //set 0 degree led off
//facos=acos(vdf)*180/pi;
//Serial << "vdf: " << vdf << ", Acos: " << facos << " [-Tx_0]\n";
dfm=dfm-1; //decreases the value of dfm, for testing the algorithm programmatically
if(dfm<=-dft)
dfm=0.001;
}
else
{
Serial << "No doppler-frequency.\n";
ledsoff(); //turns off all leds
}
//delay(100);
Serial << "-----------------------------------------------------------------------------------------------------------------------\n";
dfm2=dfm; //copies the value of dfm to dfm2 to see if the last dfm(dfm2) is smaller than the new dfm of next calculation
}
}

double dtow(double x, double y, double z) //function for calculating theoretical value of frequency where Rx would be moving straight towards to Tx at certain speed
{
double vft=0;
vft=x*((y+z)/y);
return(vft);
}

/*double dawa(double x, double y, double z) //function for calculating theoretical value of frequency where Rx would be moving straight backwards from Tx at certain speed (this function is not necessary)
{
double vft=0;
vft=x*((y-z)/y);
return(vft);
}*/

void ledscr(double x) //fuction for controlling the led pointer display
{
```

```
if(x>-5.625&&x<5.625) //0' led17 on
{
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, HIGH); //disables IC2 outputs
digitalWrite(led17Pin, HIGH); //set 0' led on
digitalWrite(led18Pin, LOW); //set 180'/backTx led off
}
else
{
digitalWrite(led17Pin, LOW); //set 0' led off
digitalWrite(led18Pin, LOW); //set 180'/backTx led off
if(x>0&&x>=5.625) //IC2 enable
{
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, LOW); //enables IC2 outputs
}
else if(x<0&&x<=-5.625)//IC1 enable
{
digitalWrite(oePin2, HIGH); //disables IC2 outputs
digitalWrite(oePin1, LOW); //enables IC1 outputs
}
else
{
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, HIGH); //disables IC2 outputs
}
digitalWrite(latchPin, LOW); //latch pin low so led's don't change while sending bits
if(x<0)
{
if(x>=-90.000&&x<=-84.375) //LED1 -90' [leds 1-8 negat.']
shiftOut(dataPin, clockPin, MSBFIRST, 0x01);
else if(x>-84.375&&x<=-73.125) //LED2 -78.75'
shiftOut(dataPin, clockPin, MSBFIRST, 0x02);
else if(x>-73.125&&x<=-61.875) //LED3 -67.5'
shiftOut(dataPin, clockPin, MSBFIRST, 0x04);
else if(x>-61.875&&x<=-50.625) //LED4 -56.25'
shiftOut(dataPin, clockPin, MSBFIRST, 0x08);
else if(x>-50.652&&x<=-39.375) //LED5 -45'
shiftOut(dataPin, clockPin, MSBFIRST, 0x10);
else if(x>-39.375&&x<=-28.125) //LED6 -33.75'
shiftOut(dataPin, clockPin, MSBFIRST, 0x20);
else if(x>-28.125&&x<=-16.875) //LED7 -22.5'
shiftOut(dataPin, clockPin, MSBFIRST, 0x40);
else if(x>-16.875&&x<=-5.625) //LED8 -11.25'
shiftOut(dataPin, clockPin, MSBFIRST, 0x80);
}
else
{
if(x<16.875&&x>=5.625) //LED9 +11.25' [leds 9-16 posit.']
shiftOut(dataPin, clockPin, MSBFIRST, 0x01);
else if(x<28.125&&x>=16.875) //LED10 +22.5'
shiftOut(dataPin, clockPin, MSBFIRST, 0x02);
else if(x<39.375&&x>=28.125) //LED11 +33.75'
shiftOut(dataPin, clockPin, MSBFIRST, 0x04);
else if(x<50.652&&x>=39.375) //LED12 +45'
shiftOut(dataPin, clockPin, MSBFIRST, 0x08);
else if(x<61.875&&x>=50.625) //LED13 +56.25'
shiftOut(dataPin, clockPin, MSBFIRST, 0x10);
else if(x<73.125&&x>=61.875) //LED14 +67.5'
shiftOut(dataPin, clockPin, MSBFIRST, 0x20);
```

```
else if(x<84.375&&x>=73.125) //LED15 +78.75'
shiftOut(dataPin, clockPin, MSBFIRST, 0x40);
else if(x<=90.000&&x>=84.375) //LED16 +90'
shiftOut(dataPin, clockPin, MSBFIRST, 0x80);
}
digitalWrite(latchPin, HIGH); //latch pin high so leds will light up
}
}

void ledsoff() //function to turn all leds off
{
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, HIGH); //disables IC2 outputs
digitalWrite(led17Pin, LOW); //set 0' led off
digitalWrite(led18Pin, LOW); //set 180'/backTx led off
}

void counterISR() //frequency counter function
{
counter.poll();
}
```

```
/*************************************************************************************************************************************
*************************************************************************************
Project : Doppler-bearing algorithm v.1.0 (programmatically only test)
Hardware: Arduino Due [AT91SAM3X8E]
Software: Arduino software
Date    : 14.05.2014
Author  : Matti Rekis
Changes : Added code for LED screen and for averaging frequency measurements. Removed
the "moving away"-value calculation of the doppler-shift (dawa()).
Comments: Frontside leds in half circle and one led at back pointing if moving away from Tx.
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------
Using 64 bit double attributes for calculations/measurements.
Attributes  : fgen = original Tx frequency for reference [when still; no doppler shift]
              v = speed of the receiver [m/s] [Tx stays still, Rx moves]
              fm = measured (or simulated) Tx frequency when moving [doppler shift included]
              ft = calculated theoretical frequency value where Rx would be moving straight to Tx [or
straign away from Tx <- it's not necessary to calculate the "moving away" value]
              dfm = value of the doppler shift of fm [dfm=fm-fgen;fm=fgen+dfm]
              dft = value of the theoretically calculated doppler shift of ft [dft=ft-fgen]
              vdf = doppler shift in fm [dfm] / calculated theoretical doppler shift in ft [dft];
[vdf=dfm/dft]
              facos = the angle between Tx and Rx; calculated by arccos-function [facos=acos(vdf)]
              dfm2 = used to observe if the angle of Rx and Tx changes correctly when moving to-
wards to Tx [fm frequency should not decrease if moving closer to Tx]
*************************************************************************************************************************************
*************************************************************************************/
#include <Streaming.h>
//#include <FreqPeriodCounter.h>

double dtow(double x, double y, double z); //functions
//double dawa(double x, double y, double z);
void ledscr(double x);
void ledsoff();

//const byte counterPin = 3; // pin 3 = fgen (pin 3 is used for the measurement of the frequencies
with FreqPeriodCounter function)
//const byte counterInterrupt = 1;

const byte dataPin=24, clockPin=25, latchPin=26, oePin1=27, oePin2=28; //D24=SI[DS],
D25=SCK[SH_CP], D26=RCK[ST_CP], D27=!G[!OE] for IC1, D28=!G[!OE] for IC2
const byte led17Pin=29, led18Pin=30; //D29=LED17[0 degree], D30=LED18[180 de-
grees/Txback]

//FreqPeriodCounter counter(counterPin, micros, 0); //calculates the time that takes to HIGH-
LOW-period (or. vice versa) to determine T  -> f=1/T

void setup(void)
{
pinMode(dataPin, OUTPUT); //set pins to outputs to control the shift registers
pinMode(clockPin, OUTPUT);
pinMode(latchPin, OUTPUT);
pinMode(oePin1, OUTPUT); //IC1 output enable pin
pinMode(oePin2, OUTPUT); //IC2 output enable pin
pinMode(led17Pin, OUTPUT); //set pin to output for 0 degrees
pinMode(led18Pin, OUTPUT); //set pin to output for 180 degrees/Txback
Serial.begin(9600);
//pinMode(9,OUTPUT); //pin 9 have to be connected to pin 3 to test with arduinos PWM-signal
//analogWrite(9, 127); //pin 9 = 50% duty cycle (arduino PWM-signal in pin 9)
//delay(500); //wait for stable PWM-signal
```

```
//attachInterrupt(counterInterrupt, counterISR, CHANGE); //using interrupt for the frequency
measurements
ledsoff(); //turns off all leds
}

void loop(void)
{
const double pi=3.14159265359, c=299792458;
double v=2.7778, fgen=900000000, fm=0, ft=0, dfm=0.001, dft=0, facos=0, vdf=0, rcircled=0,
dfm2=0; //attributes for calculating and measuring doppler shift
int rcircle=0, rcc=0;
bool reset=true, rcflag=true;
byte i=0;

/*if(counter.ready()) //tests that the counter function is ready
Serial << "counter_function: " << counter.hertz(1000) << "\n";*/

while(1)
{
fm=0; //formats the frequency result before new measurement/calculation

/*if(reset==true) //reset -> Tx [fgen] frequency measurement and averanging when still (without
doppler shift) (used with external frequency source)
{
while(i<20)//averages the frequency measurement [accuracy 20 frequency results]
{
if(counter.ready())
{
fgen=fgen+(double)counter.hertz(1000)/1000; //frequency accuracy in mHz .000 Hz
i++;
Serial << "Freq_value [fgen] " << i << ": " << fgen << "\n"; //shows the frequency sums in serial
monitor (while loop won't work without this command for some strange reason...)
}
else if(counter.elapsedTime > 50000) //detects if there is no frequency signal for 50ms
{
Serial << "No signal.\n";
ledsoff(); //turns off all leds
i=(byte)0;
fgen=0;
}
}
fgen=fgen/(double)i;
i=(byte)0;
reset=false;
//fgen=fgen*1000000; //converts to higher frequency [ex. MHz] to see the doppler shift with Due
PWM-signal or low frequency with function generator (BETA)
ft=dtow(fgen,c,v); //calculates the theoretical value of doppler frequency where Rx would be
moving straight towards to Tx
dft=ft-fgen;
Serial << "----------------------------------------------------------------------------------------------------
----------\n";
}*/

ft=dtow(fgen,c,v);
dft=ft-fgen;
fm=fgen+dfm; //this code is used when dfm value is set programmatically

/*while(i<10) //frequency measurements and averaging when moving [accuracy 10 frequency
results] (this code section is used with external frequency source)
{
```

```
if(counter.ready())
{
fm=fm+(double)counter.hertz(1000)/1000; //frequency accuracy in mHz .000 Hz
i++;
Serial << "Freq_value [fm] " << i << ": " << fm << "\n"; //shows the frequency sums in Serial
Monitor (while loop won't work without this command for some strange reason...)
}
else if(counter.elapsedTime > 50000) //detects if there is no frequency signal for 50ms
{
Serial << "No signal.\n";
ledsoff(); //turns off all leds
i=(byte)0;
fm=0;
}
}
fm=fm/(double)i;
//fm=fm*1000000; //convert to higher frequency [ex. MHz] to see the doppler shift with Due pwm
signal or low frequency with function generator (BETA)
dfm=fm-fgen;
i=(byte)0;
Serial << "----------------------------------------------------------------------------------------------------------
----------\n";*/

/*if(fm>fgen) //this code section is not needed, because the Tx-away function (dawa()) will not
be used
ft=dtow(fgen,c,v);
else if(fm<fgen)
ft=dawa(fgen,c,v);
dft=ft-fgen;*/

vdf=dfm/dft; //measured (or simulated) doppler shift in fm [dfm=fm-fgen] / calculated theoretical
doppler shift in ft [dft=ft-fgen]
Serial << "dfm: " << dfm << " Hz, v: " << v << " m/s, fgen: " << fgen << " Hz, fm: " << fm << " Hz,
ft: " << ft << " Hz, vdf: " << vdf << "\n";

if(vdf>1) //converts vdf-value when vdf>1 (vdf is always a posit.value), [ex. if vdf=1.01 ->
(new)vdf=0.01||0.99, this code section changes the flow of full cycles]
{
Serial << "vdf_old: " << vdf << "\n";
rcircle=(int)vdf; //cycle count from double to int (how many full frequency cycles have happened
(ex. dfm=100 & dft=50 -> two cycles)
rcircled=(double)rcircle; //cycle count from int to double (converts to full cycles only, no decimal
numbers)
if(rcc!=rcircle&&rcircle>=2) //alternates the vdf-values calculation parametres of a vdf>1-state on
every second change when a full cycle happens (first cycle is not included)
rcflag=!rcflag;
if(rcflag==true)
vdf=1-(vdf-rcircled);
else
vdf=vdf-rcircled;
Serial << "rcircle: " << rcircle << ", rcircled: " << rcircled << ", vdf_new: " << vdf << "\n";
}

rcc=rcircle; //copies the value of rcircle to rcc, so that a full cycle change can be seen when
rcircles value changes

if(fm>fgen) //moving towards to Tx
{
Serial << "Heading to Tx.\n";
dfm=dfm-0.000001; //angle repair test [-angles] (programmatically simulated dfm)
```

```
facos=acos(vdf)*180/pi;
Serial << "[Original angle] vdf: " << vdf << ", Acos: " << facos << " [+Tx_0]\n"; //shows the original angle in the serial monitor
if(dfm<dfm2&&vdf>=cos(45*pi/180)) //if dfm<dfm2 the Rx is going to the wrong way and needs to repair its direction, only angles that are <=45 degrees is repaired
{
facos=-facos*2; //ex. +44 degrees -> after direction repair -88 degrees
Serial << "[Repaired angle] vdf: " << vdf << ", Acos: " << facos << " [+Tx_1]\n"; //shows the repaired angle in the serial monitor
}
ledscr(facos); //function for LED screen
dfm=dfm+0.02; //increases the value of dfm, for testing the algorithm programmatically
if(dfm>=(dft*4)) //for testing the cycle calculation code when vdf>1
dfm=-0.001;
}
else if(fm<fgen) ////moving away from Tx. 1 led at the back of the led screen shows when moving away, so no need to calculate the angle of the Tx
{
Serial << "Heading away Tx.\n";
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, HIGH); //disables IC2 outputs
digitalWrite(led18Pin, HIGH); //set 180 degrees/backTx led on
digitalWrite(led17Pin, LOW); //set 0 degree led off
//facos=acos(vdf)*180/pi;
//Serial << "vdf: " << vdf << ", Acos: " << facos << " [-Tx_0]\n";
dfm=dfm-1; //decreases the value of dfm, for testing the algorithm programmatically
if(dfm<=(dft*-2))
dfm=0.001;
}
else
{
Serial << "No doppler-frequency.\n";
ledsoff(); //turns off all leds
}
//delay(100);
Serial << "-------------------------------------------------------------------------------------------------------------------------------\n";
dfm2=dfm; //copies the value of dfm to dfm2 to see if the last dfm(dfm2) is smaller than the new dfm of next calculation
}
}

double dtow(double x, double y, double z) //function for calculating theoretical value of frequency where Rx would be moving straight towards to Tx at certain speed
{
double vft=0;
vft=x*((y+z)/y);
return(vft);
}

/*double dawa(double x, double y, double z) //function for calculating theoretical value of frequency where Rx would be moving straight backwards from Tx at certain speed (this function is not necessary)
{
double vft=0;
vft=x*((y-z)/y);
return(vft);
}*/

void ledscr(double x) //fuction for controlling the led pointer display
```

```
{
if(x>-5.625&&x<5.625) //0' led17 on
{
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, HIGH); //disables IC2 outputs
digitalWrite(led17Pin, HIGH); //set 0' led on
digitalWrite(led18Pin, LOW); //set 180'/backTx led off
}
else
{
digitalWrite(led17Pin, LOW); //set 0' led off
digitalWrite(led18Pin, LOW); //set 180'/backTx led off
if(x>0&&x>=5.625) //IC2 enable
{
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, LOW); //enables IC2 outputs
}
else if(x<0&&x<=-5.625)//IC1 enable
{
digitalWrite(oePin2, HIGH); //disables IC2 outputs
digitalWrite(oePin1, LOW); //enables IC1 outputs
}
else
{
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, HIGH); //disables IC2 outputs
}
digitalWrite(latchPin, LOW); //latch pin low so led's don't change while sending bits
if(x<0)
{
if(x>=-90.000&&x<=-84.375) //LED1 -90' [leds 1-8 negat.']
shiftOut(dataPin, clockPin, MSBFIRST, 0x01);
else if(x>-84.375&&x<=-73.125) //LED2 -78.75'
shiftOut(dataPin, clockPin, MSBFIRST, 0x02);
else if(x>-73.125&&x<=-61.875) //LED3 -67.5'
shiftOut(dataPin, clockPin, MSBFIRST, 0x04);
else if(x>-61.875&&x<=-50.625) //LED4 -56.25'
shiftOut(dataPin, clockPin, MSBFIRST, 0x08);
else if(x>-50.652&&x<=-39.375) //LED5 -45'
shiftOut(dataPin, clockPin, MSBFIRST, 0x10);
else if(x>-39.375&&x<=-28.125) //LED6 -33.75'
shiftOut(dataPin, clockPin, MSBFIRST, 0x20);
else if(x>-28.125&&x<=-16.875) //LED7 -22.5'
shiftOut(dataPin, clockPin, MSBFIRST, 0x40);
else if(x>-16.875&&x<=-5.625) //LED8 -11.25'
shiftOut(dataPin, clockPin, MSBFIRST, 0x80);
}
else
{
if(x<16.875&&x>=5.625) //LED9 +11.25' [leds 9-16 posit.']
shiftOut(dataPin, clockPin, MSBFIRST, 0x01);
else if(x<28.125&&x>=16.875) //LED10 +22.5'
shiftOut(dataPin, clockPin, MSBFIRST, 0x02);
else if(x<39.375&&x>=28.125) //LED11 +33.75'
shiftOut(dataPin, clockPin, MSBFIRST, 0x04);
else if(x<50.652&&x>=39.375) //LED12 +45'
shiftOut(dataPin, clockPin, MSBFIRST, 0x08);
else if(x<61.875&&x>=50.625) //LED13 +56.25'
shiftOut(dataPin, clockPin, MSBFIRST, 0x10);
else if(x<73.125&&x>=61.875) //LED14 +67.5'
```

```
shiftOut(dataPin, clockPin, MSBFIRST, 0x20);
else if(x<84.375&&x>=73.125) //LED15 +78.75'
shiftOut(dataPin, clockPin, MSBFIRST, 0x40);
else if(x<=90.000&&x>=84.375) //LED16 +90'
shiftOut(dataPin, clockPin, MSBFIRST, 0x80);
}
digitalWrite(latchPin, HIGH); //latch pin high so leds will light up
}
}

void ledsoff() //function to turn all leds off
{
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, HIGH); //disables IC2 outputs
digitalWrite(led17Pin, LOW); //set 0' led off
digitalWrite(led18Pin, LOW); //set 180'/backTx led off
}

/*void counterISR() //frequency counter function
{
counter.poll();
}*/
```

```
/**************************************************************************************************************************************************
*******************************************************************************
Project : Doppler-bearing algorithm v.1.0 [superheterodyne] (external frequency source)
Hardware: Arduino Due [AT91SAM3X8E]
Software: Arduino software
Date    : 23.05.2014
Author  : Matti Rekis
Changes : Added code for LED screen and for averaging frequency measurements. Removed
the "moving away"-value calculation of the doppler-shift (dawa()).
Comments: Frontside leds in half circle and one led at back pointing if moving away from Tx.
-------------------------------------------------------------------------------------------------------------------------------
Using 64 bit double attributes for calculations/measurements.
Attributes  : forig = original Tx frequency (programmatically simulated) [when still; no doppler
shift]
          fgen = superheterodyne Tx frequency for reference [when still; no doppler shift]
          v = speed of the receiver [m/s] [Tx stays still, Rx moves]
          fm = measured (or simulated) superheterodyne Tx frequency when moving [doppler
shift included]
          ft = calculated theoretical frequency value where Rx would be moving straight to Tx [or
straign away from Tx <- it's not necessary to calculate the "moving away" value]
          dfm = value of the doppler shift of fm [dfm=fm-fgen;fm=fgen+dfm]
          dft = value of the theoretically calculated doppler shift of ft [dft=ft-forig]
          vdf = doppler shift in fm [dfm] / calculated theoretical doppler shift in ft [dft];
[vdf=dfm/dft]
          facos = the angle between Tx and Rx; calculated by arccos-function [facos=acos(vdf)]
          dfm2 = used to observe if the angle of Rx and Tx changes correctly when moving to-
wards to Tx [fm frequency should not decrease if moving closer to Tx]
**************************************************************************************************************************************************
*******************************************************************************/
#include <Streaming.h>
#include <FreqPeriodCounter.h>

double dtow(double x, double y, double z); //functions
//double dawa(double x, double y, double z);
void ledscr(double x);
void ledsoff();

const byte counterPin = 3; // pin 3 = fgen (pin 3 is used for the measurement of the frequencies
with FreqPeriodCounter function)
const byte counterInterrupt = 1;

const byte dataPin=24, clockPin=25, latchPin=26, oePin1=27, oePin2=28; //D24=SI[DS],
D25=SCK[SH_CP], D26=RCK[ST_CP], D27=!G[!OE] for IC1, D28=!G[!OE] for IC2
const byte led17Pin=29, led18Pin=30; //D29=LED17[0 degree], D30=LED18[180 de-
grees/Txback]

FreqPeriodCounter counter(counterPin, micros, 0); //calculates the time that takes to HIGH-
LOW-period (or. vice versa) to determine T  -> f=1/T

void setup(void)
{
pinMode(dataPin, OUTPUT); //set pins to outputs to control the shift registers
pinMode(clockPin, OUTPUT);
pinMode(latchPin, OUTPUT);
pinMode(oePin1, OUTPUT); //IC1 output enable pin
pinMode(oePin2, OUTPUT); //IC2 output enable pin
pinMode(led17Pin, OUTPUT); //set pin to output for 0 degrees
pinMode(led18Pin, OUTPUT); //set pin to output for 180 degrees/Txback
Serial.begin(9600);
```

```
pinMode(9,OUTPUT); //pin 9 have to be connected to pin 3 to test with arduinos PWM-signal
analogWrite(9, 127); //pin 9 = 50% duty cycle (arduino PWM-signal in pin 9)
delay(500); //wait for stable PWM-signal
attachInterrupt(counterInterrupt, counterISR, CHANGE); //using interrupt for the frequency
measurements
ledsoff(); //turns off all leds
}

void loop(void)
{
const double pi=3.14159265359, c=299792458;
double v=2.7778, forig=2400000000, fgen=0, fm=0, ft=0, dfm=0, dft=0, facos=0, vdf=0,
rcircled=0, dfm2=0; //attributes for calculating and measuring doppler shift
int rcircle=0, rcc=0;
bool reset=true, rcflag=true;
byte i=0;

ft=dtow(forig,c,v); //calculates the theoretical value of doppler frequency when Rx would be
moving straight towards to Tx
dft=ft-forig; //the doppler shift of ft

if(counter.ready()) //tests that the counter function is ready
Serial << "counter_function: " << counter.hertz(1000) << "\n";

while(1)
{
fm=0; //formats the frequency result before new measurement/calculation

if(reset==true) //reset -> Tx [fgen] frequency measurement and averanging when still (without
doppler shift) (used with external frequency source)
{
while(i<50)//averages the frequency measurement [accuracy 50 frequency results]
{
if(counter.ready())
{
fgen=fgen+(double)counter.hertz(1000)/1000; //frequency accuracy in mHz .000 Hz
i++;
Serial << "Freq_value [fgen] " << i << ": " << fgen << "\n"; //shows the frequency sums in serial
monitor (while loop won't work without this command for some strange reason...)
}
else if(counter.elapsedTime > 50000) //detects if there is no frequency signal for 50ms
{
Serial << "No signal.\n";
ledsoff(); //turns off all leds
i=(byte)0;
fgen=0;
}
}
fgen=fgen/(double)i;
i=(byte)0;
reset=false;
Serial << "----------------------------------------------------------------------------------------------------------
----------\n";
}

//fm=fgen+dfm; //this code is used when dfm value is set programmatically

while(i<50) //frequency measurements and averaging when moving [accuracy 50 frequency
results] (this code section is used with external frequency source)
{
```

```
if(counter.ready())
{
fm=fm+(double)counter.hertz(1000)/1000; //frequency accuracy in mHz .000 Hz
i++;
Serial << "Freq_value [fm] " << i << ": " << fm << "\n"; //shows the frequency sums in Serial
Monitor (while loop won't work without this command for some strange reason...)
}
else if(counter.elapsedTime > 50000) //detects if there is no frequency signal for 50ms
{
Serial << "No signal.\n";
ledsoff(); //turns off all leds
i=(byte)0;
fm=0;
}
}
fm=fm/(double)i;
dfm=fm-fgen;
i=(byte)0;
Serial << "-----------------------------------------------------------------------------------------------------------------
----------\n";

/*if(fm>fgen) //this code section is not needed, because the Tx-away function (dawa()) will not
be used
ft=dtow(forig,c,v);
else if(fm<fgen)
ft=dawa(forig,c,v);
dft=ft-forig;*/

vdf=dfm/dft; //measured (or simulated) doppler shift in fm [dfm=fm-fgen] / calculated theoretical
doppler shift in ft [dft=ft-forig]
Serial << "dfm: " << dfm << " Hz, v: " << v << " m/s, fgen: " << fgen << " Hz, fm: " << fm << " Hz,
ft: " << ft << " Hz, vdf: " << vdf << "\n";

/*if(vdf>1) //converts vdf-value when vdf>1 (vdf is always a posit.value), [ex. if vdf=1.01 ->
(new)vdf=0.01||0.99, this code section changes the flow of full cycles]
{
Serial << "vdf_old: " << vdf << "\n";
rcircle=(int)vdf; //cycle count from double to int (how many full frequency cycles have happened
(ex. dfm=100 & dft=50 -> two cycles)
rcircled=(double)rcircle; //cycle count from int to double (converts to full cycles only, no decimal
numbers)
if(rcc!=rcircle&&rcircle>=2) //alternates the vdf-values calculation parametres of a vdf>1-state on
every second change when a full cycle happens (first cycle is not included)
rcflag=!rcflag;
if(rcflag==true)
vdf=1-(vdf-rcircled);
else
vdf=vdf-rcircled;
Serial << "rcircle: " << rcircle << ", rcircled: " << rcircled << ", vdf_new: " << vdf << "\n";
}

rcc=rcircle; //copies the value of rcircle to rcc, so that a full cycle change can be seen when
rcircles value changes*/

if(fm>fgen&&vdf<=1&&vdf>=0) //moving towards to Tx
{
Serial << "Heading to Tx.\n";
//dfm=dfm-0.000001; //angle repair test [-angles] (programmatically simulated dfm)
facos=acos(vdf)*180/pi;
```

```
Serial << "[Original angle] vdf: " << vdf << ", Acos: " << facos << " [+Tx_0]\n"; //shows the original angle in the serial monitor
if(dfm<(dfm2-1)&&vdf>=cos(45*pi/180)) //if dfm<dfm2 the Rx is going to the wrong way and needs to repair its direction, only angles that are <=45 degrees is repaired (in dfm2-1 the 1 stands for how many Hz smaller the dfm2 have to be to dfm)
{
facos=-facos*2; //ex. +44 degrees -> after direction repair -88 degrees
Serial << "[Repaired angle] vdf: " << vdf << ", Acos: " << facos << " [+Tx_1]\n"; //shows the repaired angle in the serial monitor
}
ledscr(facos); //function for LED screen
//dfm=dfm+0.1; //increases the value of dfm, for testing the algorithm programmatically
//if(dfm>=(dft*4)) //for testing the cycle calculation code when vdf>1
//dfm=-0.001;
}
else if(fm<fgen&&vdf>=-1&&vdf<=1) //moving away from Tx. 1 led at the back of the led screen shows when moving away, so no need to calculate the angle of the Tx
{
Serial << "Heading away Tx.\n";
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, HIGH); //disables IC2 outputs
digitalWrite(led18Pin, HIGH); //set 180 degrees/backTx led on
digitalWrite(led17Pin, LOW); //set 0 degree led off
//facos=acos(vdf)*180/pi;
//Serial << "vdf: " << vdf << ", Acos: " << facos << " [-Tx_0]\n";
//dfm=dfm-1; //decreases the value of dfm, for testing the algorithm programmatically
//if(dfm<=(dft*-2))
//dfm=0.001;
}
else
{
Serial << "No doppler-frequency.\n";
ledsoff(); //turns off all leds
}
//delay(100);
Serial << "-----------------------------------------------------------------------------------------------------------------------\n";
dfm2=dfm; //copies the value of dfm to dfm2 to see if the last dfm(dfm2) is smaller than the new dfm of next calculation
}
}

double dtow(double x, double y, double z) //function for calculating theoretical value of frequency where Rx would be moving straight towards to Tx at certain speed
{
double vft=0;
vft=x*((y+z)/y);
return(vft);
}

/*double dawa(double x, double y, double z) //function for calculating theoretical value of frequency where Rx would be moving straight backwards from Tx at certain speed (this function is not necessary)
{
double vft=0;
vft=x*((y-z)/y);
return(vft);
}*/

void ledscr(double x) //fuction for controlling the led pointer display
```

```
{
if(x>-5.625&&x<5.625) //0' led17 on
{
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, HIGH); //disables IC2 outputs
digitalWrite(led17Pin, HIGH); //set 0' led on
digitalWrite(led18Pin, LOW); //set 180'/backTx led off
}
else
{
digitalWrite(led17Pin, LOW); //set 0' led off
digitalWrite(led18Pin, LOW); //set 180'/backTx led off
if(x>0&&x>=5.625) //IC2 enable
{
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, LOW); //enables IC2 outputs
}
else if(x<0&&x<=-5.625)//IC1 enable
{
digitalWrite(oePin2, HIGH); //disables IC2 outputs
digitalWrite(oePin1, LOW); //enables IC1 outputs
}
else
{
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, HIGH); //disables IC2 outputs
}
digitalWrite(latchPin, LOW); //latch pin low so led's don't change while sending bits
if(x<0)
{
if(x>=-90.000&&x<=-84.375) //LED1 -90' [leds 1-8 negat.']
shiftOut(dataPin, clockPin, MSBFIRST, 0x01);
else if(x>-84.375&&x<=-73.125) //LED2 -78.75'
shiftOut(dataPin, clockPin, MSBFIRST, 0x02);
else if(x>-73.125&&x<=-61.875) //LED3 -67.5'
shiftOut(dataPin, clockPin, MSBFIRST, 0x04);
else if(x>-61.875&&x<=-50.625) //LED4 -56.25'
shiftOut(dataPin, clockPin, MSBFIRST, 0x08);
else if(x>-50.652&&x<=-39.375) //LED5 -45'
shiftOut(dataPin, clockPin, MSBFIRST, 0x10);
else if(x>-39.375&&x<=-28.125) //LED6 -33.75'
shiftOut(dataPin, clockPin, MSBFIRST, 0x20);
else if(x>-28.125&&x<=-16.875) //LED7 -22.5'
shiftOut(dataPin, clockPin, MSBFIRST, 0x40);
else if(x>-16.875&&x<=-5.625) //LED8 -11.25'
shiftOut(dataPin, clockPin, MSBFIRST, 0x80);
}
else
{
if(x<16.875&&x>=5.625) //LED9 +11.25' [leds 9-16 posit.']
shiftOut(dataPin, clockPin, MSBFIRST, 0x01);
else if(x<28.125&&x>=16.875) //LED10 +22.5'
shiftOut(dataPin, clockPin, MSBFIRST, 0x02);
else if(x<39.375&&x>=28.125) //LED11 +33.75'
shiftOut(dataPin, clockPin, MSBFIRST, 0x04);
else if(x<50.652&&x>=39.375) //LED12 +45'
shiftOut(dataPin, clockPin, MSBFIRST, 0x08);
else if(x<61.875&&x>=50.625) //LED13 +56.25'
shiftOut(dataPin, clockPin, MSBFIRST, 0x10);
else if(x<73.125&&x>=61.875) //LED14 +67.5'
```

```
shiftOut(dataPin, clockPin, MSBFIRST, 0x20);
else if(x<84.375&&x>=73.125) //LED15 +78.75'
shiftOut(dataPin, clockPin, MSBFIRST, 0x40);
else if(x<=90.000&&x>=84.375) //LED16 +90'
shiftOut(dataPin, clockPin, MSBFIRST, 0x80);
}
digitalWrite(latchPin, HIGH); //latch pin high so leds will light up
}
}

void ledsoff() //function to turn all leds off
{
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, HIGH); //disables IC2 outputs
digitalWrite(led17Pin, LOW); //set 0' led off
digitalWrite(led18Pin, LOW); //set 180'/backTx led off
}

void counterISR() //frequency counter function
{
counter.poll();
}
```

```
/*************************************************************************************************************
*****************************************************************************
Project : Doppler-bearing algorithm v.1.0 [superheterodyne] (programmatically only
test,+angles)
Hardware: Arduino Due [AT91SAM3X8E]
Software: Arduino software
Date    : 23.05.2014
Author  : Matti Rekis
Changes : Added code for LED screen and for averaging frequency measurements. Removed
the "moving away"-value calculation of the doppler-shift (dawa()).
Comments: Frontside leds in half circle and one led at back pointing if moving away from Tx.
-------------------------------------------------------------------------------------------------------------
-----------------------------------------
Using 64 bit double attributes for calculations/measurements.
Attributes  : forig = original Tx frequency (programmatically simulated) [when still; no doppler
shift]
        fgen = superheterodyne Tx frequency for reference [when still; no doppler shift]
        v = speed of the receiver [m/s] [Tx stays still, Rx moves]
        fm = measured (or simulated) superheterodyne Tx frequency when moving [doppler
shift included]
        ft = calculated theoretical frequency value where Rx would be moving straight to Tx [or
straign away from Tx <- it's not necessary to calculate the "moving away" value]
        dfm = value of the doppler shift of fm [dfm=fm-fgen;fm=fgen+dfm]
        dft = value of the theoretically calculated doppler shift of ft [dft=ft-forig]
        vdf = doppler shift in fm [dfm] / calculated theoretical doppler shift in ft [dft];
[vdf=dfm/dft]
        facos = the angle between Tx and Rx; calculated by arccos-function [facos=acos(vdf)]
        dfm2 = used to observe if the angle of Rx and Tx changes correctly when moving to-
wards to Tx [fm frequency should not decrease if moving closer to Tx]
*************************************************************************************************************
*****************************************************************************/
#include <Streaming.h>
#include <FreqPeriodCounter.h>

double dtow(double x, double y, double z); //functions
//double dawa(double x, double y, double z);
void ledscr(double x);
void ledsoff();

const byte counterPin = 3; // pin 3 = fgen (pin 3 is used for the measurement of the frequencies
with FreqPeriodCounter function)
const byte counterInterrupt = 1;

const byte dataPin=24, clockPin=25, latchPin=26, oePin1=27, oePin2=28; //D24=SI[DS],
D25=SCK[SH_CP], D26=RCK[ST_CP], D27=!G[!OE] for IC1, D28=!G[!OE] for IC2
const byte led17Pin=29, led18Pin=30; //D29=LED17[0 degree], D30=LED18[180 de-
grees/Txback]

FreqPeriodCounter counter(counterPin, micros, 0); //calculates the time that takes to HIGH-
LOW-period (or. vice versa) to determine T  -> f=1/T

void setup(void)
{
pinMode(dataPin, OUTPUT); //set pins to outputs to control the shift registers
pinMode(clockPin, OUTPUT);
pinMode(latchPin, OUTPUT);
pinMode(oePin1, OUTPUT); //IC1 output enable pin
pinMode(oePin2, OUTPUT); //IC2 output enable pin
pinMode(led17Pin, OUTPUT); //set pin to output for 0 degrees
pinMode(led18Pin, OUTPUT); //set pin to output for 180 degrees/Txback
```

```
Serial.begin(9600);
pinMode(9,OUTPUT); //pin 9 have to be connected to pin 3 to test with arduinos PWM-signal
analogWrite(9, 127); //pin 9 = 50% duty cycle (arduino PWM-signal in pin 9)
delay(500); //wait for stable PWM-signal
attachInterrupt(counterInterrupt, counterISR, CHANGE); //using interrupt for the frequency
measurements
ledsoff(); //turns off all leds
}

void loop(void)
{
const double pi=3.14159265359, c=299792458;
double v=2.7778, forig=2400000000, fgen=100, fm=0, ft=0, dfm=0.001, dft=0, facos=0, vdf=0,
rcircled=0, dfm2=0; //attributes for calculating and measuring doppler shift
int rcircle=0, rcc=0;
bool reset=true, rcflag=true;
byte i=0;

ft=dtow(forig,c,v); //calculates the theoretical value of doppler frequency when Rx would be
moving straight towards to Tx
dft=ft-forig; //the doppler shift of ft

//if(counter.ready()) //tests that the counter_function is ready
//Serial << "counter_function: " << counter.hertz(1000) << "\n";

while(1)
{
fm=0; //formats the frequency result before new measurement/calculation

/*if(reset==true) //reset -> Tx [fgen] frequency measurement and averanging when still (without
doppler shift) (used with external frequency source)
{
while(i<20)//averages the frequency measurement [accuracy 20 frequency results]
{
if(counter.ready())
{
fgen=fgen+(double)counter.hertz(1000)/1000; //frequency accuracy in mHz .000 Hz
i++;
Serial << "Freq_value [fgen] " << i << ": " << fgen << "\n"; //shows the frequency sums in serial
monitor (while loop won't work without this command for some strange reason...)
}
else if(counter.elapsedTime > 50000) //detects if there is no frequency signal for 50ms
{
Serial << "No signal.\n";
ledsoff(); //turns off all leds
i=(byte)0;
fgen=0;
}
}
fgen=fgen/(double)i;
i=(byte)0;
reset=false;
Serial << "----------------------------------------------------------------------------------------------------
----------\n";
}*/

fm=fgen+dfm; //this code is used when dfm value is set programmatically

/*while(i<10) //frequency measurements and averaging when moving [accuracy 10 frequency
results] (this code section is used with external frequency source)
```

```
{
if(counter.ready())
{
fm=fm+(double)counter.hertz(1000)/1000; //frequency accuracy in mHz .000 Hz
i++;
Serial << "Freq_value [fm] " << i << ": " << fm << "\n"; //shows the frequency sums in Serial
Monitor (while loop won't work without this command for some strange reason...)
}
else if(counter.elapsedTime > 50000) //detects if there is no frequency signal for 50ms
{
Serial << "No signal.\n";
ledsoff(); //turns off all leds
i=(byte)0;
fm=0;
}
}
fm=fm/(double)i;
dfm=fm-fgen;
i=(byte)0;
Serial << "-----------------------------------------------------------------------------------------------------------------
----------\n";*/

/*if(fm>fgen) //this code section is not needed, because the Tx-away function (dawa()) will not
be used
ft=dtow(forig,c,v);
else if(fm<fgen)
ft=dawa(forig,c,v);
dft=ft-forig;*/

vdf=dfm/dft; //measured (or simulated) doppler shift in fm [dfm=fm-fgen] / calculated theoretical
doppler shift in ft [dft=ft-forig]
Serial << "dfm: " << dfm << " Hz, v: " << v << " m/s, fgen: " << fgen << " Hz, fm: " << fm << " Hz,
ft: " << ft << " Hz, vdf: " << vdf << "\n";

/*if(vdf>1) //converts vdf-value when vdf>1 (vdf is always a posit.value), [ex. if vdf=1.01 ->
(new)vdf=0.01||0.99, this code section changes the flow of full cycles]
{
Serial << "vdf_old: " << vdf << "\n";
rcircle=(int)vdf; //cycle count from double to int (how many full frequency cycles have happened
(ex. dfm=100 & dft=50 -> two cycles)
rcircled=(double)rcircle; //cycle count from int to double (converts to full cycles only, no decimal
numbers)
if(rcc!=rcircle&&rcircle>=2) //alternates the vdf-values calculation parametres of a vdf>1-state on
every second change when a full cycle happens (first cycle is not included)
rcflag=!rcflag;
if(rcflag==true)
vdf=1-(vdf-rcircled);
else
vdf=vdf-rcircled;
Serial << "rcircle: " << rcircle << ", rcircled: " << rcircled << ", vdf_new: " << vdf << "\n";
}*/

rcc=rcircle; //copies the value of rcircle to rcc, so that a full cycle change can be seen when
rcircles value changes

if(fm>fgen&&vdf<=1&&vdf>=0) //moving towards to Tx
{
Serial << "Heading to Tx.\n";
//dfm=dfm-0.000001; //angle repair test [-angles] (programmatically simulated dfm)
facos=acos(vdf)*180/pi;
```

```
Serial << "[Original angle] vdf: " << vdf << ", Acos: " << facos << " [+Tx_0]\n"; //shows the original angle in the serial monitor
if(dfm<(dfm2-0)&&vdf>=cos(45*pi/180)) //if dfm<dfm2 the Rx is going to the wrong way and needs to repair its direction, only angles that are <=45 degrees is repaired (in dfm2-0 the 0 stands for how many Hz smaller the dfm2 have to be to dfm)
{
facos=-facos*2; //ex. +44 degrees -> after direction repair -88 degrees
Serial << "[Repaired angle] vdf: " << vdf << ", Acos: " << facos << " [+Tx_1]\n"; //shows the repaired angle in the serial monitor
}
ledscr(facos); //function for LED screen
dfm=dfm+0.1; //increases the value of dfm, for testing the algorithm programmatically
if(dfm>=dft)
dfm=-0.001;
}
else if(fm<fgen&&vdf>=-1&&vdf<=1) //moving away from Tx. 1 led at the back of the led screen shows when moving away, so no need to calculate the angle of the Tx
{
Serial << "Heading away Tx.\n";
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, HIGH); //disables IC2 outputs
digitalWrite(led18Pin, HIGH); //set 180 degrees/backTx led on
digitalWrite(led17Pin, LOW); //set 0 degree led off
//facos=acos(vdf)*180/pi;
//Serial << "vdf: " << vdf << ", Acos: " << facos << " [-Tx_0]\n";
dfm=dfm-1; //decreases the value of dfm, for testing the algorithm programmatically
if(dfm<=-dft)
dfm=0.001;
}
else
{
Serial << "No doppler-frequency.\n";
ledsoff(); //turns off all leds
}
//delay(100);
Serial << "-----------------------------------------------------------------------------------------------------------------------\n";
dfm2=dfm; //copies the value of dfm to dfm2 to see if the last dfm(dfm2) is smaller than the new dfm of next calculation
}
}

double dtow(double x, double y, double z) //function for calculating theoretical value of frequency where Rx would be moving straight towards to Tx at certain speed
{
double vft=0;
vft=x*((y+z)/y);
return(vft);
}

/*double dawa(double x, double y, double z) //function for calculating theoretical value of frequency where Rx would be moving straight backwards from Tx at certain speed (this function is not necessary)
{
double vft=0;
vft=x*((y-z)/y);
return(vft);
}*/

void ledscr(double x) //fuction for controlling the led pointer display
```

```
{
if(x>-5.625&&x<5.625) //0' led17 on
{
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, HIGH); //disables IC2 outputs
digitalWrite(led17Pin, HIGH); //set 0' led on
digitalWrite(led18Pin, LOW); //set 180'/backTx led off
}
else
{
digitalWrite(led17Pin, LOW); //set 0' led off
digitalWrite(led18Pin, LOW); //set 180'/backTx led off
if(x>0&&x>=5.625) //IC2 enable
{
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, LOW); //enables IC2 outputs
}
else if(x<0&&x<=-5.625)//IC1 enable
{
digitalWrite(oePin2, HIGH); //disables IC2 outputs
digitalWrite(oePin1, LOW); //enables IC1 outputs
}
else
{
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, HIGH); //disables IC2 outputs
}
digitalWrite(latchPin, LOW); //latch pin low so led's don't change while sending bits
if(x<0)
{
if(x>=-90.000&&x<=-84.375) //LED1 -90' [leds 1-8 negat.']
shiftOut(dataPin, clockPin, MSBFIRST, 0x01);
else if(x>-84.375&&x<=-73.125) //LED2 -78.75'
shiftOut(dataPin, clockPin, MSBFIRST, 0x02);
else if(x>-73.125&&x<=-61.875) //LED3 -67.5'
shiftOut(dataPin, clockPin, MSBFIRST, 0x04);
else if(x>-61.875&&x<=-50.625) //LED4 -56.25'
shiftOut(dataPin, clockPin, MSBFIRST, 0x08);
else if(x>-50.652&&x<=-39.375) //LED5 -45'
shiftOut(dataPin, clockPin, MSBFIRST, 0x10);
else if(x>-39.375&&x<=-28.125) //LED6 -33.75'
shiftOut(dataPin, clockPin, MSBFIRST, 0x20);
else if(x>-28.125&&x<=-16.875) //LED7 -22.5'
shiftOut(dataPin, clockPin, MSBFIRST, 0x40);
else if(x>-16.875&&x<=-5.625) //LED8 -11.25'
shiftOut(dataPin, clockPin, MSBFIRST, 0x80);
}
else
{
if(x<16.875&&x>=5.625) //LED9 +11.25' [leds 9-16 posit.']
shiftOut(dataPin, clockPin, MSBFIRST, 0x01);
else if(x<28.125&&x>=16.875) //LED10 +22.5'
shiftOut(dataPin, clockPin, MSBFIRST, 0x02);
else if(x<39.375&&x>=28.125) //LED11 +33.75'
shiftOut(dataPin, clockPin, MSBFIRST, 0x04);
else if(x<50.652&&x>=39.375) //LED12 +45'
shiftOut(dataPin, clockPin, MSBFIRST, 0x08);
else if(x<61.875&&x>=50.625) //LED13 +56.25'
shiftOut(dataPin, clockPin, MSBFIRST, 0x10);
else if(x<73.125&&x>=61.875) //LED14 +67.5'
```

```
shiftOut(dataPin, clockPin, MSBFIRST, 0x20);
else if(x<84.375&&x>=73.125) //LED15 +78.75'
shiftOut(dataPin, clockPin, MSBFIRST, 0x40);
else if(x<=90.000&&x>=84.375) //LED16 +90'
shiftOut(dataPin, clockPin, MSBFIRST, 0x80);
}
digitalWrite(latchPin, HIGH); //latch pin high so leds will light up
}
}

void ledsoff() //function to turn all leds off
{
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, HIGH); //disables IC2 outputs
digitalWrite(led17Pin, LOW); //set 0' led off
digitalWrite(led18Pin, LOW); //set 180'/backTx led off
}

void counterISR() //frequency counter function
{
counter.poll();
}
```

```
/*************************************************************************************************************************************
*************************************************************************************
Project : Doppler-bearing algorithm v.1.0 [superheterodyne] (programmatically only test)
Hardware: Arduino Due [AT91SAM3X8E]
Software: Arduino software
Date    : 23.05.2014
Author  : Matti Rekis
Changes : Added code for LED screen and for averaging frequency measurements. Removed
the "moving away"-value calculation of the doppler-shift (dawa()).
Comments: Frontside leds in half circle and one led at back pointing if moving away from Tx.
-------------------------------------------------------------------------------------------------------------------------------
Using 64 bit double attributes for calculations/measurements.
Attributes  : forig = original Tx frequency (programmatically simulated) [when still; no doppler
shift]
          fgen = superheterodyne Tx frequency for reference [when still; no doppler shift]
          v = speed of the receiver [m/s] [Tx stays still, Rx moves]
          fm = measured (or simulated) superheterodyne Tx frequency when moving [doppler
shift included]
          ft = calculated theoretical frequency value where Rx would be moving straight to Tx [or
straign away from Tx <- it's not necessary to calculate the "moving away" value]
          dfm = value of the doppler shift of fm [dfm=fm-fgen;fm=fgen+dfm]
          dft = value of the theoretically calculated doppler shift of ft [dft=ft-forig]
          vdf = doppler shift in fm [dfm] / calculated theoretical doppler shift in ft [dft];
[vdf=dfm/dft]
          facos = the angle between Tx and Rx; calculated by arccos-function [facos=acos(vdf)]
          dfm2 = used to observe if the angle of Rx and Tx changes correctly when moving to-
wards to Tx [fm frequency should not decrease if moving closer to Tx]
*************************************************************************************************************************************
*************************************************************************************/
#include <Streaming.h>
#include <FreqPeriodCounter.h>

double dtow(double x, double y, double z); //functions
//double dawa(double x, double y, double z);
void ledscr(double x);
void ledsoff();

const byte counterPin = 3; // pin 3 = fgen (pin 3 is used for the measurement of the frequencies
with FreqPeriodCounter function)
const byte counterInterrupt = 1;

const byte dataPin=24, clockPin=25, latchPin=26, oePin1=27, oePin2=28; //D24=SI[DS],
D25=SCK[SH_CP], D26=RCK[ST_CP], D27=!G[!OE] for IC1, D28=!G[!OE] for IC2
const byte led17Pin=29, led18Pin=30; //D29=LED17[0 degree], D30=LED18[180 de-
grees/Txback]

FreqPeriodCounter counter(counterPin, micros, 0); //calculates the time that takes to HIGH-
LOW-period (or. vice versa) to determine T  -> f=1/T

void setup(void)
{
pinMode(dataPin, OUTPUT); //set pins to outputs to control the shift registers
pinMode(clockPin, OUTPUT);
pinMode(latchPin, OUTPUT);
pinMode(oePin1, OUTPUT); //IC1 output enable pin
pinMode(oePin2, OUTPUT); //IC2 output enable pin
pinMode(led17Pin, OUTPUT); //set pin to output for 0 degrees
pinMode(led18Pin, OUTPUT); //set pin to output for 180 degrees/Txback
Serial.begin(9600);
```

```
pinMode(9,OUTPUT); //pin 9 have to be connected to pin 3 to test with arduinos PWM-signal
analogWrite(9, 127); //pin 9 = 50% duty cycle (arduino PWM-signal in pin 9)
delay(500); //wait for stable PWM-signal
attachInterrupt(counterInterrupt, counterISR, CHANGE); //using interrupt for the frequency
measurements
ledsoff(); //turns off all leds
}

void loop(void)
{
const double pi=3.14159265359, c=299792458;
double v=2.7778, forig=2400000000, fgen=100, fm=0, ft=0, dfm=0.001, dft=0, facos=0, vdf=0,
rcircled=0, dfm2=0; //attributes for calculating and measuring doppler shift
int rcircle=0, rcc=0;
bool reset=true, rcflag=true;
byte i=0;

ft=dtow(forig,c,v); //calculates the theoretical value of doppler frequency when Rx would be
moving straight towards to Tx
dft=ft-forig; //the doppler shift of ft

//if(counter.ready()) //tests that the counter function is ready
//Serial << "counter_function: " << counter.hertz(1000) << "\n";

while(1)
{
fm=0; //formats the frequency result before new measurement/calculation

/*if(reset==true) //reset -> Tx [fgen] frequency measurement and averanging when still (without
doppler shift) (used with external frequency source)
{
while(i<20)//averages the frequency measurement [accuracy 20 frequency results]
{
if(counter.ready())
{
fgen=fgen+(double)counter.hertz(1000)/1000; //frequency accuracy in mHz .000 Hz
i++;
Serial << "Freq_value [fgen] " << i << ": " << fgen << "\n"; //shows the frequency sums in serial
monitor (while loop won't work without this command for some strange reason...)
}
else if(counter.elapsedTime > 50000) //detects if there is no frequency signal for 50ms
{
Serial << "No signal.\n";
ledsoff(); //turns off all leds
i=(byte)0;
fgen=0;
}
}
fgen=fgen/(double)i;
i=(byte)0;
reset=false;
Serial << "-----------------------------------------------------------------------------------------------------------
----------\n";
}*/

fm=fgen+dfm; //this code is used when dfm value is set programmatically

/*while(i<10) //frequency measurements and averaging when moving [accuracy 10 frequency
results] (this code section is used with external frequency source)
{
```

```
if(counter.ready())
{
fm=fm+(double)counter.hertz(1000)/1000; //frequency accuracy in mHz .000 Hz
i++;
Serial << "Freq_value [fm] " << i << ": " << fm << "\n"; //shows the frequency sums in Serial
Monitor (while loop won't work without this command for some strange reason...)
}
else if(counter.elapsedTime > 50000) //detects if there is no frequency signal for 50ms
{
Serial << "No signal.\n";
ledsoff(); //turns off all leds
i=(byte)0;
fm=0;
}
}
fm=fm/(double)i;
dfm=fm-fgen;
i=(byte)0;
Serial << "----------------------------------------------------------------------------------------------------------
----------\n";*/

/*if(fm>fgen) //this code section is not needed, because the Tx-away function (dawa()) will not
be used
ft=dtow(forig,c,v);
else if(fm<fgen)
ft=dawa(forig,c,v);
dft=ft-forig;*/

vdf=dfm/dft; //measured (or simulated) doppler shift in fm [dfm=fm-fgen] / calculated theoretical
doppler shift in ft [dft=ft-forig]
Serial << "dfm: " << dfm << " Hz, v: " << v << " m/s, fgen: " << fgen << " Hz, fm: " << fm << " Hz,
ft: " << ft << " Hz, vdf: " << vdf << "\n";

if(vdf>1) //converts vdf-value when vdf>1 (vdf is always a posit.value), [ex. if vdf=1.01 ->
(new)vdf=0.01||0.99, this code section changes the flow of full cycles]
{
Serial << "vdf_old: " << vdf << "\n";
rcircle=(int)vdf; //cycle count from double to int (how many full frequency cycles have happened
(ex. dfm=100 & dft=50 -> two cycles)
rcircled=(double)rcircle; //cycle count from int to double (converts to full cycles only, no decimal
numbers)
if(rcc!=rcircle&&rcircle>=2) //alternates the vdf-values calculation parametres of a vdf>1-state on
every second change when a full cycle happens (first cycle is not included)
rcflag=!rcflag;
if(rcflag==true)
vdf=1-(vdf-rcircled);
else
vdf=vdf-rcircled;
Serial << "rcircle: " << rcircle << ", rcircled: " << rcircled << ", vdf_new: " << vdf << "\n";
}

rcc=rcircle; //copies the value of rcircle to rcc, so that a full cycle change can be seen when
rcircles value changes

if(fm>fgen) //moving towards to Tx
{
Serial << "Heading to Tx.\n";
dfm=dfm-0.000001; //angle repair test [-angles] (programmatically simulated dfm)
facos=acos(vdf)*180/pi;
```

```
Serial << "[Original angle] vdf: " << vdf << ", Acos: " << facos << " [+Tx_0]\n"; //shows the original angle in the serial monitor
if(dfm<(dfm2-0)&&vdf>=cos(45*pi/180)) //if dfm<dfm2 the Rx is going to the wrong way and needs to repair its direction, only angles that are <=45 degrees is repaired (in dfm2-0 the 0 stands for how many Hz smaller the dfm2 have to be to dfm)
{
facos=-facos*2; //ex. +44 degrees -> after direction repair -88 degrees
Serial << "[Repaired angle] vdf: " << vdf << ", Acos: " << facos << " [+Tx_1]\n"; //shows the repaired angle in the serial monitor
}
ledscr(facos); //function for LED screen
dfm=dfm+0.05; //increases the value of dfm, for testing the algorithm programmatically
if(dfm>=(dft*4))
dfm=-0.001;
}
else if(fm<fgen) //moving away from Tx. 1 led at the back of the led screen shows when moving away, so no need to calculate the angle of the Tx
{
Serial << "Heading away Tx.\n";
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, HIGH); //disables IC2 outputs
digitalWrite(led18Pin, HIGH); //set 180 degrees/backTx led on
digitalWrite(led17Pin, LOW); //set 0 degree led off
//facos=acos(vdf)*180/pi;
//Serial << "vdf: " << vdf << ", Acos: " << facos << " [-Tx_0]\n";
dfm=dfm-1; //decreases the value of dfm, for testing the algorithm programmatically
if(dfm<=(dft*-2))
dfm=0.001;
}
else
{
Serial << "No doppler-frequency.\n";
ledsoff(); //turns off all leds
}
//delay(100);
Serial << "-------------------------------------------------------------------------------------------------------------------------\n";
dfm2=dfm; //copies the value of dfm to dfm2 to see if the last dfm(dfm2) is smaller than the new dfm of next calculation
}
}

double dtow(double x, double y, double z) //function for calculating theoretical value of frequency where Rx would be moving straight towards to Tx at certain speed
{
double vft=0;
vft=x*((y+z)/y);
return(vft);
}

/*double dawa(double x, double y, double z) //function for calculating theoretical value of frequency where Rx would be moving straight backwards from Tx at certain speed (this function is not necessary)
{
double vft=0;
vft=x*((y-z)/y);
return(vft);
}*/

void ledscr(double x) //fuction for controlling the led pointer display
```

```
{
if(x>-5.625&&x<5.625) //0' led17 on
{
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, HIGH); //disables IC2 outputs
digitalWrite(led17Pin, HIGH); //set 0' led on
digitalWrite(led18Pin, LOW); //set 180'/backTx led off
}
else
{
digitalWrite(led17Pin, LOW); //set 0' led off
digitalWrite(led18Pin, LOW); //set 180'/backTx led off
if(x>0&&x>=5.625) //IC2 enable
{
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, LOW); //enables IC2 outputs
}
else if(x<0&&x<=-5.625)//IC1 enable
{
digitalWrite(oePin2, HIGH); //disables IC2 outputs
digitalWrite(oePin1, LOW); //enables IC1 outputs
}
else
{
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, HIGH); //disables IC2 outputs
}
digitalWrite(latchPin, LOW); //latch pin low so led's don't change while sending bits
if(x<0)
{
if(x>=-90.000&&x<=-84.375) //LED1 -90' [leds 1-8 negat.']
shiftOut(dataPin, clockPin, MSBFIRST, 0x01);
else if(x>-84.375&&x<=-73.125) //LED2 -78.75'
shiftOut(dataPin, clockPin, MSBFIRST, 0x02);
else if(x>-73.125&&x<=-61.875) //LED3 -67.5'
shiftOut(dataPin, clockPin, MSBFIRST, 0x04);
else if(x>-61.875&&x<=-50.625) //LED4 -56.25'
shiftOut(dataPin, clockPin, MSBFIRST, 0x08);
else if(x>-50.652&&x<=-39.375) //LED5 -45'
shiftOut(dataPin, clockPin, MSBFIRST, 0x10);
else if(x>-39.375&&x<=-28.125) //LED6 -33.75'
shiftOut(dataPin, clockPin, MSBFIRST, 0x20);
else if(x>-28.125&&x<=-16.875) //LED7 -22.5'
shiftOut(dataPin, clockPin, MSBFIRST, 0x40);
else if(x>-16.875&&x<=-5.625) //LED8 -11.25'
shiftOut(dataPin, clockPin, MSBFIRST, 0x80);
}
else
{
if(x<16.875&&x>=5.625) //LED9 +11.25' [leds 9-16 posit.']
shiftOut(dataPin, clockPin, MSBFIRST, 0x01);
else if(x<28.125&&x>=16.875) //LED10 +22.5'
shiftOut(dataPin, clockPin, MSBFIRST, 0x02);
else if(x<39.375&&x>=28.125) //LED11 +33.75'
shiftOut(dataPin, clockPin, MSBFIRST, 0x04);
else if(x<50.652&&x>=39.375) //LED12 +45'
shiftOut(dataPin, clockPin, MSBFIRST, 0x08);
else if(x<61.875&&x>=50.625) //LED13 +56.25'
shiftOut(dataPin, clockPin, MSBFIRST, 0x10);
else if(x<73.125&&x>=61.875) //LED14 +67.5'
```

```
shiftOut(dataPin, clockPin, MSBFIRST, 0x20);
else if(x<84.375&&x>=73.125) //LED15 +78.75'
shiftOut(dataPin, clockPin, MSBFIRST, 0x40);
else if(x<=90.000&&x>=84.375) //LED16 +90'
shiftOut(dataPin, clockPin, MSBFIRST, 0x80);
}
digitalWrite(latchPin, HIGH); //latch pin high so leds will light up
}
}

void ledsoff() //function to turn all leds off
{
digitalWrite(oePin1, HIGH); //disables IC1 outputs
digitalWrite(oePin2, HIGH); //disables IC2 outputs
digitalWrite(led17Pin, LOW); //set 0' led off
digitalWrite(led18Pin, LOW); //set 180'/backTx led off
}

void counterISR() //frequency counter function
{
counter.poll();
}
```