Pro gradu -tutkielma
Maantiede
Geoinformatiikka

SYVYYSMALLIPINTOJEN MUOKKAUS AUTOMAATTISESSA SYVYYSKÄYRÄNMÄÄRITYKSESSÄ: TARKASTELUSSA ROLLING COIN JA LAPLACE-INTERPOLOINTI

Topi Filppula

2018

Ohjaaja: Tuuli Toivonen

HELSINGIN YLIOPISTO
MATEMAATTIS-LUONNONTIETEELLINEN TIEDEKUNTA
GEOTIETEIDEN JA MAANTIETEEN LAITOS
MAANTIEDE

PL 64 (Gustaf Hällströmin katu 2)
00014 Helsingin yliopisto

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Tiedekunta/Osasto Fakultet/Sektion – Faculty	Laitos/Institution– Department
Matemaattis-luonnontieteellinen tiedekunta	Geotieteiden ja maantieteen laitos

Tekijä/Författare – Author
Topi Matias Filppula

Työn nimi / Arbetets titel – Title
Syvyysmallipintojen muokkaus automaattisessa syvyyskäyränmäärityksessä: tarkastelussa Rolling Coin ja Laplace-interpolointi

Oppiaine /Läroämne – Subject
Maantiede

Työn laji/Arbetets art – Level	Aika/Datum – Month and year	Sivumäärä/ Sidoantal – Number of pages
Pro Gradu -tutkielma	6/2018	117

Tiivistelmä/Referat – Abstract

Merikarttojen syvyystietojen tuottaminen perustuu merenmittausteknologiassa viime vuosikymmeninä tapahtuneesta merkittävästä kehittymisestä huolimatta edelleen pääosin perinteisiin manuaalisiin menetelmiin. Vielä nykyäänkin merikartoilla esitettävät syvyystiedot tuotetaan määrittämällä syvyyskäyrät käsin ja valitsemalla kartoilla esitettävät syvyyspisteet tiheästä pisteaineistosta yksitellen. Vähenevien henkilöresurssien johdosta merikartoilla esitettävän syvyystiedon tuottamisesta on muodostunut Liikenneviraston merikartantuotantoon pullonkaula.

Tässä työssä tutkittiin pistepilviaineistoista luotujen syvyysmallien käsittelyä automatisoidun syvyyskäyräntuotannon mahdollistamiseksi. Tutkittujen syvyysmallipintojen käsittelyn menetelmien, navigointiturvalliseksi muokatun Laplace-interpoloinnin ja erityisesti tätä työtä varten kehitetyn Rolling Coin -menetelmän, toivottiin helpottavan teknisesti ja kartografisesti laadukkaiden syvyyskäyrien määrityksen automatisointia. Tutkituilla menetelmillä tuotettujen syvyysmallipintojen ominaisuuksia ja keskinäisiä eroja tutkittiin erotuspintojen, niistä johdettujen poikkeamamittareiden ja erilaisten naapurustoon perustuvien, syvyysmallipintojen paikallista syvyysvaihtelua kuvaavien indeksien avulla. Menetelmien soveltuvuutta syvyyskäyrien automaattiseen tuotantoon arvioitiin luomalla tuotetuista syvyysmallipinnoista automaattisesti määritetyt syvyyskäyrät ja vertaamalla saatuja syvyyskäyriä merikartoilla esitettyihin syvyyskäyriin. Syvyyskäyrien vertailussa keskityttiin syvyyskäyrien lukumäärien ja pituuksien arviointiin.

Molemmat tutkitut syvyysmallipintojen käsittelyn menetelmät osoittautuivat navigoinnin kannalta turvallisiksi ja automatisoidun syvyyskäyränmäärityksen näkökulmasta lupaaviksi. Rolling Coin -menetelmä osoittautui erityisen kiinnostavaksi ja toimivaksi myös hankalilla kapeilla ja ruopatuilla väyläosuuksilla. Liikennevirasto onkin ottamassa Rolling Coin -menetelmän tuotantokäyttöön. Liikenneviraston uuden merikarttojen tuotantojärjestelmän toimittava CARIS sisällyttää menetelmän myös osaksi tarjoamansa sovellusten perustoiminnallisuuksia.

Avainsanat – Nyckelord – Keywords
Syvyysmalli, syvyyskäyrä, käyränmääritys, Rolling Coin, Laplace-interpolointi

Säilytyspaikka – Förvaringställe – Where deposited
HELDA – Helsingin yliopiston digitaalinen arkisto

Muita tietoja – Övriga uppgifter – Additional information

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Tiedekunta/Osasto Fakultet/Sektion – Faculty	Laitos/Institution– Department
Faculty of Science	Department of Geosciences and Geography

Tekijä/Författare – Author
Topi Matias Filppula

Työn nimi / Arbetets titel – Title
Syvyysmallipintojen muokkaus automaattisessa syvyyskäyränmäärityksessä: tarkastelussa Rolling Coin ja Laplace-interpolointi

Oppiaine /Läroämne – Subject
Geography

Työn laji/Arbetets art – Level	Aika/Datum – Month and year	Sivumäärä/ Sidoantal – Number of pages
Master's Thesis	6/2018	117

Tiivistelmä/Referat – Abstract

The production of depth information of navigation charts is, despite significant developments in bathymetric surveying technology in the last decades, still mostly based on traditional manual methods. Even today the depth contours are generally digitized by hand and the chart soundings are individually selected from the dense bathymetric point data. With decreasing human resources these labor intensive methods have created a bottleneck in the Finnish Transport Agency's navigation chart production.

The main goal of this study was to examine two different depth model surface smoothing methods in order to enable the automated production of quality depth contours. The methods used to smooth the depth model surfaces were navigationally safe Laplace-interpolation and Rolling Coin, a method developed specifically for this study. The properties and differences of the smoothed depth model surfaces were examined using surface differences (separation surfaces), deviation metrics and different neighborhood based indices describing local variations in the depth model surface. The suitability of the smoothing methods for automated depth contour generation was evaluated by creating automatically generated depth contours from the smoothed depth model surfaces and comparing these contours to the contours on the current navigation chart products. The comparison of the contours focused on evaluation of the number and lengths of the contours. Cartographic evaluation of the contours was not included in this thesis.

Both depth model surface smoothing methods proved to be navigationally safe and promising in terms of automation of depth contour generation. The Rolling Coin method proved to be especially interesting and functional even in the generally troublesome narrow and dredged channels. The Finnish Hydrographic Office is going to take the Rolling Coin method for production use.

Avainsanat – Nyckelord – Keywords
Depth model, depth contour, contour creation, Rolling Coin, Laplace-interpolation

Säilytyspaikka – Förvaringställe – Where deposited
HELDA - Digital Repository of the University of Helsinki

Muita tietoja – Övriga uppgifter – Additional information

TERMIT JA LYHENTEET

ALIPÄÄSTÖSUODIN	Suodin, joka päästää läpi matalataajuiset signaalit, mutta suodattaa korkeataajuiset signaalit.
BAG	*Bathymetry Attributed Grid*. *Open Navigation Surface Working Group:n* määrittelemä avoin, HDF-5 -formaattiin perustuva kaksikanavainen rasterimuotoinen syvyysmalliformaatti. Toimii virallisen, navigointikäyttöön tuotettavan IHO S-102 syvyysmallistandardin pohjana.
BUFFERING	*Puskurointi, puhekielessä myös bufferointi.* Läheisyysanalyysissa käytettävä paikkatietoanalyysimenetelmä, jonka tuloksena on alue, jonka reuna ulottuu määrätylle etäisyydelle tarkasteltavasta kohteesta tai kohdejoukosta (Sanastokeskus TSK 2018).
GNSS	*Global Navigation Satellite System.* Globaali satelliittipaikannusjärjestelmä.
HARAUSTASO	Jonkin tietyn alueen tasomainen, varmistettu vesisyvyys. Voidaan määrittää esimerkiksi mekaanisella tankoharalla tai koko kohdealueen pohjan peittävän monikeilainmittauksen tuottamasta pistepilvestä.
IHO	*International Hydrographic Organization*, kansainvälinen merikartoitusjärjestö. Vastaa myös navigoinnissa käytettävien tietotuotteiden standardeista.
IMO	*International Maritime Organization*, kansainvälinen merenkulkujärjestö. Säätelee kansainvälistä merenkulkua tavoitteena merenkulun turvallisuuden parantaminen ja merenkulusta ympäristölle aiheutuvan kuormituksen pienentäminen.
KERNEL	*Painokerroinmatriisi (Convolution Kernel).* Rasteriaineistojen konvoluutioissa käytettävä naapuruston painokertoimet määrittävä matriisi.

MBES	*Multi-Beam Echo Sounder.* Monikeilain eli monikanavainen kaikuluotain. Mittaa syvyydet viuhkamaisesti mittausveneen alapuolelta.
NO DATA	Rasterimuotoisten aineistojen erityisesti varattu soluarvo, jonka tarkoitus on kuvata puuttuvaa tietoa. *No Data* -arvon saaneiden solujen katsotaan olevan tyhjiä, vaikka niillä todellisuudessa onkin tarkoitukseen erityisesti valittu arvo.
RTK	*Real-Time Kinematic.* Reaaliaikainen kinemaattinen satelliittipaikannusmenetelmä, jossa paikannuksen virheet on korjattu mahdollisimman hyvin. Nykyaikaisista reaaliaikaisista satelliittipaikannusmenetelmistä tarkin.
SBES	*Single Beam Echo Sounder.* (Yksikanavainen) kaikuluotain.
SHOAL BIAS	*Matalia suosiva*. Merikartoituksessa ja merenmittaustietojen käsittelyssä yleinen käytäntö, jonka mukaan esitettäväksi tai tuotteelle tallennettavaksi valitaan matalin syvyyshavainto. Syvyysmallien tapauksessa tämä on sama kuin maksimimalli.
SOLAS	*Safety Of Life At Sea*. Kansainvälinen merenkulkua koskeva sopimus, joka määrittelee merenkulun kansainvälisiä käytäntöjä ja vaatimuksia.
TIN	*Triangulated Irregular Network.* Kolmioverkko, joka muodostuu epäsäännöllisesti jakautuneita pisteitä yhdistävistä janoista.

SISÄLLYSLUETTELO

1. JOHDANTO 5

2. TAUSTA 8
2.1. Navigointiturvallisten syvyysmallipintojen tuottaminen 12
2.2. Syvyysmallipintojen käsittely 13
2.2.1. Syvyysmallipintojen yleistäminen eli spatiaalisen resoluution karkeistaminen 14
2.2.2. Syvyysmallipintojen pehmentäminen 15
2.2.3. Fokaali- eli naapurusto-operaatiot ja konvoluutio 19
2.3. Navigointiturvallinen iteratiivinen Laplace-interpolointi 21
2.4. Rolling Coin 29
2.4.1. Alkuperäinen, syvyyskäyrärajat tuottava Rolling Coin 30
2.4.2. Syvyysmallipintoja muokkaava 2,5-ulotteinen Rolling Coin 32

3. TUTKIMUSALUEET 34
3.1. Pietarsaaren meriväylä 35
3.2. Uudenkaupungin meriväylä 36
3.3. Sköldvikin meriväylä 37

4. AINEISTO 39

5. MENETELMÄT 40
5.1. Syvyyskäyrien automaattinen määritys (syvyysmallipintojen vektorointi) 41
5.2. Erotuspinnat 41
5.3. Root Mean Square Difference, RMSD 42
5.4. Bathymetric Position Index, BPI 42
5.5. Terrain Ruggedness Index, TRI 43
5.6. Roughness Index 44
5.7. Syvyyskäyrien lukumäärä ja käyrien pituuksien jakauma 44

6. TULOKSET 45
6.1. Tutkimusalueiden syvyydet 45
6.2. Erotuspinnat 46
6.3. Root Mean Square Difference, RMSD 47
6.4. Bathymetric Position Index, BPI 49
6.5. Terrain Ruggedness Index, TRI 53
6.6. Roughness 56
6.7. Syvyyskäyrät 59
6.7.1. Syvyyskäyrien lukumäärä 59
6.7.2. Syvyyskäyrien pituudet 62

7. KESKUSTELU 65
7.1. Syvyysmallipintojen pehmennysmenetelmien turvallisuus 66
7.2. Syvyysmallipintojen keskinäiset erot 67
7.3. Automaattisesti määritetyt syvyyskäyrät 71
7.4. Tulosten kokonaistarkastelu 75

8. JOHTOPÄÄTÖKSET 80

9. KIITOKSET 82

10. KIRJALLISUUS 83

LIITE 1

1. JOHDANTO

Ajantasainen, virallinen merikartta on merenkululle paitsi korvaamattoman arvokas tietolähde, myös kansainvälisten sopimusten nojalla (IMO 1974) kauppamerenkulun aluksille pakollinen varuste. Itämeren laivaliikenne on kasvanut viime vuosina voimakkaasti eikä onnettomuuksiltakaan olla vältytty. Kujala et al. (2009) mukaan yleisin Suomenlahden meriliikenteessä tapahtuva onnettomuus on karilleajo. Karille ajaneista aluksista likimain yksi kymmenestä on öljytankkeri. Merenkulun turvallisuus onkin tänä päivänä erittäin ajankohtainen aihe niin inhimilliseltä, taloudelliselta kuin ympäristönkin kannalta.

Merikartta on merenkulkijan ainoa virallinen syvyystietoja tarjoava lähde. Vanhimmat merikarttojen syvyystietojen hankinnan menetelmät olivat mekaanisia luotausmenetelmiä, kuten punttiluotaus, jossa vesialueen syvyyksiä mitattiin naruun kiinnitetyn painon avulla (IHO 2005). Punttiluotausmenetelmään sisältyi huomattavia epävarmuuksia, sillä koko merenpohjaa ei voitu mitata ja mittauspisteiden väliin jäi kartoittamattomia matalikkoja, karikkoja ja lohkareita. Lisäksi aikakaudelle ominaiset, perinteiset optiset paikannusmenetelmät aiheuttivat mitattuihin pisteisiin nykymittapuulla arvioituna suuria sijaintiepävarmuuksia. Vanhanaikaisten merenmittausmenetelmien tuottama syvyysaineisto oli luonteeltaan harvaa pisteaineistoa, jonka pohjalta merikartoilla esitettävät syvyyspisteet valittiin yksitellen tiettyjen valintasääntöjen mukaisesti (Liikennevirasto 2017a). Syvyyskäyrät merikartoille tuotettiin mittauspisteiden perusteella käsin piirtämällä eli interpoloimalla (Peters et al. 2014).

Teknologian kehittyessä mekaaninen punttiluotaus korvautui 1950-luvulta alkaen ensin kaikuluotauksella *(SBES, Single Beam Echo Sounder)*, jolla saatiin mitattua mittausveneen ajolinjojen mukaisia syvyysprofiileja. Kuitenkin vasta nykyaikaisten monikeilaimien *(MBES, Multi-Beam Echo Sounder)* ja syvyysmittauksiin soveltuvien laserkeilaimien myötä on voitu saavuttaa koko mitattavan alueen pohjan kattava mittaustulos (IHO 2005). Nykyään myös merenmittausaineistoihin liittyvät sijaintiepävarmuudet ovat hyvin pieniä modernien differentiaali- ja reaaliaikaisten kinemaattisten (RTK) GNSS -paikannusmenetelmien käytön myötä.

Vaikka sekä merenmittaus- että siihen kiinteästi liittyvä paikannusteknologia ovat kehittyneet huomattavasti merenmittauksen alkuajoista, ovat mittausaineiston käsittelyn menetelmät pysyneet hyvin samankaltaisina kuin punttiluotauksen aikoina (Calder ja Mayer 2003; Peters et al. 2014). Myös Suomessa merikarttojen syvyystietoja tuotetaan yhä edelleen tiheästä merenmittausaineistosta digitoimalla syvyyskäyrät käsin ja valitsemalla tiheästä pistemassasta merikarttatuotteilla esitettävät syvyyspisteet manuaalisesti yksi kerrallaan (Liikennevirasto 2017a). Merikartoilla esitettävien, yleistettyjen syvyystietojen tuottamisesta onkin muodostunut merikarttatuotantoon pullonkaula, jonka poistaminen parantaisi merenkulkijalle merikartoilla välitettävien syvyystietojen ajantasaisuutta ja turvallisuutta.

Nykyisten merikartoilla esitettävien syvyystietojen tuottamisen työvoimaintensiivisyys, korkeat kustannukset ja käytettyjen menetelmien hitaus yhdessä modernien merenmittausaineistojen kattavuuden jatkuvan laajenemisen kanssa tuovat mukanaan vaatimuksia käytettyjen menetelmien ja prosessien tehostamiseksi ja automatisoinniksi. Lisäksi uusien, navigointikäyttöön suunnattujen syvyysmallituotteiden (IHO 2012) tuotannon aloittaminen aiheuttaa uusia vaatimuksia navigointituotteiden keskinäiselle ristiriidattomuudelle.

Syvyysaineistojen tuotannon tehostamiseksi ja osittaisen automatisoinnin edistämiseksi on Liikenneviraston merikartoituspalvelut -yksikössä tutkittu ja kehitetty syvyysmallien käyttöä perinteisten syvyystietojen tuottamisen lähtöaineistoina (Heiskanen 2008; Liikennevirasto 2017b). Syvyysmallipintojen käsittelyyn on näissä yhteyksissä sovellettu eri menetelmiä, joiden on toivottu vastaavan syvyyskäyräntuotannon teknisiin ja kartografisiin laatuvaatimuksiin, kuten automaattisesti tuotettujen syvyyskäyrien navigointiturvallisuuteen ja eri karttatuotteiden mittakaavatasoille sopiviin yleistystasoihin.

Tämän työn tavoitteena on:

1. tutkia kahden syvyysmallipintojen pehmennysmenetelmän turvallisuutta,
2. selvittää miten näillä pehmennysmenetelmillä tuotetut syvyysmallipinnat eroavat toisistaan ja alkuperäisistä syvyysmallipinnoista ja
3. selvittää vaikuttavatko mainitut menetelmät käyttökelpoisilta automatisoidun syvyyskäyrien tuotannon näkökulmasta.

Näihin tutkimuskysymyksiin pyritään vastaamaan tuottamalla pehmennetyt syvyysmallipinnat tätä työtä varten kirjoitetulla syvyysmallipintojen muokkaukseen soveltuvalla sovelluksella (liite 1). Syvyysmallipintojen muokkaukseen käytettyjen menetelmien navigointiturvallisuus selvitetään erotuspintojen avulla. Erotuspinnat lasketaan vähentämällä muokatuista syvyysmallipinnoista alkuperäiset, käsittelemättömät syvyysmallipinnat. Erotuspinnat ja siten käytettyjen syvyysmallipintojen muokkausmenetelmien navigointiturvallisuus ovat tämän työn tärkein tutkimustulos. Syvyysmallipintojen keskinäisiä eroavaisuuksia selvitetään erilaisilla naapurustoon perustuvilla, pintojen tasaisuutta kuvaavilla indekseillä. Muokattujen syvyysmallipintojen keskinäisiä eroavaisuuksia tutkitaan lisäksi erotuspintojen ja niistä johdettujen RMSD-arvojen avulla.

Menetelmien käyttökelpoisuutta automatisoidun syvyyskäyrien tuotannon näkökulmasta arvioidaan vertaamalla syvyysmallipinnoista automaattisesti luotujen syvyyskäyrien lukumäärää ja pituuksia toisiinsa ja merikartoituksen nykyisen tuotantotietokannan sisältämiin syvyyskäyriin. Tässä työssä arvioidaan ainoastaan syvyyskäyrien teknisiä ominaisuuksia. Syvyyskäyrien laatuun ja käyttökelpoisuuteen sinällään olennaisesti liittyvä kartografinen tulkinta on jätetty tämän työn rajauksen ulkopuolelle.

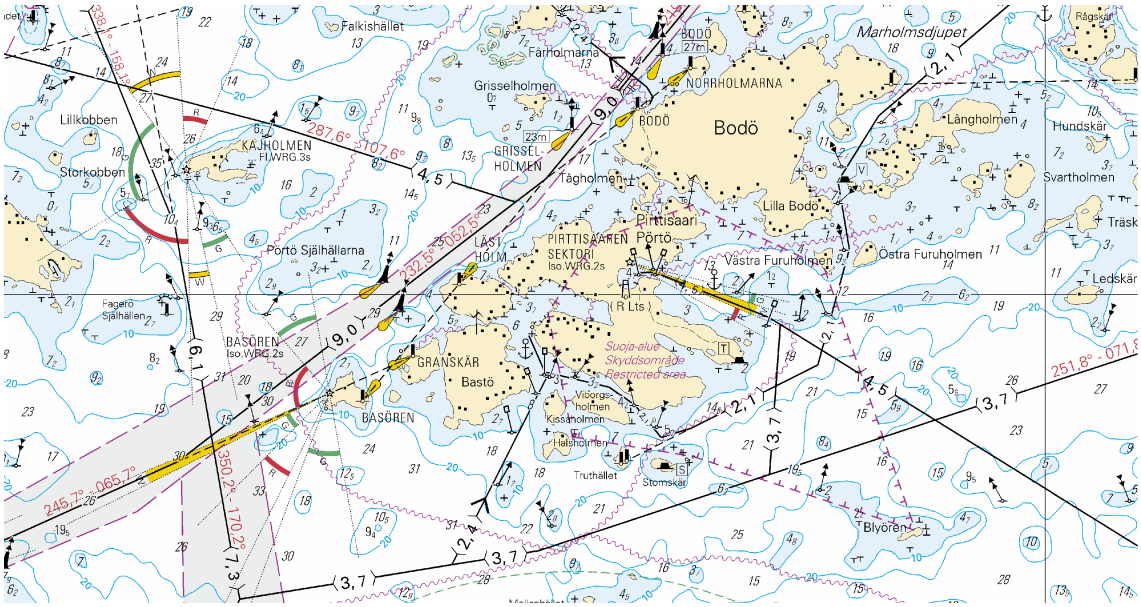
Näiden, työn sisällöllisten, tavoitteiden lisäksi tavoitteena on tuottaa sovellus, joka sisältää syvyysmallipintojen pehmennyksessä käytettyjen menetelmien implementaatiot. Syvyysmallipintojen käsittely on usein aineistojen laajuuden vuoksi laskennallisesti vaativaa ja tämä kannattaa huomioida algoritmisuunnittelun lisäksi myös ohjelmointikielen valinnassa. C -ohjelmointikieli (Kernighan ja Ritchie 1988) onkin valittu tuotettavan sovelluksen kieleksi sen korkean suorituskyvyn ja yleisen tunnettuuden vuoksi. Tuotetun sovelluksen lähdekoodi sisällytetään osaksi tätä työtä liitteenä (liite 1). Edelleen yhtenä työn yleisenä tavoitteena on mahdollisimman suuri avoimien ohjelmistojen ja ohjelmakirjastojen käyttö. Tämän työn tuloksien tilastolliseen tarkasteluun ja visualisointiin on käytetty avointa R -ohjelmointikieltä (R Core Team 2017), lukuun ottamatta kuvassa 3 nähtävää syvyysmallipintojen 3D-visualisointia, joka on tehty Fledermaus -sovelluksella (QPS 2010).

2. TAUSTA

Vuonna 1912 tapahtunut RMS Titanicin uppoaminen sai osakseen valtavaa huomiota ja vaikutti osaltaan merenkulun nykyisin laajasti tunnustettujen kansainvälisten sopimusten syntymiseen. Merenkulun turvallisuuteen keskittyvä SOLAS *(Safety of Life at Sea)* -sopimus (IMO 1974) määrittelee ne alustyypit, joita sopimuksen velvoitteet mm. pakollisesta ajantasaisten merikarttojen kantovaatimuksesta koskevat. Kunkin sopimuksen ratifioineen valtion kansallinen lainsäädäntö puolestaan huolehtii näiden velvoitteiden käytännön toteutumisesta.

Merikartoitustoiminta on luonteeltaan viranomaistoimintaa, jossa kukin kansainvälisen merikartoitusjärjestö IHO:n (IHO 2018) jäsenvaltio huolehtii omien merialueidensa kartoituksen järjestämisestä. Karttojen laatimisen ja ylläpidon järjestämisen ohella kukin valtio vastaa myös merikarttojensa tietojen paikkansapitävyydestä. Suomessa merikartoituksesta vastaavana viranomaisena toimii liikenne- ja viestintäministeriön alainen Liikennevirasto (Laki Liikennevirastosta 862/2009 § 2, Liikennevirasto 2018a). Liikennevirasto tuottaa merikarttatuotteita rannikkoalueiden lisäksi myös tärkeimmiltä sisävesiltä.

Ajantasaisesta merikarttatuotteesta merenkulkija löytää vedenalaista topografiaa ja paikallista vesisyvyyttä kuvaavien syvyyskäyrien, syvyysalueiden ja pistemäisten syvyyslukujen lisäksi myös muut navigointiin olennaisesti liittyvät kohteet, kuten väylien turvalaitteet ja erilaiset maamerkit. Nykyiset viralliset, kauppamerenkulun käyttämät navigointituotteet kattavat kuvassa 1 nähtävän rannikkokartan (Liikennevirasto 2017) kaltaisten perinteisten painettujen merikarttojen (IHO 2017a) lisäksi elektronisessa navigointiympäristössä tarkoitukseen erityisesti kehitetyillä ja hyväksytyillä ECDIS *(Electronic Chart Display and Information System)* -laitteilla (IHO 2014) käytettävät vektorimuotoiset merikartat, niin kutsutut ENC-karttasolut (IHO 2000).



Kuva 1. Merikartan tietosisältö on valtava. Kuvassa on osa 1:50000 mittakaavan rannikkokarttaa numero 17 Suomenlahdelta. Aineisto © Liikennevirasto 2018.

Yksi merikarttojen tärkeimmistä tehtävistä on antaa kartan käyttäjälle ajantasainen, selkeä ja todenmukainen kuva navigoitavan vesialueen syvyyksistä. Nykyisillä merikarttatuotteilla syvyys- ja pohjatopografiatietoja viestitään pääasiassa kolmella eri keinolla:

1. syvyyskäyrinä,
2. syvyyskäyristä muodostetuilla syvyysalueilla ja
3. erikseen kartoilla esitettäväksi valituilla syvyyspisteillä.

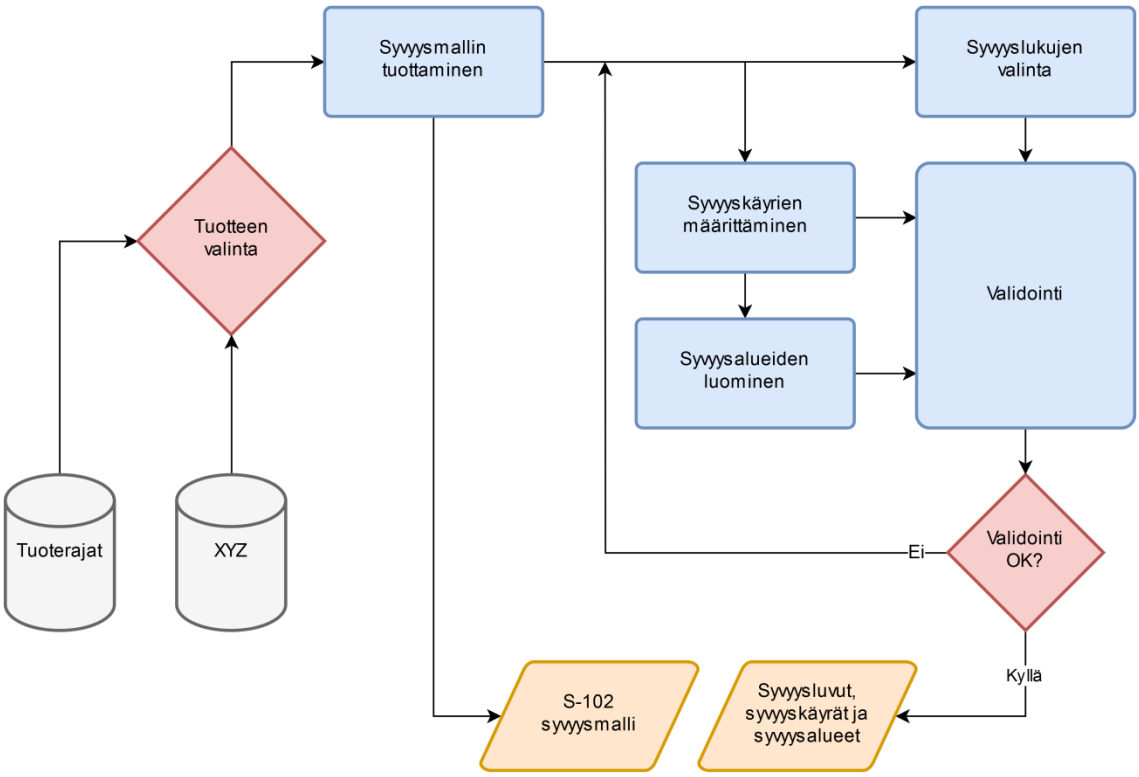
Näiden lisäksi syvyystietoja voidaan välittää esimerkiksi väylien nimellisten kulkusyvyyksien ja haraustasojen kaltaisilla aluemaisilla syvyystiedoilla sekä muilla pistemäisillä tiedoilla, kuten merenkulun esteillä ja hylkyjen vähimmäissyvyyksillä (Liikennevirasto 2010).

Merenkulun turvallisuuden takaamiseksi kaiken merikarttatuotteilla esitettäväksi valittavan syvyystiedon tulee olla ajantasaista, mahdollisimman tarkkaa ja navigoinnin kannalta turvallisesti tuotettua. Nämä vaatimukset näkyvät sekä merenmittausten teknisissä vaatimuksissa, että mittausaineistoista merikartoille yleistettyjen karttatietojen tuottamisessa. Merenmittausten tuottamasta raakadatasta merikarttatuotteilla esitettäviksi yleistettyjen tietojen navigointiturvallisuus tarkoittaa käytännössä sitä, että missään ei saa esiintyä kartalla kyseiselle kohdalle ilmoitettua syvyyttä matalampia kiinteitä kohteita.

Merikartoilla esitettävien syvyystietojen tulee perustua merenmittausten tekniset laatuvaatimukset ja mittausaineistojen laadun luokitukset määrittelevään IHO:n S-44 standardin (IHO 2008) mukaan tehtyihin merenmittauksiin. Standardin etenkin sen korkeimmille laatuluokille *(Special Order, Order 1a)* asettamat tekniset laatuvaatimukset ovat tiukkoja. Käytännössä kaikki standardin mukaiset, nykyaikaiset merenmittaukset suoritetaan nk. monikeilainmenetelmällä *(Multi-Beam Echo Sounder, MBES)*. Mittauksissa käytettävä monikeilain on käytännössä monikanavainen kaikuluotain, jonka tuottama merenmittausaineisto on tyypiltään tiheää pistepilveä.

Merenmittaus- ja paikkatietoteknologian viimeaikaisen nopean kehittymisen myötä myös kokonaan uudentyyppisten navigointituotteiden tuottaminen on tullut mahdolliseksi. Nykyisin käytössä olevien merikarttatuotteiden rinnalle onkin tulossa myös tasaväliseen ruudukkoon perustuva *grid*- eli rasterimuotoinen syvyysmallituote, joka on määritelty IHO:n standardissa S-102 (IHO 2012). Standardin määrittelemä syvyysmallituote perustuu kaksikanavaiseen tasaväliseen ruudukkoon, jonka jokaiseen soluun on tallennettu tieto solun syvyydestä ja syvyyden epävarmuudesta. Yhdessä muiden seuraavan, S-100 -sukupolven IHO-standardien (IHO 2013, 2017b) mukaisten tuotteiden kanssa käytettynä S-102 -syvyysmallituotteet mahdollistavat reaaliaikaisiin olosuhdetietoihin perustuvien syvyystietojen esittämisen ja turvallisten alueiden rajojen määrittämisen aluskohtaisesti sen hetkinen lastitilanne ja olosuhteet huomioiden.

Merikarttatuotteilla esitettävien yleistettyjen syvyystietojen tuottamisen tehostamiseksi, syvyystietojen laadun tasaistamiseksi sekä erityyppisten navigointituotteiden keskinäisen ristiriidattomuuden varmistamiseksi Liikenneviraston merikartoituspalvelut suunnittelee siirtyvänsä uuteen tuotantoprosessimalliin, jossa merikarttatuotteiden yleistetyt syvyystiedot tuotetaan navigointikäyttöä varten tuotettujen syvyysmallien pohjalta (Liikennevirasto 2017c). Uutta syvyysaineiston tuotantoprosessia on sen pääpiirteissään kuvattu kuvan 2 prosessikaaviossa.

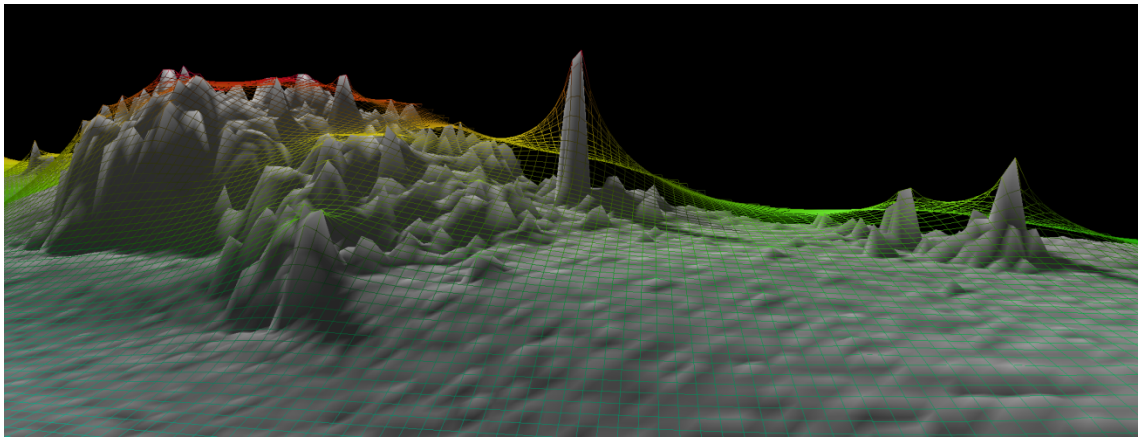


Kuva 2. Yksinkertaistettu prosessikaavio perinteisten syvyystietojen syvyysmallipohjaisesta tuottamisesta. Myös itse syvyysmallipintaa voidaan käyttää siitä johdettujen syvyystietojen laaduntarkastukseen.

Tällä hetkellä virallisia, navigointikäyttöön suunnattuja syvyysmallituotteita ei ole Suomen osalta vielä julkaistu ja niiden tuottamiseen liittyvät käytännöt ja määrittelyt ovat vielä kehitettävinä. Liikenneviraston digitalisaatiohankkeen (Liikennevirasto 2017d) osahankkeena toteutettava merenkulun älyväylä -hanke (Liikennevirasto 2017b) tähtää muiden tavoitteidensa ohella myös navigointikäyttöön soveltuvien syvyysmallituotteiden tuotannon määrittelyyn ja tuotannon aloittamiseen.

Kuvassa 2 esitetyssä yksinkertaistetussa prosessimallissa perinteisten, merikartalla esitettävien syvyystietojen tuottaminen perustuu syvyysmalleihin. Tällaisessa prosessimallissa lähtöaineiston laatu, lähinnä syvyysmallipintojen tasaisuus, vaikuttaa suuresti niistä johdettujen aineistojen tekniseen ja kartografiseen laatuun ja tätä kautta aineiston tuottamisessa tarvittavan manuaalisen työn määrään (Lecordix et al. 2009; Rosenkranz et al. 2012; Kettunen et al. 2017). Syvyysmallituotteiden tuotannossa käytettävistä menetelmistä riippuen saattaa valmiissa syvyysmalleissa esiintyä runsaastikin kohinaa. Kohina näkyy syvyysmallipinnan terävänä piikikkyytenä ja epätasaisuutena (kuva 3),

joka puolestaan vaikuttaa suoraan pinnasta automaattisesti tuotettujen syvyyskäyrien tekniseen ja kartografiseen laatuun (Calder ja Mayer 2003; Peters et al. 2014).



Kuva 3. Esimerkki kohinapitoisen ja pehmennetyn syvyysmallipinnan välisistä eroista. Harmaa pinta on kohinapitoinen syvyysmalli, värillinen verkko kuvaa navigointiturvallisesti pehmennettyä syvyysmallipintaa. Esimerkissä käytetty pehmennysmenetelmä on kappaleessa 2.3. esitelty navigointiturvallinen Laplace-interpolointi. Aineisto © Liikennevirasto 2018.

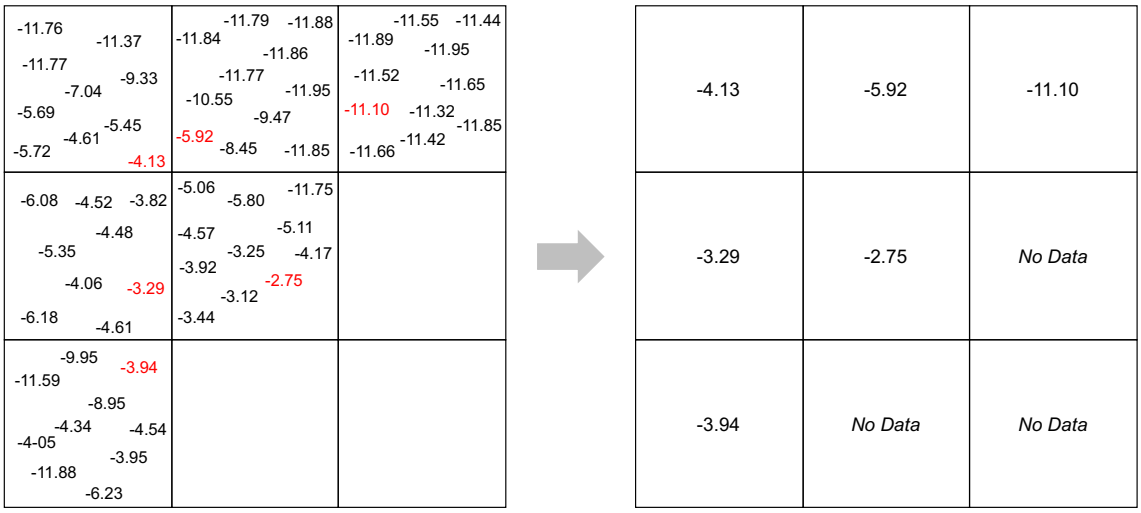
Kohinan vähentäminen tarkoittaa syvyysmallipinnoissa esiintyvän piikikkyyden ja epätasaisuuden vähentämistä syvyysmallipintoja tasoittamalla. Tällaista tasoittamista, eli mallipinnan *pehmentämistä (smoothing),* voidaan tehdä useilla eri menetelmillä. Tässä työssä selvitetään navigointiturvallisen *Laplace-interpoloinnin* ja kaksoispuskurointiin *(double buffering)* perustuvan *Rolling Coin* -menetelmän toimivuutta syvyysmallipintojen kohinan vähentämisessä. Työn päätavoite ei ole itse syvyysmallien muokkaus vaan syvyysmalleista automaattisesti johdettujen syvyyskäyrien laadun parantaminen. Seuraavissa kappaleissa käsitellään syvyysmallien tuottamista merenmittausaineistoista sekä syvyysmallipintojen yleistämisen ja pehmennyksen menetelmiä ja periaatteita.

2.1. Navigointiturvallisten syvyysmallipintojen tuottaminen

Parhaiten todellista pohjatopografiaa kuvaavat ja silti navigointiturvallisuuden säilyttävät syvyysmallit ovat pehmentämättömiä, matalia suosivalla *(Shoal Bias)* menetelmällä tuotettuja syvyysmallipintoja. Tämän tyyppisessä syvyysmallissa kukin syvyysmallin solu saa syvyysarvokseen matalimman kyseisen solun alueelle osuneen syvyyshavainnon arvon. Ne solut, joiden alueelle ei osu lainkaan syvyyshavaintoja, saavat arvokseen

tyhjää tarkoittavan *No Data* -arvon (kuva 4). Tällaisen syvyysmallin navigointiturvallisuus toteutuu, kun kunkin mallin solun arvon katsotaan kuvaavan syvyyttä koko kyseisen solun alueella, ei ainoastaan solun keskipisteessä (Peters 2012).

Laadukkaan ja luotettavan syvyysmallin pohjana toimivan pisteaineiston on oltava navigointiin tuotettavien syvyysaineistojen tuottamiseen soveltuvaa, käytännössä IHO:n S-44 -standardin (IHO 2008) mukaista, pisteaineistoa. Pisteaineistojen on oltava tarkastettuja ja merikartanvalmistukseen hyväksyttyjä.



Kuva 4. *Shoal Bias* -periaate rasterimuotoisten syvyysmallien tuottamisessa pisteaineistosta. Syvyysmallin solun arvoksi valitaan matalimman solun alueelle osuvan mittaushavainnon syvyys. Ne solut, joiden alueelle ei osu lainkaan mittaushavaintoja saavat tyhjää solua kuvaavan *No Data* -arvon. Turvallisuuden varmistamiseksi syvyysmallin soluarvoja ei interpoloida mallia tuotettaessa.

2.2. Syvyysmallipintojen käsittely

Puhekielessä rasterimuotoisten syvyysmallipintojen yleistämisellä voidaan tarkoittaa kahta toisistaan eroavaa asiaa:

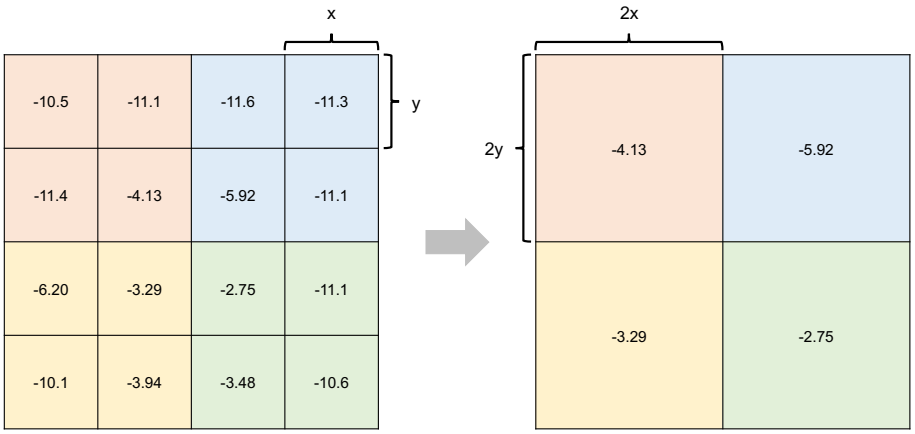
1. mallin spatiaalisen resoluution karkeistamista eli mallin solukoon kasvattamista
2. mallipinnan korkeataajuisen vaihtelun, eli *kohinan*, vähentämistä häivyttämällä syvyysmallipinnasta yksityiskohtia. Tätä voidaan kutsua myös syvyysmallipinnan *pehmentämiseksi* tai *silottamiseksi*.

Tässä työssä syvyysmallipintojen yleistämisellä tarkoitetaan pääsääntöisesti syvyysmallipintojen *pehmentämistä* korkeataajuisen kohinan poistamiseksi. Termi *pehmennys* on tässä työssä valittu *silottamisen* sijaan siitä syytä, etteivät käytetyt pintojen käsittelyn menetelmät välttämättä tuota matemaattiselta määritelmältään sileitä pintoja.

2.2.1. Syvyysmallipintojen yleistäminen eli spatiaalisen resoluution karkeistaminen

Syvyysmallipintojen yleistämisellä tarkoitetaan mallien spatiaalisen resoluution heikentämistä mallin solukokoa kasvattamalla. Yleistetyssä syvyysmallissa yksittäinen mallin solu vastaa siis maastossa suurempaa aluetta, kuin yleistämättömän syvyysmallin solu.

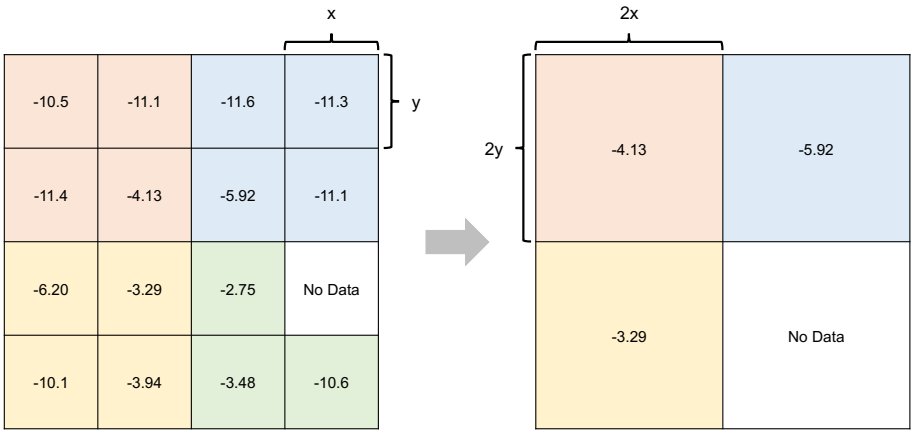
Navigointikäyttöön tarkoitettujen syvyysmallien tapauksessa myös syvyysmallien yleistäminen tulee tehdä turvallisuus huomioiden – syvyysmalli ei saa muuttua turvattomaksi, vaikka sitä yleistetäänkin. Kuvassa 5 on esitetty syvyysmallin yleistämistä navigointiturvallisin periaattein. Yksinkertaisin tapa yleistää syvyysmallipintoja navigointiturvallisuus huomioiden on soveltaa mallien tuottamisessa käytettävää matalia suosivaa periaatetta myös mallipintojen yleistämisessä. Tällöin yleistetyn syvyysmallin solun arvoksi määräytyy matalin niistä yleistämättömän syvyysmallin soluarvoista, jotka osuvat yleistetyn syvyysmallin solun alueelle. Yleistetyn ja yleistämättömän syvyysmallin solurajojen yhtyminen varmistetaan käyttämällä alkuperäisen syvyysmallin origoa ja spatiaalisen resoluution monikertaa *(yleistyskerrointa)*, kuten kuvan 5 esimerkissä.



Kuva 5. Matalia suosiva *Shoal bias* -periaate syvyysmallirasterien yleistämisessä karkeampaan spatiaaliseen resoluutioon. Yleistetyn syvyysmallin soluarvoksi valitaan matalin vastaavista alkuperäisen syvyysmallin solujen arvoista. Esimerkissä yleistyskerroin on 2.

Yleistyskerroin n voidaan määritellä joko x- ja y- akseleille erikseen tai käyttää samaa yleistyskerrointa molemmille akseleille. Kun syvyysmallin yleistäminen tehdään alkuperäisen mallin origoa ja spatiaalista resoluutiota ja erityistä yleistyskerrointa käyttämällä, voidaan varmistua että myös tuotettava yleistetty syvyysmalli säilyy navigoinnin kannalta turvallisena.

Kuvan 5 tapauksessa *yleistyskerroin* n on 2 ja yleistetyn syvyysmallin solun arvo määräytyy vastaavien yleistämättömän syvyysmallin solujen arvojen mukaan. Mikäli jollain yleistämättömän syvyysmallin soluista ei ole syvyysarvoa *(No Data)*, tulee turvallisuuden varmistamiseksi valita yleistetyn syvyysmallin soluarvoksi *No Data*. Kuvassa 6 on esitetty tilanne, jossa yhdessä yleistämättömän syvyysmallin solussa solun arvona on *No Data*. Koska tuntemattoman solun todellisesta syvyydestä ei ole tietoa, tulee myös yleistetyn mallin soluarvoksi valita *No Data*.



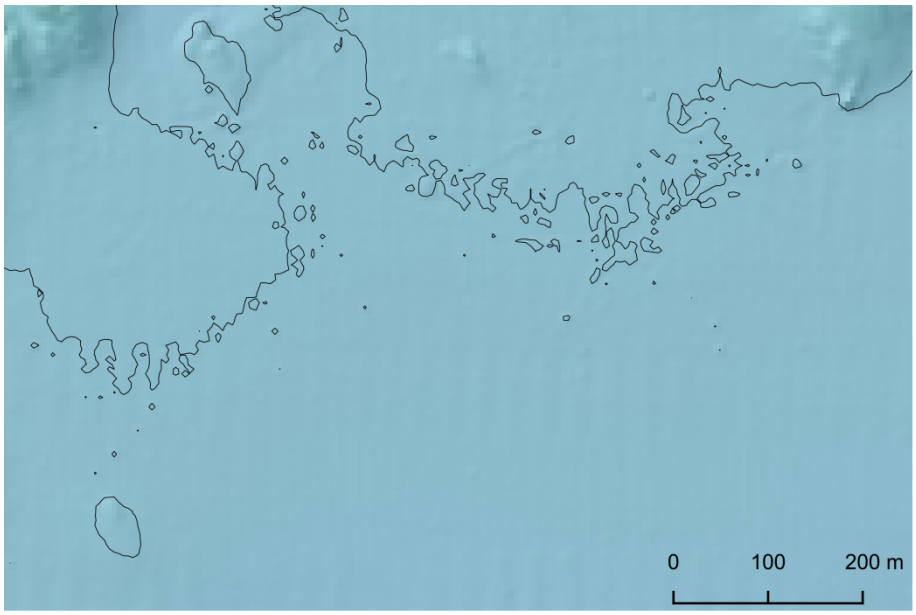
Kuva 6. Matalia suosiva *Shoal Bias* -periaate syvyysmallirasterien yleistämisessä karkeampaan spatiaaliseen resoluutioon. Puuttuva syvyystieto *(No Data)* tulkitaan aina vaarallisimmaksi ja siksi se valitaan myös yleistetyn syvyysmallin soluarvoksi. Esimerkissä yleistyskerroin on 2.

2.2.2. Syvyysmallipintojen pehmentäminen

Syvyysmallipintojen pehmentämisellä tarkoitetaan syvyysmallipinnan yksityiskohtien häivyttämistä mallin sisältämää korkeataajuista kohinaa tasoittamalla. Jotta syvyysmallien navigointiturvallisuus säilyisi, on kaikki mallipintojen pehmennys tehtävä turvalliseen suuntaan eli mallipintojen syvyyksiä madaltamalla.

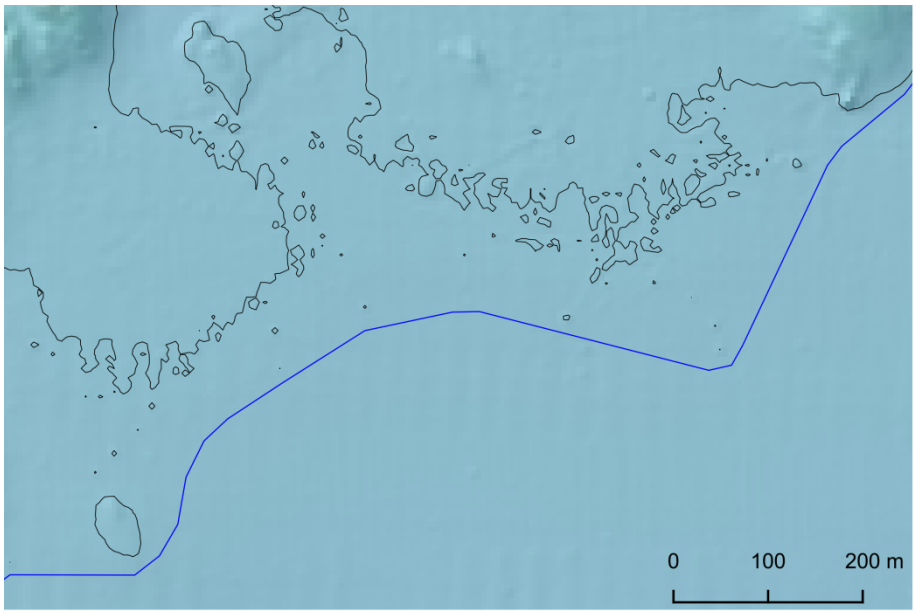
Kappaleessa 2.1. esitelty yksinkertainen, navigointiturvallinen syvyysmallipinta toimii melko hyvin varsinaisen navigointikäytön lisäksi todellista pohjatopografiaa kuvaavana tietotuotteena. Tällaisessa syvyysmallissa voi sen tuotantomenetelmästä johtuen kuitenkin olla mukana suuri määrä kohinaa, joka ilmenee käytännössä syvyysmallipinnan epätasaisuutena ja piikikkyytenä (kuva 3).

Mikäli tämänkaltaista kohinapitoista syvyysmallipintaa käytetään sellaisenaan syvyyskäyrien automaattisessa määrittämisessä, syntyy tuloksena usein hyvin suuri määrä laadultaan melko kehnoja syvyyskäyriä (Calder ja Mayer 2003; Calder ja Wells 2007; Peters et al. 2014). Käyrien kehno laatu ilmenee hyvin lyhyiden syvyyskäyrien suurena määränä ja yleisesti syvyyskäyrien voimakkaana mutkikkuutena (kuva 7).



Kuva 7. Esimerkki pehmentämättömän, matalia suosivalla *Shoal Bias* -menetelmällä tuotetun syvyysmallin käytöstä syvyyskäyrien automaattisessa määrittämisessä. Syvyyskäyrien lukumäärä on korkea, käyrät ovat hyvin mutkikkaita ja usein myös lyhyitä. Näistä syistä johtuen myös käyrien kartografinen laatu varsinaisten tuotteiden mittakaavoilla on heikko. Aineisto © Liikennevirasto 2018.

Merikarttatuotteilla esitettävien syvyyskäyrien tulee kartan luettavuuden vuoksi olla yleistettyjä (Zhang ja Guilbert 2011) – ideaalitilanteessa yksittäinen syvyyskäyrä yleistää sisäänsä lähekkäisten matalikkojen huiput eikä kaikkia erillisiä matalikkoja kuvata omalla syvyyskäyrällään. Kuvassa 8 nähdään esimerkki kuvan 7 alueesta, kun mukaan on liitetty yleistetty syvyyskäyrä.

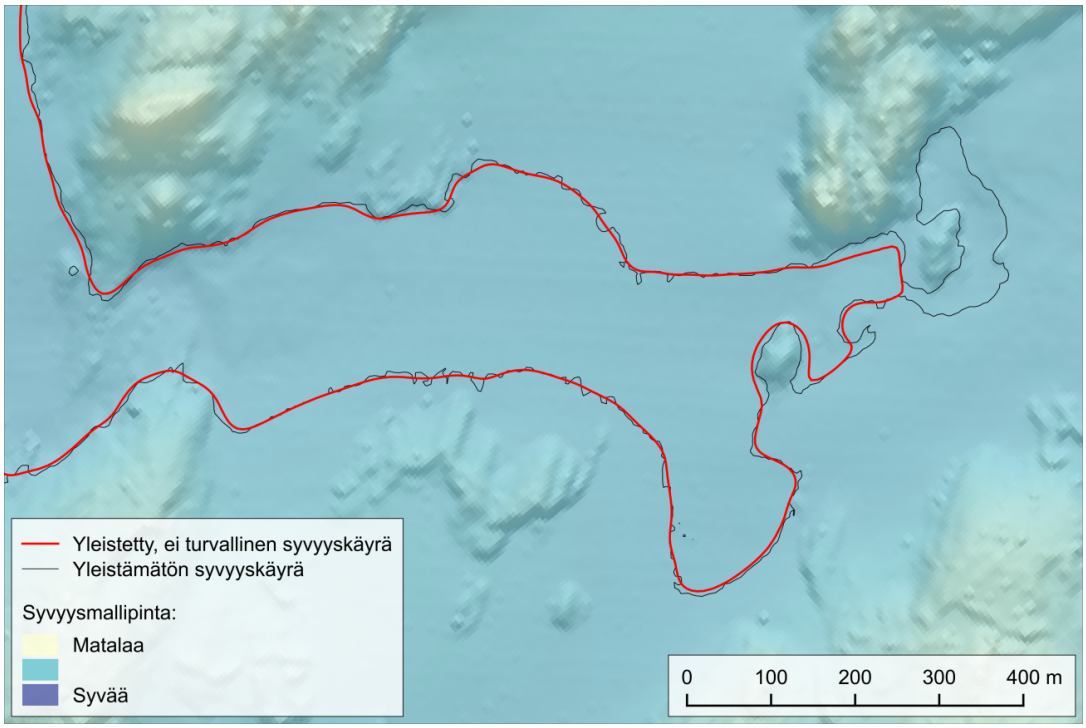


Kuva 8. Kuvan 7 esimerkkialue, johon on lisätty turvalliseen suuntaan yleistetty syvyyskäyrä. Aineisto © Liikennevirasto 2018.

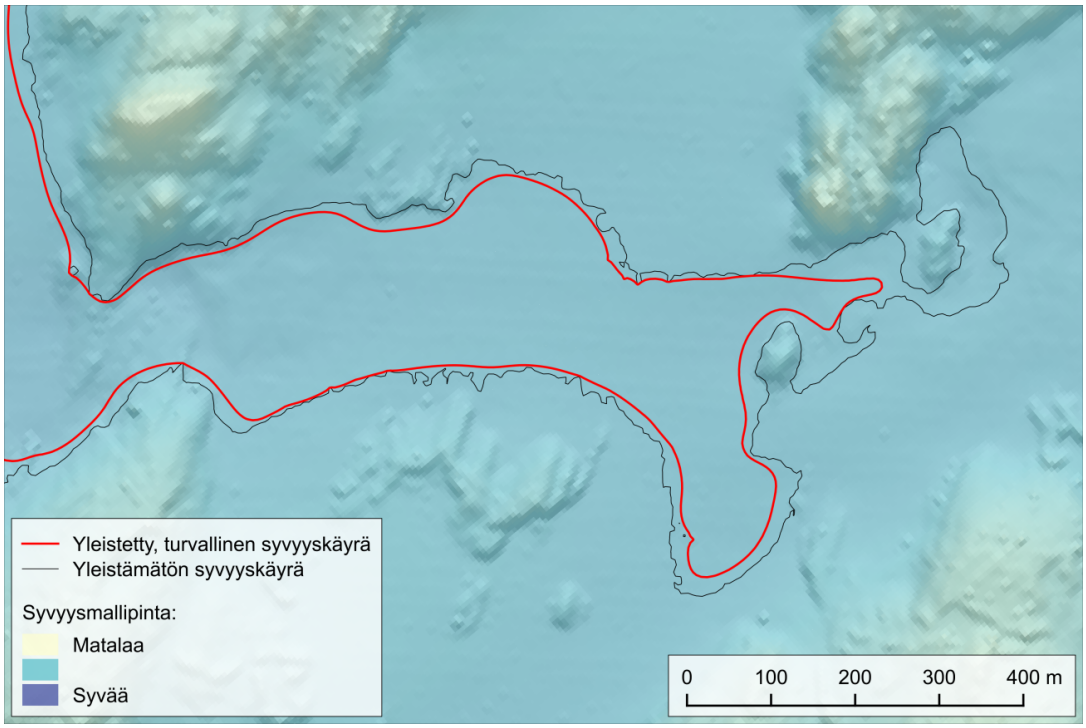
Kohinapitoista syvyysmallipintaa pehmentämällä voidaan saada aikaan pinta, josta automaattisin menetelmin määritetyt syvyyskäyrät yleistävät jo sisäänsä vierekkäisiä matalia ja ovat lisäksi kartografisesti ja tekniseltä laadultaan parempia kuin pehmentämättömästä raakapinnasta määritetyt syvyyskäyrät. Syvyysmallipintojen pehmennyksessä käytettävän menetelmän tulee kuitenkin olla navigoinnin kannalta turvallinen.

Kuvissa 9 ja 10 nähdään kaksi esimerkkiä pehmennetyistä syvyysmallipinnoista luoduista syvyyskäyristä. Yleistettyjen syvyyskäyrien (punaiset käyrät) kartografisen laadun voidaan nopeasti todeta olevan yleistämättömiä syvyyskäyriä (mustat käyrät) huomattavasti parempi.

Kuvan 9 tapauksessa syvyysmallipinnan pehmennyksessä käytetty menetelmä ei ole navigointiturvallinen. Tämä ilmenee pehmennetystä pinnasta määritetyn syvyyskäyrän (punainen) ja pehmentämättömästä pinnasta irrotetun käyrän (musta) useina risteämisinä – yleistetty syvyyskäyrä kulkee monin paikoin todellisen syvyyskäyrärajan matalammalla puolella. Kuvan 10 esimerkissä nähdään navigointiturvallisesti yleistetty syvyyskäyrä, joka kulkee aina annetun syvyysrajan syvemmällä puolella.



Kuva 9. Esimerkki ei-turvallisesta syvyyskäyrien yleistämisestä. Yleistetty syvyyskäyrä risteää monin paikoin todellista syvyysrajaa ja jättää pienen erillisen matalikon alueen vasemmassa reunassa huomioimatta. Aineisto © Liikennevirasto 2018.



Kuva 10. Esimerkki turvallisesta syvyyskäyrien yleistämisestä. Yleistetty syvyyskäyrä ei risteä todellista syvyysrajaa ja pysyy koko matkalta rajasyvyyden syvemmällä puolella. Alueen vasemman reunan erillinen pieni matalikko on huomioitu ja jää syvyyskäyrän matalammalle puolelle. Aineisto © Liikennevirasto 2018.

2.2.3. Fokaali- eli naapurusto-operaatiot ja konvoluutio

Rasterimuotoisen syvyysmalliaineiston pintojen pehmentämisessä voidaan hyödyntää rasteriaineistojen yhteydessä laajasti käytettyjä fokaali- eli naapurusto-operaatioita, joissa kohdesolun saama arvo määräytyy kyseisen solun ja sen naapuruston soluarvojen perusteella (Li ja Hodgson 2004). Käytettävän naapuruston koon ja siihen kuuluvien solujen painokertoimet määrittelee erityinen painokerroinmatriisi eli *kernel* (kuva 11). Fokaalioperaatioissa jokainen lähtöaineiston solu käsitellään vuorollaan siirtämällä painokerroinmatriisi solun kohdalle. Näin saadaan käsiteltyä lähes koko aineisto; ainoastaan lähtöaineistona toimivan rasteriaineiston reunat ja nurkat vaativat erityisen päätöksen operaatioiden toteuttamistavasta tai niiden toteuttamatta jättämisestä.

(-1,-1)	(-1,0)	(-1,1)
(0,-1)	(0,0)	(0,1)
(1,-1)	(1,0)	(1,1)

Kuva 11. Esimerkki fokaalioperaatioissa käytettävästä painokerroinmatriisista eli *kernelistä.* Kerneli määrittelee käytettävän naapuruston ja kunkin siihen kuuluvan solun painokertoimen. Suluissa esitetyt koordinaatit ovat matriisin alkioiden suhteellista sijaintia kuvaavia *indeksejä*.

Jos naapuruston painokertoimet määrittelevän painokerroinmatriisin kaikki painokertoimet ovat positiivisia, lasketaan kohdesolun arvoksi käytännössä kernelin määrittelemän naapuruston painotettua keskiarvoa. Tällainen fokaalioperaatio toimii *alipäästösuotimen* tavoin suodattamalla aineistosta sen sisältämää korkeataajuista kohinaa eli vähentämällä aineiston yksityiskohtaisuutta.

Yllä kuvattujen fokaalioperaatioiden toteutusta kutsutaan *konvoluutioksi* (Ritter et al. 1990). Syvyysmallin kaltaisten rasterimuotoisten aineistojen konvoluutio on painokerroinmatriisin ja lähtöaineiston lineaarikombinaatio. Lineaarikombinaatiossa termien joukko, tässä tapauksessa tarkasteltava syvyysmallin solu ja sen painokerroinmatriisissa

määritelty naapurusto, kerrotaan kukin omalla painokertoimellaan ja saadut tulot lasketaan yhteen.

Konvoluutiossa käytettävän painokerroinmatriisin koko voi periaatteessa olla mielivaltainen, mutta usein mielekkäintä on käyttää neliömatriisia eli matriisia, jossa rivien ja sarakkeiden lukumäärä on sama. Kun rivien ja sarakkeiden lukumääräksi valitaan pariton luku, saadaan konvoloitavaa syvyysmallin solua vastaamaan painokerroinmatriisin keskimmäinen alkio ja konvoluutiossa käytettävä naapurusto on konvoloitavan syvyysmallin solun suhteen symmetrinen.

Edellä kuvattujen määritelmien mukaisesti konvoluution painokerroinmatriisin tyypilliseksi kooksi saadaan $n \times n$, jossa $n = 2k + 1$. Näin määriteltynä syvyysmallin M konvoluutio painokerroinmatriisilla W on muotoa $f = W * M$ eli:

$$f_{(x,y)} = \sum_{j=-k}^{k} \sum_{i=-k}^{k} W_{(i,j)} \cdot M_{(x-i,\ y-j)}$$

Jos konvoluution tarkoituksena on laskea konvoloitavan solun ja sen naapuruston painokerroinmatriisin mukaisilla kertoimilla painotettu keskiarvo, tulee huomioida, että painokerroinmatriisin alkioiden summan on oltava 1. Kaikissa muissa tapauksissa päädytään tilanteeseen, jossa itse konvoluution tulos saa jonkin kiinteän, painokertoimien summasta riippuvan painokertoimen. Painokertoimien normalisointi kannattaakin lisätä osaksi konvoluutiota. Tällöin saadaan konvoluution kaavaksi:

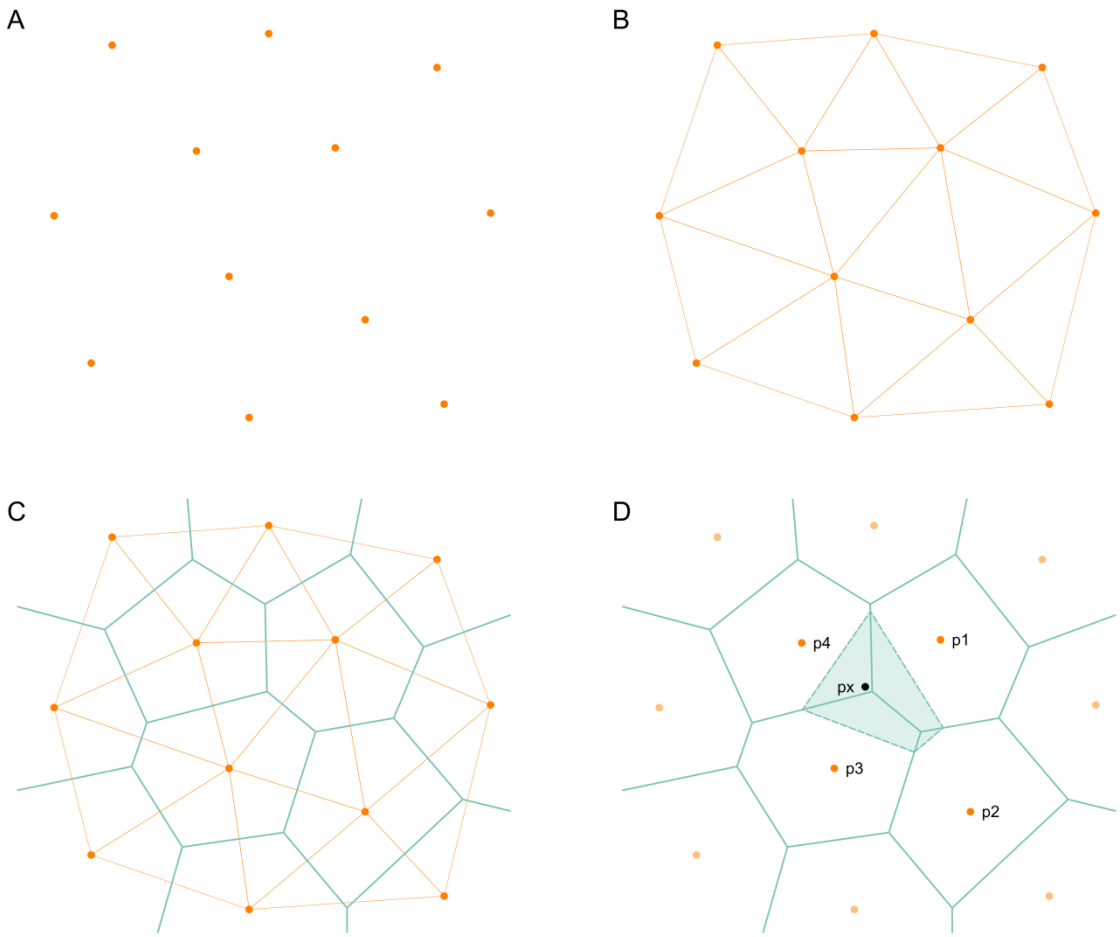
$$f_{(x,y)} = \frac{1}{\sum_{i=1}^{n} \sum_{j=1}^{n} W_{(i,j)}} \cdot \sum_{j=-k}^{k} \sum_{i=-k}^{k} W_{(i,j)} \cdot M_{(x-i,\ y-j)}$$

2.3. Navigointiturvallinen iteratiivinen Laplace-interpolointi

Syvyysmallipintojen navigointiturvallisessa pehmennyksessä Laplace-interpolointia on tutkittu aiemmin Petersin (2012) ja Peters et al. (2014) toimesta. Navigointiturvallinen Laplace-interpolointiin perustuva syvyysmallipintojen pehmennysmenetelmä on implementoitu myös joihinkin kaupallisiin sovelluksiin, kuten merikartoituksen erityistarpeisiin erikoistuneeseen CARIS Base Editor -ohjelmaan (CARIS 2017).

Menetelmän tausta on Sibsonin vuonna 1981 (Sibson 1981; Sukumar et al. 2000; Kotulak et al. 2017) kehittämässä epäsäännöllisille pistejoukoille soveltuvassa spatiaalisessa interpolointimenetelmässä, jossa interpoloitavan pisteen arvo lasketaan kyseisen pisteen *luonnolliseen naapurustoon (natural neighbor)* kuuluvien pisteiden lineaarikombinaationa. Sibsonin kehittämä menetelmä perustuu pisteiden vaikutusalueiksi ajateltuihin Voronoi-polygoneihin, joiden perusteella myös luonnolliseen naapurustoon kuuluvat pisteet ja niiden painokertoimet määräytyvät.

Kuvassa 12 (12.A, 12.B, 12.C) on esitetty epäsäännöllinen pistejoukko ja siitä Delaunay-kolmioinnilla muodostetut TIN-verkko ja Voronoi-polygonit. Kuvassa 12.D samaan pistejoukkoon on lisätty uusi, interpoloitava piste p_x. Tämän pisteen muodostama uusi Voronoi-polygoni on esitetty vihertävällä täyttövärillä. Sibsonin (1981) määritelmän mukaisesti interpoloitavan pisteen p_x luonnollinen naapurusto koostuu niistä pisteistä, joiden Voronoi-polygoni jakaa rajan pisteen p_x Voronoi-polygonin kanssa (kuva 12.D). Siten kuvan 12 esimerkissä pisteen p_x luonnollisen naapuruston muodostavat pisteet p_1, p_2, p_3 ja p_4.

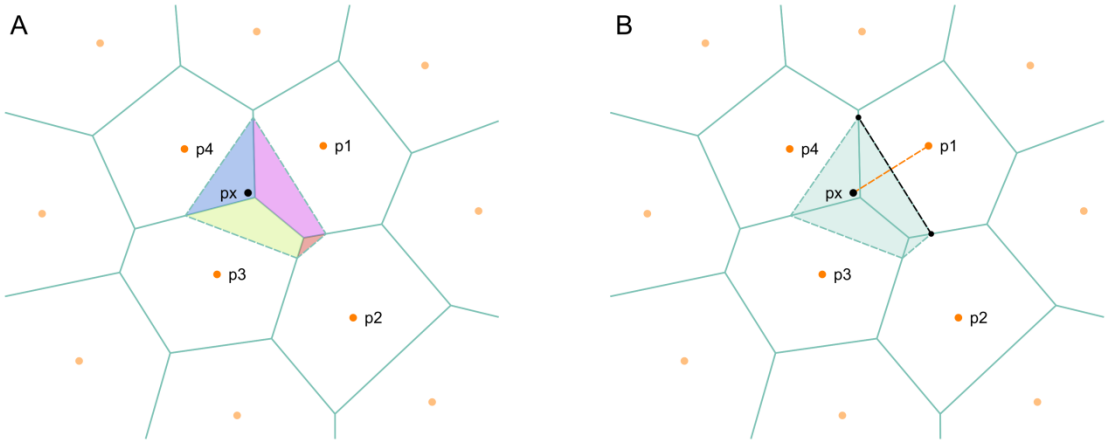


Kuva 12. Epäsäännöllisen pistejoukon (A) Delaunay-kolmioinnilla muodostettu TIN-verkko (B), Voronoi-polygonit (C) ja uuden, interpoloidun pisteen p_x muodostama Voronoi-polygoni (D).

Sibsonin (1981) kehittämässä *Sibsonin interpoloinnissa* interpoloitavan pisteen p_x luonnolliseen naapurustoon kuuluvan pisteen p_i painokerroin riippuu pisteen p_x muodostaman Voronoi-polygonin V_{p_x} ja pisteen p_x luonnolliseen naapurustoon kuuluvan pisteen p_i muodostaman Voronoi-polygonin V_{p_i} välisen leikkauksen pinta-alan suhteesta pisteen p_x Voronoi-polygonin V_{p_x} pinta-alaan:

$$W_i = \frac{Pinta - ala(V_{p_x} \cap V_{p_i})}{Pinta - ala(V_{p_x})}$$

Kuvassa 13.A on kuvattu Sibsonin interpoloinnissa käytettävien painokertoimien määräytymistä. Pisteen p_x Voronoi-polygonin värilliset osapolygonit ovat pisteen p_x Voronoi-polygonin ja luonnolliseen naapurustoon kuuluvien pisteiden Voronoi-polygonien leikkauksia.

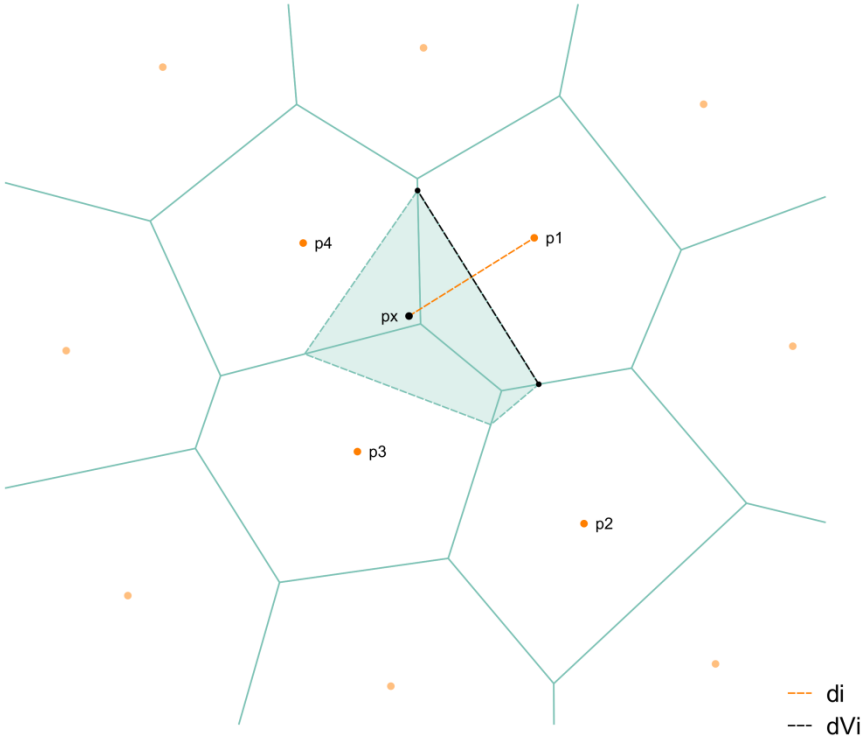


Kuva 13. Sibsonin- (A) ja Laplace- (B) interpolointimenetelmien painokertoimien määritys. Molemmat menetelmät perustuvan Sibsonin luonnollisen naapuruston *(Natural Neighbor)* määritelmään. Esimerkissä pisteen p_x luonnolliseen naapurustoon kuuluvat pisteet p_1, p_2, p_3 ja p_4.

Sibsonin interpoloinnissa käytettyjen painokertoimien määrittäminen vaatii pinta-alojen laskemista ja on siksi menetelmänä työläs ja laskennallisesti vaativa. Luonnollisen naapuruston pisteiden painokertoimien määrittämiseen löytyy kuitenkin suoraviivaisempikin tapa, kuvassa 13.B esitetty *Laplace-interpolointi* (Christ et al. 1982, Belikov et al. 1997, Hiyoshi ja Sugihara 1999, Peters 2012). Laplace-interpoloinnissa painokertoimet lasketaan interpoloitavan pisteen p_x ja tämän luonnolliseen naapurustoon kuuluvan pisteen p_i välisen euklidisen etäisyyden ja kyseisten pisteiden välisen janan kanssa kohtisuoran Voronoi-polygonin reunan pituuksien suhteena (Peters 2012; Peters et al. 2014 mukaillen):

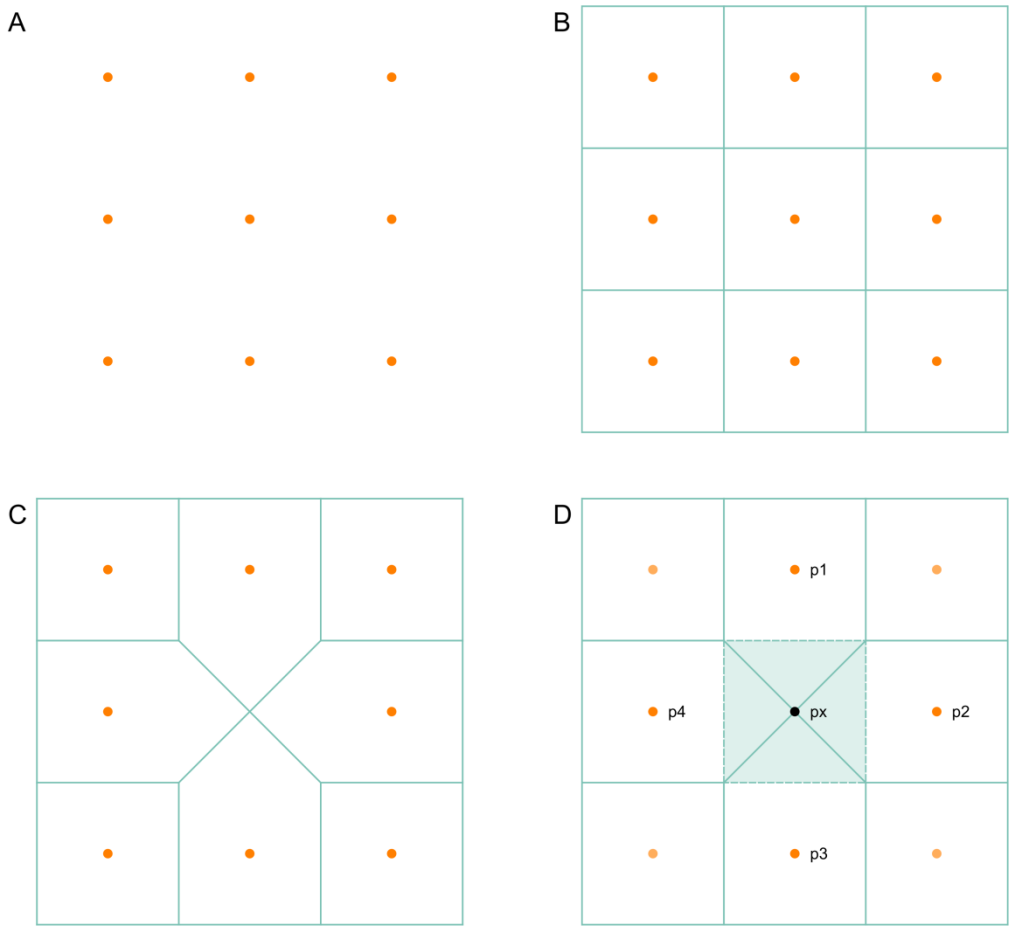
$$W_i = \frac{d_i}{d_{V_i}}$$

jossa d_i on pisteiden p_x ja p_i välinen euklidinen etäisyys ja d_{V_i} näitä pisteitä yhdistävän janan suhteen kohtisuoran Voronoi-polygonin reunan pituus (kuva 14).



Kuva 14. Luonnolliseen naapurustoon kuuluvien pisteiden saamien painokertoimien laskenta Laplace-interpolointimenetelmässä.

Koska rasterimuotoisen syvyysmalliaineiston voidaan ajatella koostuvan tasavälisestä pistejoukosta (kuva 15.A), yksinkertaistuu Laplace-interpolointi huomattavasti verrattuna edellä kuvattuun epäsäännöllisten pistejoukkojen tapaukseen. Tasavälisestä pistejoukosta muodostetun Voronoi-diagrammin (kuva 15.B) polygonit yhtyvät rasterimuotoisen syvyysmallin soluihin ja niiden rajoihin. Kun jonkin syvyysmallin solun arvo asetetaan tuntemattomaksi (kuva 15.C), muodostuu interpoloitavan pisteen p_x Voronoi-polygoni ja luonnollisen naapuruston muodostavat solut (p_1, p_2, p_3 ja p_4) kuvan 15.D mukaisesti.

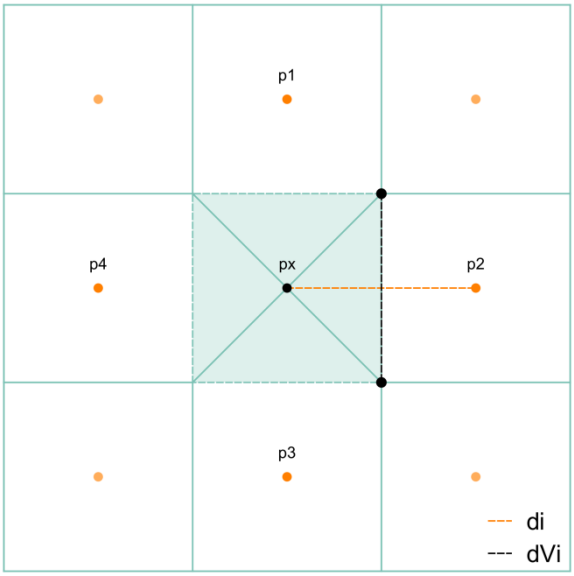


Kuva 15. Tasavälinen pistejoukko (A) ja siitä muodostetut Voronoi-polygonit (B). Jos yksi pistejoukon pisteistä asetetaan tuntemattomaksi (C), ”laajenevat” lähimpien naapurien Voronoi-polygonit tyhjään soluun. Kun puuttuvan arvon tilalle interpoloidaan uusi arvo p_x, muodostuu sen Voronoi-polygonista naapuriensa kaltainen (D). Pisteen p_x luonnollinen naapurusto on siis pisteet $p_1 - p_4$.

Rasterimuotoisen aineiston tapauksessa myös interpoloitavan solun naapurustoon kuuluvien solujen painokertoimien määrittäminen on epäsäännöllisiä pistejoukkoja huomattavasti yksinkertaisempaa. Rasterimuotoisella syvyysmalliaineistolla luonnolliseen naapurustoon kuuluvien syvyysmallin solujen saamat painokertoimet voidaan laskea suoraan syvyysmallirasterin spatiaalisen resoluution (tai spatiaalisten resoluutioiden) perusteella kuvassa 16 esitetyn periaatteen mukaisesti. Itse painokertoimet lasketaan samaan tapaan kuin edellä esitettyjen epäsäännöllisten pistejoukkojen tapauksessa:

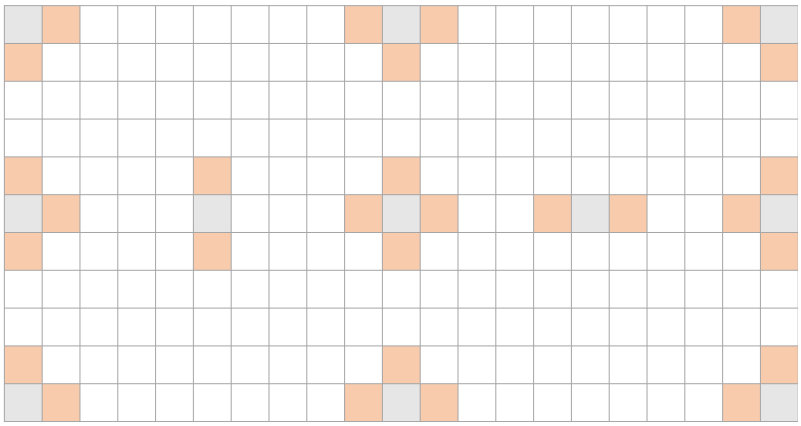
$$W_i = \frac{d_i}{d_{V_i}}$$

jossa d_{V_i} ja d_i ovat tarkasteltavasta naapurista riippuen joko syvyysmallin x- tai y-suuntaisia spatiaalisia resoluutioita (kuva 16). Koska tässä työssä käytetyn syvyysmalli-aineiston spatiaalinen resoluutio (taulukko 1) on sama x- ja y-suunnissa, on kaikkien interpoloitavan solun naapurustoon kuuluvien solujen saama painokerroin 1.



Kuva 16. Naapuruston painokertoimien laskenta rasterimuotoisen aineiston tapauksessa. Painokertoimet lasketaan yksinkertaisesti aineiston spatiaalisen resoluution avulla.

Syvyysmallin kaltaisilla rasterimuotoisilla aineistoilla interpoloitavan solun luonnollinen naapurusto koostuu kuvan 16 esimerkin mukaisesti yleensä neljästä tarkasteltavaa solua lähimmästä solusta *(Rooks contiguity)*. Erikoistapauksissa, kuten syvyysmallin kulmissa ja reunoissa, tai jos naapurustoon osuu tyhjää tarkoittavia *No Data* -arvoja, muodostuu naapurusto toisenlaiseksi. Kaikki rasterimuotoisten aineistojen mahdolliset naapurustot on esitelty kuvassa 17.



Kuva 17. Rasterimuotoisen aineiston Laplace-interpoloinnissa käytettävien naapurustojen vaihtoehdot. Interpoloitava solu on esitetty harmaalla ja käytetty naapurusto punertavalla värillä. Kulmissa ja reunoissa esitetyt naapurustot voivat luonnollisesti esiintyä myös muualla aineiston alueella.

Tässä työssä käytetyn Laplace-interpoloinnin toteuttavan sovelluksen (liite 1) toteutus on tehty siten, että interpoloinnissa käytettyjen naapurisolujen minimilukumäärä on 2. Mikäli interpoloitavalla solulla on alle 2 validia, eli todellisen syvyystiedon sisältävää (solun arvo ei ole *No Data*) naapuria, ei interpolointia suoriteta ja alkuperäinen soluarvo jää voimaan.

Kun interpoloitavan solun luonnolliseen naapurustoon kuuluvien solujen painokertoimet on laskettu, saadaan tarkasteltavan solun syvyydelle interpoloitua syvyysarvio $\hat{Z}$ syvyysmallin ja painokerroinmatriisin konvoluutiona (kaavassa on mukana myös painokertoimien normalisointi):

$$\hat{Z}_{(x,y)} = \frac{1}{\sum_{i=-1}^{1}\sum_{j=-1}^{1} W_{(i,j)}} \cdot \sum_{i=-1}^{1}\sum_{j=-1}^{1} W_{(i,j)} \cdot Z_{(x-i,\ y-j)}$$

Koska syvyysmallipintojen käsittelyssä käytettävien operaatioiden navigointiturvallisuuden voidaan katsoa toteutuvan vain siinä tapauksessa, että pintaan tehtävät muutokset eivät syvennä alkuperäisiä syvyysmallin soluarvoja, valitaan solun lopulliseksi arvoksi $Z_{(x,y)}$ joko solun alkuperäinen arvo $Z_{(x,y)}$ tai solun syvyyden interpoloitu arvio $\hat{Z}_{(x,y)}$ seuraavasti (olettaen, että syvyysmallin solujen arvot ovat negatiivisia vertaustason alapuolella eli z-akselin arvot kasvavat ylöspäin):

$$Z_{(x,y)} = \begin{cases} \hat{Z}_{(\mathrm{x,y})}, & \hat{Z}_{(\mathrm{x,y})} > Z_{(x,y)} \\ Z_{(x,y)}, & \hat{Z}_{(\mathrm{x,y})} \leq Z_{(x,y)} \end{cases}$$

Menetelmän tuottama syvyysmallipinnan lopullinen pehmennys riippuu pehmennykseen käytettyjen interpolointikierrosten lukumäärästä. Myös tässä työssä käytetty menetelmän implementaatio (liite 1) on toteutukseltaan iteratiivinen, jolloin jokainen interpolointikierros tasoittaa syvyysmallipintaa entisestään.

Kuvissa 18 ja 19 on esitelty navigointiturvallisen Laplace-interpoloinnin toimintaa kahdessa erilaisessa tilanteessa. Kuvan 18 tapauksessa interpoloitava kohdesolu on ympäristöään syvempi, paikallinen syvänne. Kun kyseiselle solulle interpoloidaan syvyysarvio naapurustoonsa perustuen, on tämä arvio matalampi kuin solun alkuperäinen syvyys. Tällöin matalampi interpoloitu syvyysarvio valitaan pehmennetyn syvyysmallipinnan soluarvoksi.

-5.80	-5.35	-5.67	-5.42	-5.83
-5.14	-5.37	-5.99	-5.61	-5.77
-5.49	-5.29	-6.05	-4.39	-6.13
-5.34	-5.33	-4.73	-4.27	-5.52
-5.08	-5.14	-5.67	-5.55	-4.96

$*$

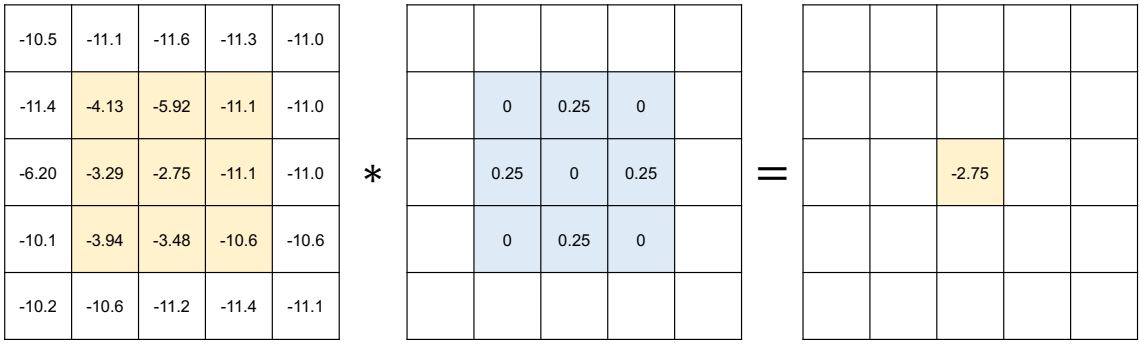
	0	0.25	0	
	0.25	0	0.25	
	0	0.25	0	

$=$

		-5.10		

Kuva 18. Laplace-interpolointi paikallisen syvänteen tapauksessa. Syvänne mataloituu interpoloinnin seurauksena.

Kuvassa 19 tilanne on toisenlainen. Tässä tapauksessa interpoloitava solu kuvaa paikallista matalan huippua eli ympäristönsä matalinta syvyyttä. Tällöin kyseisen solun naapurustoon perustuva Laplace-interpolointi tuottaa alkuperäistä syvyyttä syvemmän syvyysarvion, jolloin interpoloitua arviota matalampi alkuperäinen syvyys valitaan pehmennetyn syvyysmallipinnan soluarvoksi. Yleisesti voidaan siis todeta menetelmän säilyttävän matalien huiput, mutta madaltavan syvänteitä.



Kuva 19. Laplace-interpolointi matalikon huipun tapauksessa. Matalikon matalin syvyysarvo ei muutu interpoloinnin seurauksena.

2.4. Rolling Coin

Rolling Coin -menetelmä sai alkunsa lääketieteen kuvantamistarpeisiin kehitetyn ja sittemmin myös merikartoituksen tarvitsemien syvyysaineistojen käsittelyssä jokseenkin tunnetun *Rolling Ball* -menetelmän jatkokehitelmänä. Alkuperäisen *Rolling Ball* -menetelmän (Sternberg 1983; Hashim 1996; Rashed 2016) tarkoituksena oli erottaa harmaasävykuvien varsinainen signaali ja tausta toisistaan. Tällaisessa käyttötapauksessa menetelmän toimintaa voidaan ajatella kuvan muodostaman pinnan alla pyörivänä pallona – ne kuvan muodostaman pinnan kohdat, joihin pallon pinta ei milloinkaan kosketa, ovat varsinaista signaalia ja muu osa on taustaa. Käytettäessä menetelmää syvyysmallipintojen pehmentämiseen tulee kuvitteellista palloa kuitenkin pyörittää alkuperäisestä käyttötavasta poiketen syvyysmallipinnan yläpuolella. Näin toimimalla varmistetaan *Rolling Ball* -menetelmällä pehmennettyjen syvyysmallipintojen navigointiturvallisuus, sillä kaikki alkuperäiseen syvyysmallipintaan tehtävät muutokset tehdään turvalliseen suuntaan, ts. syvyysmallipintaa madaltamalla (Smith 2003; Heiskanen 2008; Peters 2012).

Smith esitteli vuoden 2003 tutkielmassaan *”The navigation surface: a multipurpose bathymetric database”* (Smith 2003) uuden saumattoman ja eriaikaisista ja erityyppisistä lähtöaineistoista luodun syvyysmallipinnan ja -tietokannan konseptin. Osana työtään Smith käytti syvyysmallipintojen pehmentämiseen *Rolling Ball* -menetelmää. Smith määritteli työssään menetelmän toteutuksen 3-ulotteisena *kaksoispuskurointina*, jossa kukin syvyysmallipinnan solun noodi puskuroidaan ensin 3-ulotteisesti ylöspäin (vain

alkuperäisen syvyysmallipinnan yläpuolisen puolipallon huomioiden) annetun etäisyyden päähän, jonka jälkeen saatu pinta puskuroidaan takaisin alaspäin samalla etäisyydellä. Smithin työn pohjalta jatkettu ja edelleen kehittyvä Open Navigation Surface -projekti (Open Navigation Surface Working Group 2017) tarjoaa avoimeen lähdekoodiin perustuvan kirjaston, joka sisältää esimerkiksi *Bathymetric Attributed Grid* (.BAG) -tiedostoformaatin määrittelyn (Open Navigation Surface Working Group 2004) ja lukuisia työkaluja syvyysmallitiedostojen käyttöön.

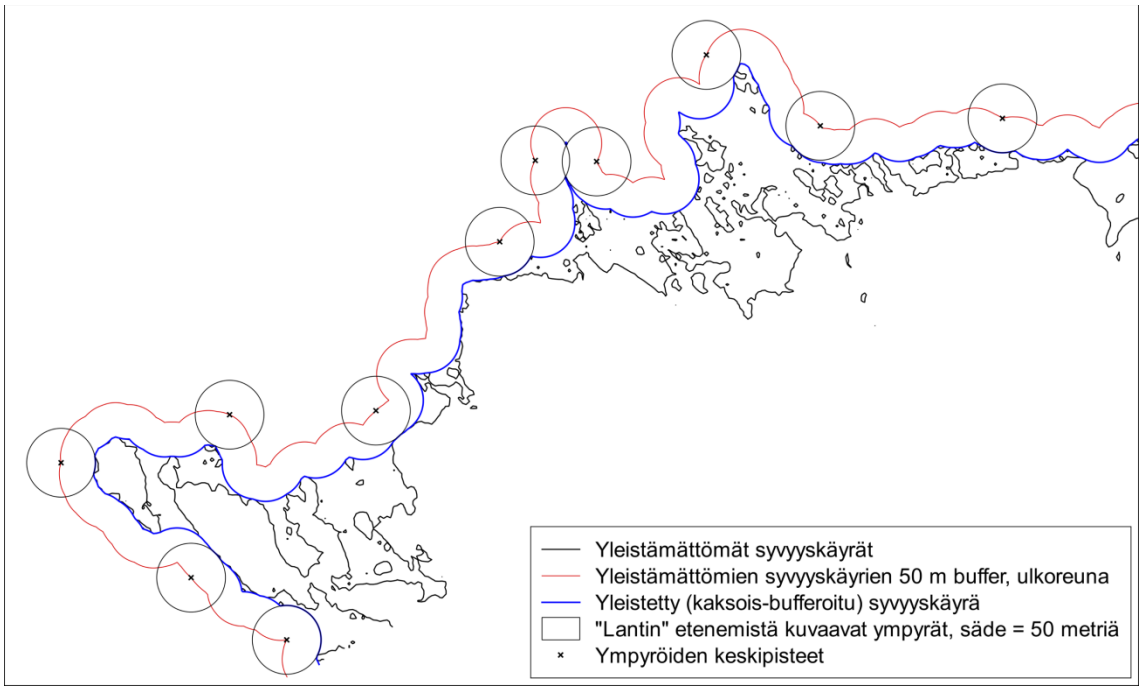
Suomen virallisessa merikartoitustoiminnassa *Rolling Ball* -menetelmän käyttöä syvyysmallipintojen pehmennykseen ja syvyyskäyrien tuotantoon on vuosien varrella testattu parillakin eri sovelluksella. Heiskanen (2008) kehitti diplomityössään menetelmän pohjalta sovelluksen, jolla syvyysmallipintoja voidaan pehmentää. Lisäksi kaupallisista sovellustarjoajista ainakin CARIS (2017) on implementoinut menetelmän osaksi tarjoamaansa Base Editor -sovellusta.

2.4.1. Alkuperäinen, syvyyskäyrärajat tuottava Rolling Coin

Rolling Ball -menetelmän käytännön testeissä (Heiskanen 2008) on saatu jokseenkin lupaavia tuloksia, mutta menetelmällä pehmennetyistä pinnoista vektoroitujen syvyyskäyrien laatu ei kuitenkaan ole edelleenkään ollut niin hyvä, että menetelmää oltaisiin otettu varsinaiseen tuotantokäyttöön. Lisäksi menetelmään liittyvien parametrien valinta ja asettaminen on koettu hankalaksi eikä yleisiä, useimmilla alueilla hyvin toimia parametreja ole saatu määritettyä. Heiskasen (2008) mukaan parametrien valinta osoittautui erityisen haastavaksi Suomessa yleisillä matalilla ja kapeilla vesialueilla. Liikenneviraston Merikartoituspalveluissa vuosina 2016–2018 toteutettavan älyväyläprojektin (Liikennevirasto 2017b) osana toteutetussa syvyysmalliprojektissa selvitettiin uusien IHO S-102 -standardin (IHO 2012) mukaisten syvyysmallituotteiden tuottamisen lisäksi myös syvyysmallipintojen pehmennysmenetelmiä ja menetelmien soveltuvuutta automatisoituun syvyyskäyräntuotantoon. Osana tätä työtä syntyi *Rolling Ball* -menetelmään perustuva *Rolling Coin* -menetelmä (Liikennevirasto 2018b), jonka perusajatuksena oli tuottaa *Rolling Ball* -menetelmästä johdettu yksinkertaistettu 2D-tasokehitelmä.

Tämän alkuperäisen *Rolling Coin* -menetelmän (Liikennevirasto 2018b) tarkoitus on tuottaa syvyysmallipinnoista abstrakteja, yleistetyt syvyyskäyrien rajat sisältäviä syvyysmallipintoja, joista varsinaiset vektorimuotoiset syvyyskäyrät voidaan määrittää.

Alkuperäisessä *Rolling Coin* -menetelmässä syvyyskäyrien rajat sisältävien pintojen tuottamiseen käytetty algoritmi on läheistä sukua vektoriaineistojen 2-ulotteiselle kaksoispuskuroinnille. Kuvassa 20 on kuvattu vektorimuotoisen aineiston 2-ulotteisessa tasossa tapahtuvan kaksoispuskuroinnin perusperiaate. Kuvasta nähdään, että kaksoispuskurointimenetelmällä yleistetty syvyyskäyrä voidaan nähdä puskurointietäisyyttä säteenään käyttävän ympyrän eli "lantin" kulkua kuvaavana ulkorajana – kun kuvitteellinen lantti kulkee syvyyskäyrärajan syvemmällä puolella, syntyy lantin kulkeman alueen ulkoreunoista tuloksena navigointiturvallinen, yleistetty syvyyskäyräraja.

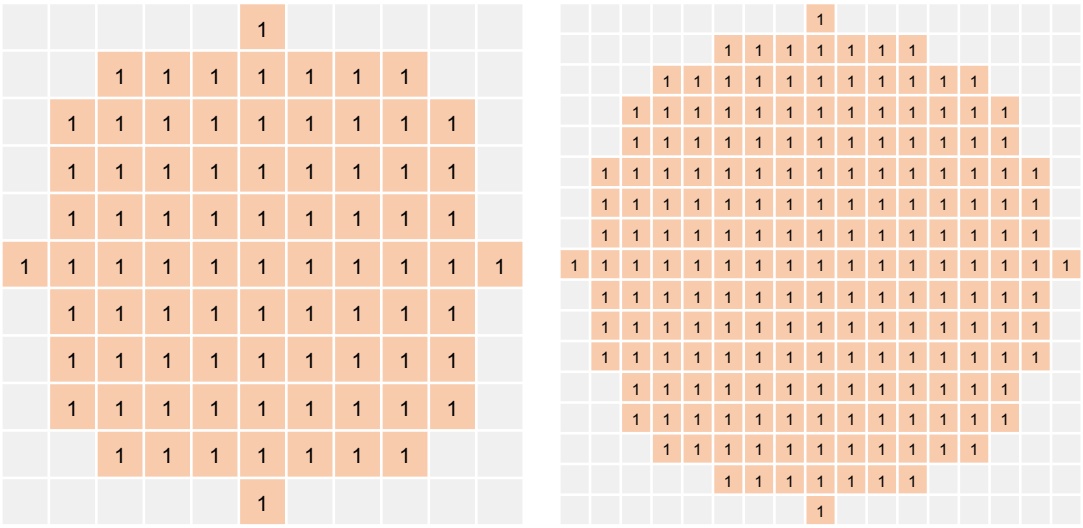


Kuva 20. Kaaviokuva kaksoispuskurointimenetelmän ja *Rolling Coin* -menetelmän yhteneväisyydestä. Aineisto © Liikennevirasto 2018.

Rolling Coin -menetelmässä syvyysmallipintojen pehmentämiseen käytettävä ”lantti” (kuva 21) on käytännössä 2-ulotteinen naapurustomatriisi, jonka alkioiden arvo on joko 0 tai 1. Tässä työssä käytetyt lantit luotiin Pythagoraan lauseen avulla ja ne approksimoivat muodoltaan ympyrää. Menetelmä itsessään ei kuitenkaan rajoita käytetyn lantin

muotoa, vaan se voisi yhtä hyvin olla esimerkiksi neliö, heksagoni, oktagoni tai jokin muu kuvio.

Rolling Coin eroaakin perinteisestä kaksoispuskuroinnista siinä, että *Rolling Coin* -menetelmässä puskurointietäisyys ei välttämättä ole vakio, vaan tarkasteltavan naapuruston määrittelee lantin koko ja muoto. Kuvassa 21 nähdään esimerkkejä eri kokoisista, ympyrää approksimoivista lanteista. Tässä työssä käytettyjen lanttien säteet olivat 5 ja 10 solua.



Kuva 21. Kaksi esimerkkiä Rolling Coin -menetelmässä käytettävistä ympyrää approksimoivista ”lanteista”. ”Lantti” on naapurustomatriisi, jonka alkiot voivat saada arvon 1 tai 0. Kuvissa harmaalla väritetyt alkiot ovat arvoltaan 0. Vasemmalla lantin säde r = 5 solua, oikealla lantin säde r = 8 solua.

2.4.2. Syvyysmallipintoja muokkaava 2,5-ulotteinen Rolling Coin

Alkuperäisen, 2-ulotteisessa tasossa toimivan *Rolling Coin* -menetelmän toiminnan periaate on laajennettavissa myös tässä työssä tutkittuun syvyysmallipintojen 2,5-ulotteiseen pehmennykseen soveltuvaksi. Syvyysmallipintojen pehmennykseen sovelletussa *Rolling Coin* -menetelmässä lantin määrittelemän naapuruston matalin syvyys laajennetaan koskemaan koko naapurustoa niin, että samalla koko syvyysmallipinta pyritään pitämään mahdollisimman syvänä. Menetelmä tarvitsee parametrikseen ainoastaan käytettyä naapurustoa kuvaavan matriisin eli lantin. Menetelmän toimintaa voidaan kuvata seuraavasti:

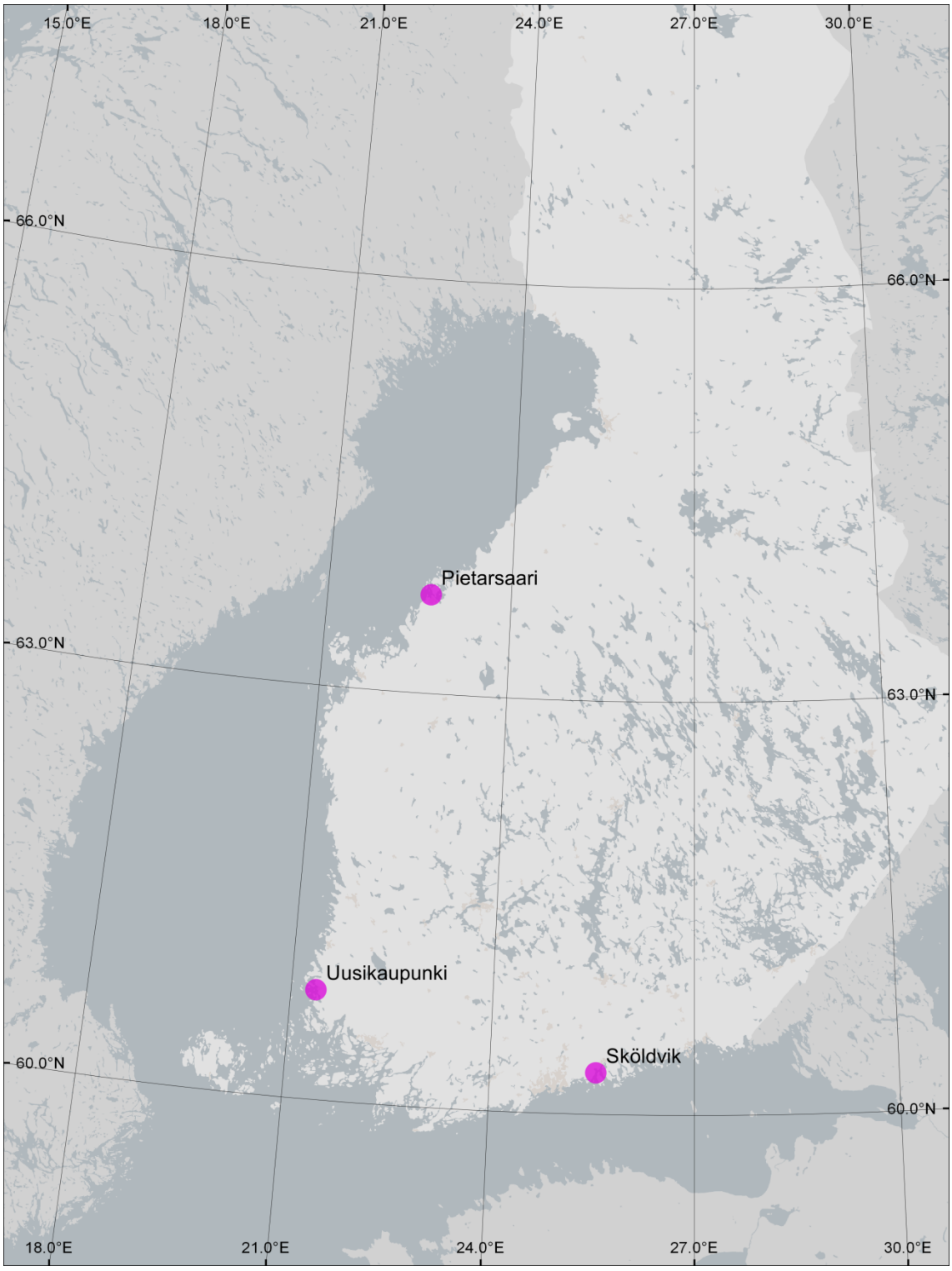
Aluksi pehmennettävän syvyysmallin solujen arvot tallennetaan 2-ulotteiseen *lähdetaulukkoon.* Lähdetaulukon lisäksi luodaan toinen, kooltaan identtinen taulukko jota voidaan kutsua *kohdetaulukoksi*. *Kohdetaulukon* kaikkien solujen arvot alustetaan esimerkiksi arvoon 10000 (metriä). Taulukoista *lähdetaulukko* sisältää ja säilyttää alkuperäisen syvyysmallin soluarvot eikä sitä muokata missään vaiheessa. *Kohdetaulukkoon* puolestaan tallennetaan pehmennyksen välivaiheet ja lopulliset tulokset.

Lähdetaulukon solut käydään yksitellen läpi ja jokaisen solun naapuruston syvyyksistä etsitään matalin syvyys (korkeuden maksimi) Z_{max}. Kun naapuruston matalin syvyys on löytynyt, verrataan *kohdetaulukon* vastaavan, lantin määrittelemän naapuruston solujen arvoja saatuun minimisyvyyteen Z_{max}. Mikäli *kohdetaulukon* solun sen hetkinen syvyys on matalampi kuin Z_{max}, päivitetään *kohdetaulukon* solun uudeksi arvoksi Z_{max}. *Kohdetaulukon* yksittäisen solun syvyysarvo voi siis muuttua prosessin edetessä useaankin otteeseen, kuitenkin aina niin, että *kohdetaulukossa* solun arvo voi ainoastaan syventyä siihen alun perin asetetusta arvosta. Koska kaikki *kohdetaulukon* solut alustettiin alun perin arvoon 10000, toimii menetelmä yleisimmillä vertaustasoilla globaalisti myös korkeuskäyrien määrittämisessä, eikä vertaustason ylitys aiheuta ongelmia.

Kun kaikki alkuperäisen syvyysmallin eli *lähdetaulukon* solut on käyty läpi, tulee vielä varmistaa alkuperäisten, tyhjää tarkoittavien *No Data* -arvojen säilyminen. Lopulta tuloksena saatu, *kohdetaulukkoon* tallennettu pehmennetty syvyysmallipinta tulostetaan tiedostoon. Menetelmän täsmällinen toteutus ilmenee tämän työn liitteenä löytyvästä sovelluksen C-kielisestä lähdekoodista (liite 1).

Rolling Coin -menetelmän käyttö aiheuttaa lähtöaineistona toimivalle syvyysmallille muutamia vaatimuksia. Jotta ”lantin” määrittämä naapurusto käyttäytyisi maastossa ennakoitavasti ja esimerkiksi tässä työssä käytetty ympyrää approksimoiva lantti kuvastuisi myös maastossa likimain ympyrän muotoisena alueena, tulee lähtöaineistona toimivan syvyysmallin olla tasoprojektiossa ja sen spatiaalisen resoluution on oltava sama sekä x- että y-akselin suuntaan. Lisäksi on huomioitava, että navigoinnin kannalta turvallinen lopputulos voidaan saavuttaa varmuudella vain siinä tapauksessa, että itse lähtöaineistona käytetty syvyysmalli on tuotettu matalia suosivalla *Shoal Bias*- tai muulla varmasti navigointiturvallisia syvyysmallipintoja tuottavalla menetelmällä.

3. TUTKIMUSALUEET



Kuva 22. Tutkimusalueiden sijainti. Taustakartta: Liikennevirasto (sisältää Maanmittauslaitoksen aineistoa lisenssillä CC BY 4.0).

Tutkimusalueiksi valikoituivat Liikennevirastossa käynnissä olevan Älyväylä-hankkeen (Liikennevirasto 2017a) osana toteutettavan syvyysmalliprojektin testialueina toimivat meriväylät Sköldvikin, Uudenkaupungin ja Pietarsaaren satamien edustalta (kuva 22). Nämä alueet valittiin tutkimusalueiksi siitä syystä, että niiden alueelta oli käytettävissä valmiita syvyysmalliaineistoja, jotka eivät olleet tiukan salassapidon piirissä.

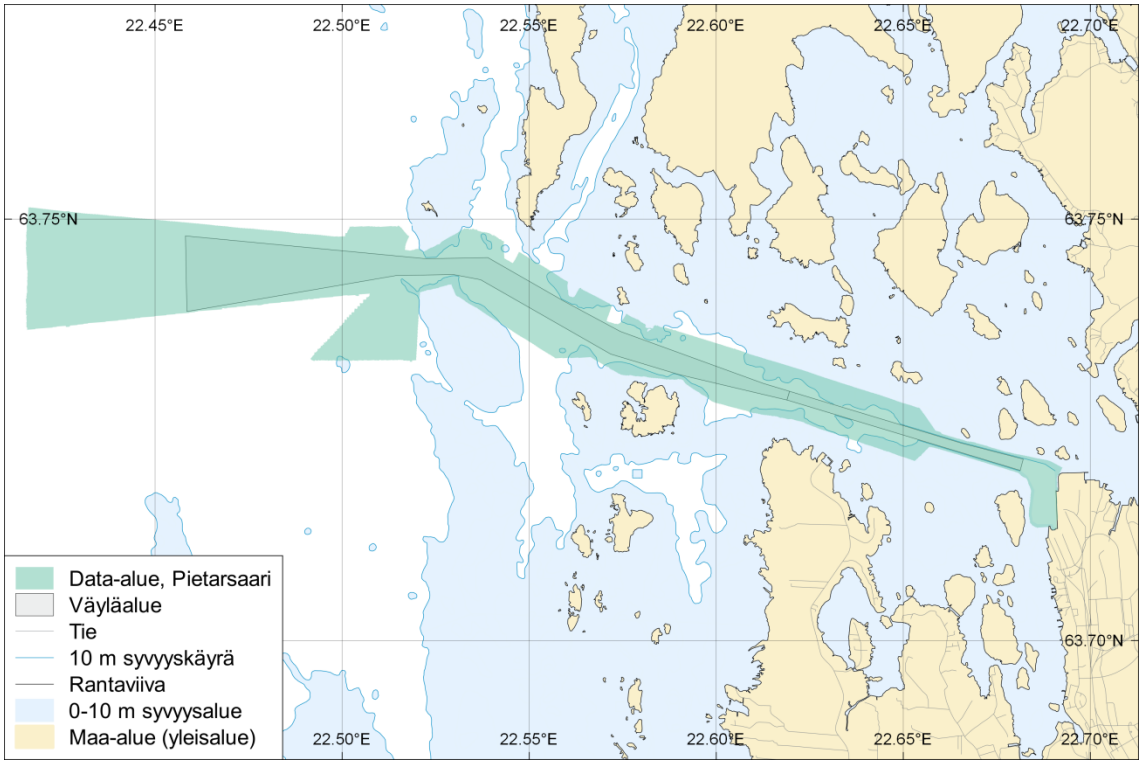
Liikenneviraston Älyväylä-hankkeen pilottiväyliin kuuluvalla Rauman meriväylällä on vuosien 2016–2017 aikana suoritettu mittavia väylän muutostöitä (Liikennevirasto 2017b), joiden yhteydessä väylän nimellistä kulkusyvyyttä on nostettu ruoppaamalla 10 metristä 12 metriin. Väylällä käynnissä olleiden ruoppaus- ja muiden muutostöiden johdosta väylällä ei olla vielä (vuoden 2018 alussa) suoritettu ruoppaustöiden jälkeistä koko väylän kattavaa merenmittausta eikä tuoretta syvyysmittausaineistoa näin ollen ole väylän ajantasaisen syvyysmallin luomiseksi ollut vielä saatavilla. Rauman meriväylän muutostyöt valmistuivat ja väylä otettiin käyttöön uudella 12 metrin nimelliskulkusyvyydellä joulukuussa 2017 (Liikennevirasto 2017b).

Ajantasaisen Rauman meriväylän syvyysaineiston puuttuessa syvyysmalliprojektissa on tuotettu varsinaisten projektiväylien (Uusikaupunki ja Sköldvik) lisäksi syvyysmalli myös Pietarsaaren meriväylältä. Pietarsaari on hyvä lisäys tutkimusalueisiin, sillä kyseessä on kahta muuta tutkimusväylää kapeampi ja useilta osin ruopattu väylä.

3.1. Pietarsaaren meriväylä

Tutkimusalueista pohjoisin on Pietarsaaren meriväylä. Väylä sijoittuu Pietarsaaren kaupungin luoteispuolelle ja johtaa kaupungin pohjoispuolella sijaitsevaan satamaan. Väyläkortin (Liikennevirasto 2017e) mukaan väylän kokonaispituus on noin 13 kilometriä. Väylän sisemmät osat, noin 3 kilometrin etäisyydelle satamasta lukien, on ruopattu kanavamaiseksi. Tällä ruopatulla, kanavamaisella osuudella väylä on myös kapeimmillaan. Väylän pienin leveys on 120 metriä.

Väylän nimellinen kulkusyvyys on 11,0 metriä ja haraustasot väylän ulko-osalla 13,1 metriä ja kapeammalla, ruopatulla sisäosalla 12,8 metriä. Satama-altaan haraussyvyydet ovat 12,3 ja 11,7 metriä. Väyläkortissa (Liikennevirasto 2017e) annetut kulku- ja haraussyvyydet on ilmoitettu vuoden 2005 keskivedestä. Pietarsaaren väylän syvyysmallin aineisto kattaa valtion väyläosuuden lisäksi myös satamalaiturin edustan (kuva 23).



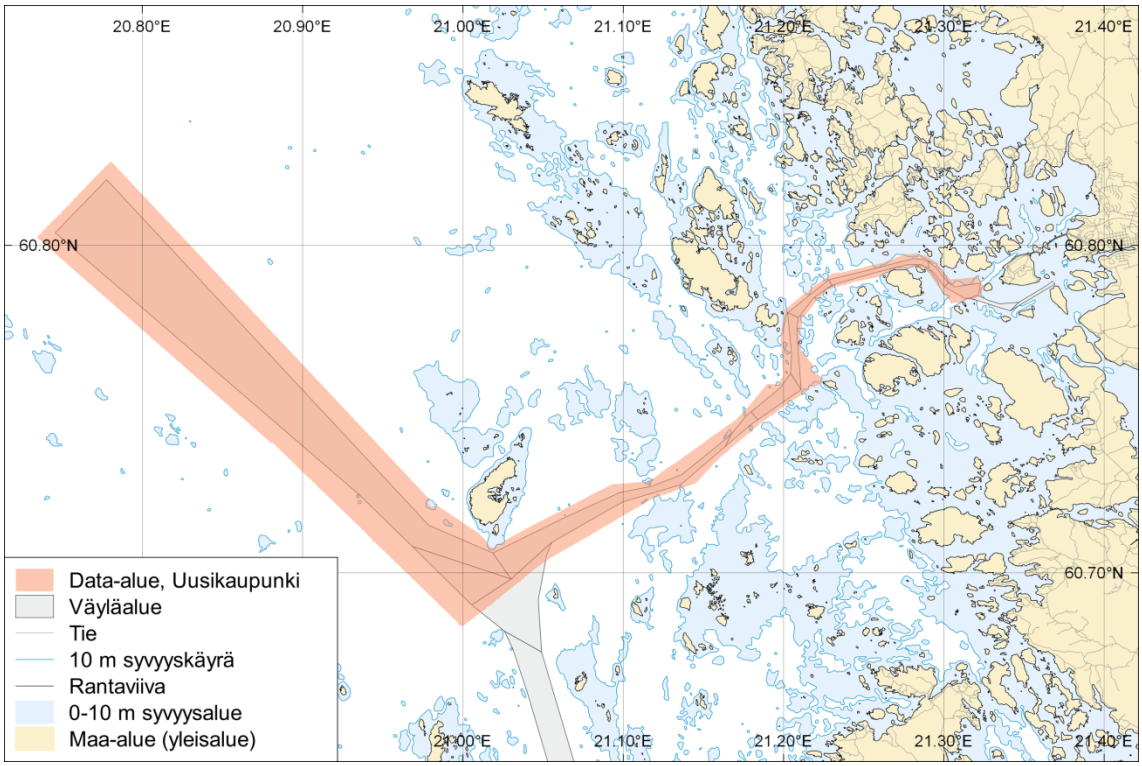
Kuva 23. Pietarsaaren syvyysmallidatan kattama alue. Aineisto ©Liikennevirasto 2018.

3.2. Uudenkaupungin meriväylä

Uudenkaupungin meriväylä sijoittuu Uudenkaupungin edustan saaristoon, jossa se kulkee Isokarin majakkasaaren eteläpuolitse ja sieltä edelleen melko jyrkästikin mutkitellen kohti Yaran satamaa. Väyläkortin (Liikennevirasto 2015) mukaan väylän kokonaispituus on noin 43 kilometriä. Väylän leveys on sen kapeimmilla osilla 140 metriä ja mutkien kaarresäde pienimmillään 900 metriä.

Väylän nimellinen kulkusyvyys on 12,5 metriä ja haraustaso väylän ulko-osalla 14,5 metriä. Väylän keskiosilla haraustaso on 13,8 metriä ja satama-altaassa 13,7 metriä.

Annetut kulku- ja haraussyvyydet on ilmoitettu vuoden 2005 keskivedestä. Uudenkaupungin väylän syvyysmallin aineisto ei toistaiseksi ulotu sataman laituriin saakka vaan kattaa ainoastaan Liikenneviraston vastuulla olevan väyläalueen (kuva 24).



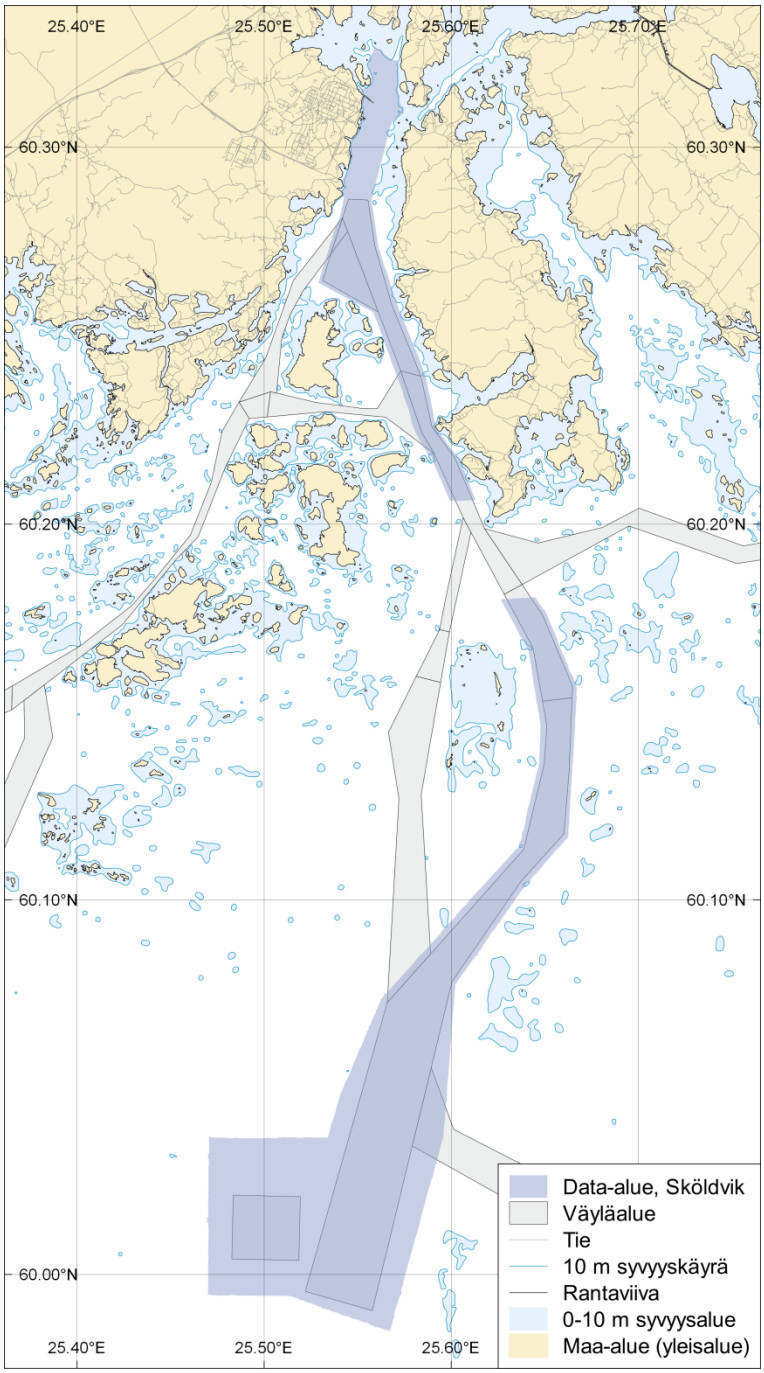
Kuva 24. Uudenkaupungin syvyysmallidatan kattama alue. Aineisto ©Liikennevirasto 2018.

3.3. Sköldvikin meriväylä

Sköldvikin (suom. Kilpilahti) meriväylä sijoittuu Porvoon Emäsalon edustalle ja johtaa sieltä Porvoon majakan itäpuolitse Sköldvikin öljysatamaan. Väyläkortin (Liikennevirasto 2011) mukaan väylän kokonaispituus on noin 38 kilometriä ja leveys kapeimmillaan 270 metriä.

Väylän nimellinen kulkusyvyys on 15,3 metriä ja haraustaso väylän ulko-osalla 17,45 metriä. Väylän sisäosilla haraustaso on 16,95 metriä. Annetut kulku- ja haraussyvyydet on ilmoitettu vuoden 2014 keskivedestä. Sköldvikin väylän syvyysmallin aineisto kattaa valtion väyläalueen lisäksi myös satamalaiturien edustan (kuva 25).

Syvyysmalliaineistossa väylän keskivaiheilla esiintyvä aukko johtuu alueelle osuvasta Puolustusvoimien suoja-alueesta. Suoja-alueiden osalta tarkkoja ja tiheitä syvyystietoja ei ole käytettävissä (Aluevalvontalaki 755/2000 § 12).



Kuva 25. Sköldvikin syvyysmallidatan kattama alue. Aineisto ©Liikennevirasto 2018.

4. AINEISTO

Tutkielman aineistona toimivat Liikenneviraston syvyysmalliprojektissa tuotetut rasterimuotoiset syvyysmallit Pietarsaaren, Uudenkaupungin ja Sköldvikin meriväyliltä. Syvyysmallit on tuotettu pistemuotoisesta merenmittausaineistosta käyttäen syvyysmalliin valittavina soluarvoina kunkin solun alueelle osuvien mittaushavaintojen matalinta arvoa eli maksimia. Tällainen maksimiarvojen valitsemista suosiva käytäntö tunnetaan merikartoitus- ja merenmittaustoiminnassa myös *Shoal Bias* -periaatteena.

Matalimpien arvojen valitsemista suosiva käytäntö takaa syvyysmallipintojen navigointiturvallisuuden, kunhan kunkin mallin solun arvon ajatellaan kuvaavan pistemäisen kohteen sijaan koko kyseisen solun aluetta – matalin pistemäinen mittaushavainto kun voi sijaita missä tahansa kohdassa kyseisen solun alueella (Peters 2012).

Taulukko 1. Työssä käytettyjen syvyysmalliaineistojen tekniset tiedot.

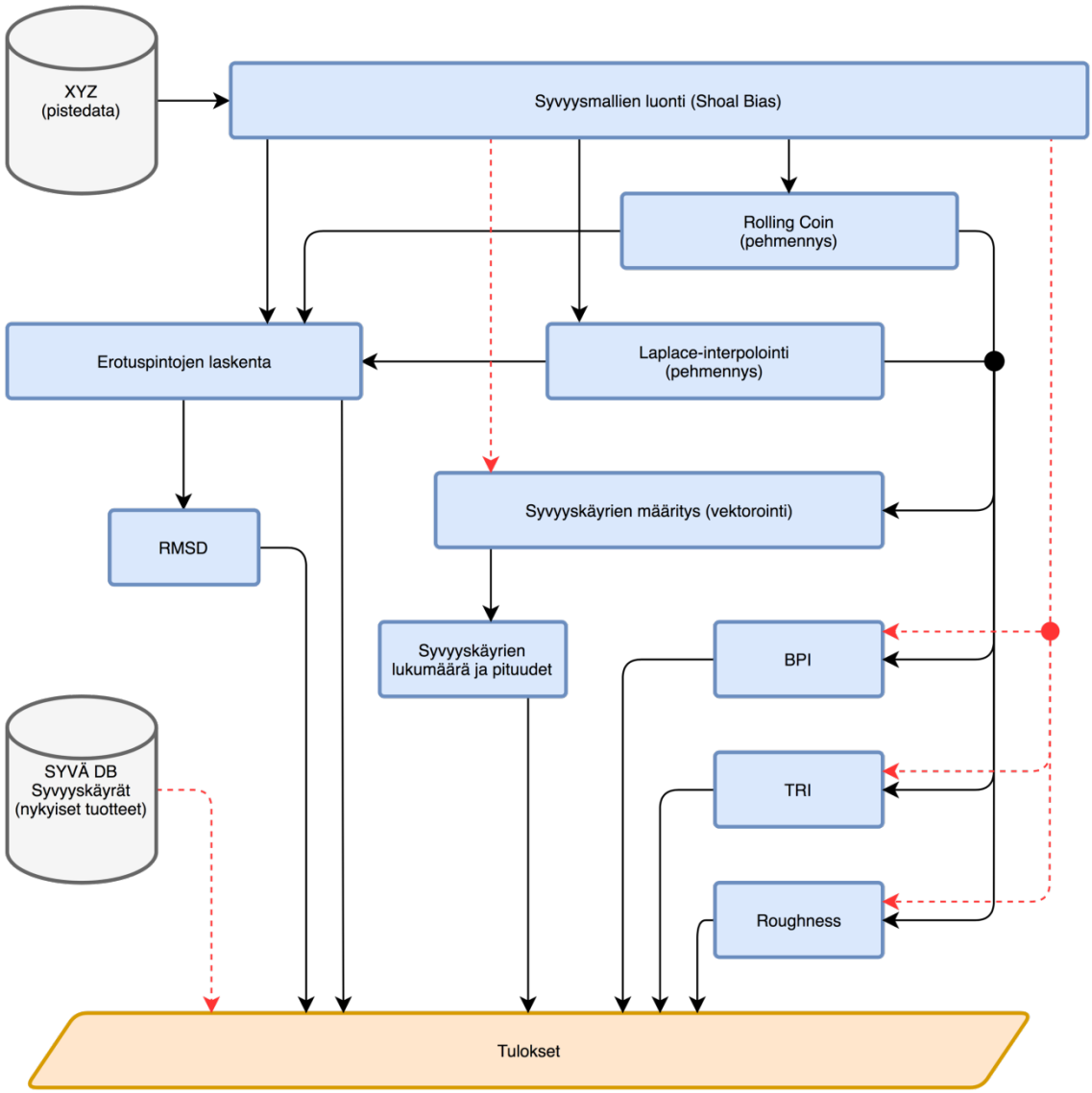
	Vertauskehys	Vertaustaso	Spat. resoluutio (x, y)	Mallinnustapa
Pietarsaari	ETRF89-UTM35N	N2000	5 m, 5 m	*Shoal Bias*
Uusikaupunki	ETRF89-UTM35N	N2000	5 m, 5 m	*Shoal Bias*
Sköldvik	ETRF89-UTM35N	N2000	5 m, 5 m	*Shoal Bias*

Syvyysmalliaineistot on tuotettu ETRS89 UTM35N -tasoprojektiossa käyttäen syvyyksien vertaustasona myöhemmin kaikille merikarttatuotteille käyttöönotettavaa N2000 -järjestelmää (Julkisen hallinnon tietohallinnon neuvottelukunta 2007; Liikennevirasto 2018c). Syvyysmallien spatiaalinen resoluutio on 5 m sekä x- että y-akselien suuntaan.

Syvyysmallit toimitettiin sekä BAG- *(Bathymetry Attributed Grid)* (Open Navigation Surface Working Group 2004) että GeoTIFF (Ritter ja Ruth 1997) -formaateissa. BAG -formaatin syvyysmallit ovat kaksikanavaisia ja GeoTIFF-formaatin syvyysmallit yksikanavaisia. Kaksikanavaisten BAG-muotoisten syvyysmallien toinen kanava on syvyyden laskennallinen epävarmuus (Open Navigation Surface Working Group 2004). Laskennallista epävarmuustietoa ei kuitenkaan tässä työssä hyödynnetä.

5. MENETELMÄT

Kuvan 26 vuokaaviossa on esitetty tässä työssä käytetyt menetelmät pääpiirteissään. Kaaviossa ylinnä näkyvä syvyysmallien tuottaminen on kuvattu työn aineistoa käsittelevässä kappaleessa 4. Koska tässä työssä käytetyt aineistot olivat valmiita syvyysmalleja, alkavat työssä varsinaisesti käytetyt menetelmät valmiiden syvyysmallipintojen käsittelyyn käytetyistä menetelmistä (*Laplace-interpolointi* ja *Rolling Coin*). Nämä syvyysmallipintojen pehmennykseen käytetyt menetelmät on kuvattu tarkemmin kappaleissa 2.3. ja 2.4.



Kuva 26. Työssä käytetyt menetelmät yksinkertaistettuna vuokaaviona. Punaiset katkoviivat kuvaavat vertailuaineistojen virtoja. SYVÄ-tietokanta on merikartoituksen syvyystiedon tuotantotietokanta.

Kuvan 26 kaavioon on merkitty pehmennettyjen syvyysmallipintojen käsittely- ja analysointimenetelmien lisäksi myös vertailukäytössä toimivien aineistojen virrat ja käsittelyvaiheet. Vertailuaineistojen virrat on merkitty kaavioon punaisin katkoviivoin.

Työn vertailuaineistoina käytettiin kappaleessa 4 kuvattuja alkuperäisiä, pehmentämättömiä syvyysmallipintoja sekä merikartoituksen syvyystietojen tuotantotietokannasta ("Syvä DB") saatuja, kunkin tutkimusalueen syvyysmallien aineiston kattamalle alueelle (kuvat 23, 24 ja 25) osuvia syvyyskäyriä. Nämä nykyisille merikarttatuotteille sisältyvät syvyyskäyrät on tuotettu nykyisen syvyystietojen uusimisprosessin (Liikennevirasto 2017a) mukaisesti manuaalisesti digitoimalla.

Syvyysmallipintojen ja niistä johdettujen (vektorimuotoisten) syvyyskäyrien käsittelyn ja analysoinnin menetelmät on koottu seuraaviin kappaleisiin.

5.1. Syvyyskäyrien automaattinen määritys (syvyysmallipintojen vektorointi)

Syvyysmallipinnoista automaattisesti luodut vektorimuotoiset syvyyskäyrät määritettiin GDAL-kirjastoon (GDAL Development Team 2018) sisältyvää syvyys- ja korkeuskäyrien määritykseen tarkoitettua gdal_contour -sovellusta (GDAL 2018) käyttämällä.

5.2. Erotuspinnat

Syvyysmallipintojen pehmentämisen tuloksien arvioinnissa keskeinen menetelmä on erotuspintojen laskenta ja tarkastelu. Pehmennettyjen syvyysmallipintojen turvallisuuden selvittämisen lisäksi erotuspinnoista saadaan määritettyä myös pehmennettyjen ja alkuperäisten syvyysmallien väliset keskivirheet (-poikkeamat). Erotuspinta D saadaan vähentämällä pehmennetystä syvyysmallipinnasta S alkuperäinen syvyysmallipinta Z:

$$D = S - Z$$

5.3. Root Mean Square Difference, RMSD

RMSD *(Root Mean Square Difference)* eli *keskimääräisen neliöpoikkeaman neliöjuuri* on yleisesti käytetty mallinnusvirhettä tai -poikkeamaa kuvaava tunnusluku (Hyndman ja Koehler 2006). Syvyysmallipintojen pehmennyksen tuloksia tarkasteltaessa RMSD kertoo alkuperäisen ja pehmennetyn syvyysmallipinnan välisestä keskimääräisestä erosta. RMSD lasketaan kaavalla

$$RMSD_Z = \sqrt{\frac{\sum_{i=1}^{n}(Z_i - \bar{Z}_i)^2}{n}}$$

jossa n on syvyysmallin (varsinaisen syvyystiedon sisältävien) solujen lukumäärä, Z_i on pehmennetyn syvyysmallipinnan arvo solussa i ja $\bar{Z}_i$ syvyyden odotusarvo eli alkuperäisen syvyysmallin syvyysarvo solussa i. Ne syvyysmallin solut, joiden soluarvo on *No Data*, tulee jättää laskelmasta pois. Mikäli näin ei tehdä, on kaavan nimittäjä n usein liian suuri ja lopputuloksena saatu RMSD vastaavasti liian pieni.

5.4. Bathymetric Position Index, BPI

BPI *(Bathymetric Position Index)* on rasterimuotoisen syvyysmallipinnan solujen suhteellista syvyyttä kuvaava indeksi. BPI-indeksi on johdettu Weiss'n (2001) kehittämän TPI *(Topographic Position Index)* -indeksin pohjalta merenpohjan paikallista korkeusvaihtelua kuvaavaksi indeksiksi (Lundblad et al. 2006; Lecours et al. 2016). BPI saa positiivisen arvon, mikäli tarkasteltava solu on naapurustoaan matalampi ja negatiivisen, mikäli tarkasteltava solu on naapurustoaan syvempi (Weiss 2001, Lundblad et al. 2006, Wilson et al. 2007).

BPI on määritelmänsä (Weiss 2001) mukaan tarkasteltavan solun ja sen naapuruston soluarvojen keskiarvon erotus. BPI:n laskenta ei ole sidottu vain tarkasteltavan solun välittömään naapurustoon, vaan haluttaessa sen laskennassa voidaan käyttää useita eri tyyppisiä naapurustoja (Lundblad et al. 2006, Wilson et al. 2007). Tässä työssä BPI:n

laskennassa on käytetty ainoastaan tarkasteltavan solun välittömän naapuruston käsittävää 3 × 3 solun naapurustomatriisia.

BPI:n laskentakaava voidaan määritellä yleisesti koskemaan eri kokoisia naapurustomatriiseja. Jos naapurustomatriisin koko on $n \times n$, saadaan TRI laskettua Wilson et al. (2007) mukaillen seuraavasti:

$$k = \frac{(n-1)}{2}$$

$$BPI_{(x,y)} = Z_{(x,y)} - \frac{\left(\left(\sum_{i=-k}^{k}\sum_{j=-k}^{k} Z_{(x-i,y-j)}\right) - Z_{(x,y)}\right)}{(n^2-1)}$$

5.5. Terrain Ruggedness Index, TRI

Terrain Ruggedness Index (Riley et al. 1999) on rasterimuotoisista korkeusmalliaineistoista laskettava maaston rosoisuutta *(Terrain Heterogeneity)* kuvaava indeksi. Syvyysmallipintojen yhteydessä indeksin ottivat käyttöön Valentine et al. (2004) tutkiessaan merenpohjan geomorfologiaa.

TRI on määritelmän mukaisesti tutkittavan solun ja sen naapuruston soluarvojen välisten erotusten itseisarvojen keskiarvo (Riley et al. 1999, Valentine et al. 2004, Wilson et al. 2007). Useimmiten TRI:n laskennassa käytetään kooltaan 3 × 3 solun naapurustomatriisia, mutta myös muun kokoisia naapurustoja voidaan käyttää. Tässä työssä TRI:n laskennassa naapurustona on käytetty indeksin alkuperäisen määritelmän (Riley et al. 1999) mukaista 3 × 3 solun naapurustomatriisia.

TRI:n laskentakaava voidaan kuitenkin BPI:n tapaan määritellä yleisesti koskemaan eri kokoisia naapurustomatriiseja. Jos naapurustomatriisin koko on $n \times n$, saadaan TRI laskettua Wilson et al. (2007) mukaillen seuraavasti:

$$k = \frac{(n-1)}{2}$$

$$TRI_{(x,y)} = \frac{\sum_{i=-k}^{k} \sum_{j=-k}^{k} \left| Z_{(x-i,y-j)} - Z_{(x,y)} \right|}{(n^2 - 1)}$$

5.6. Roughness Index

Wilson et al. (2007) määritelmän mukaan *Roughness* kuvaa syvyysmallipinnan syvyyksien vaihteluväliä tutkittavan solun ja sen naapuruston kattamalla alueella. Tässä työssä Roughness-indeksin laskennassa on käytetty 3×3 solun naapurustomatriisia W. Tällöin *Roughness* on naapurustomatriisin W käsittämien syvyysmallin solujen muodostaman joukon D maksimi- ja minimiarvojen erotus:

$$Roughness = \max D - \min D$$

5.7. Syvyyskäyrien lukumäärä ja käyrien pituuksien jakauma

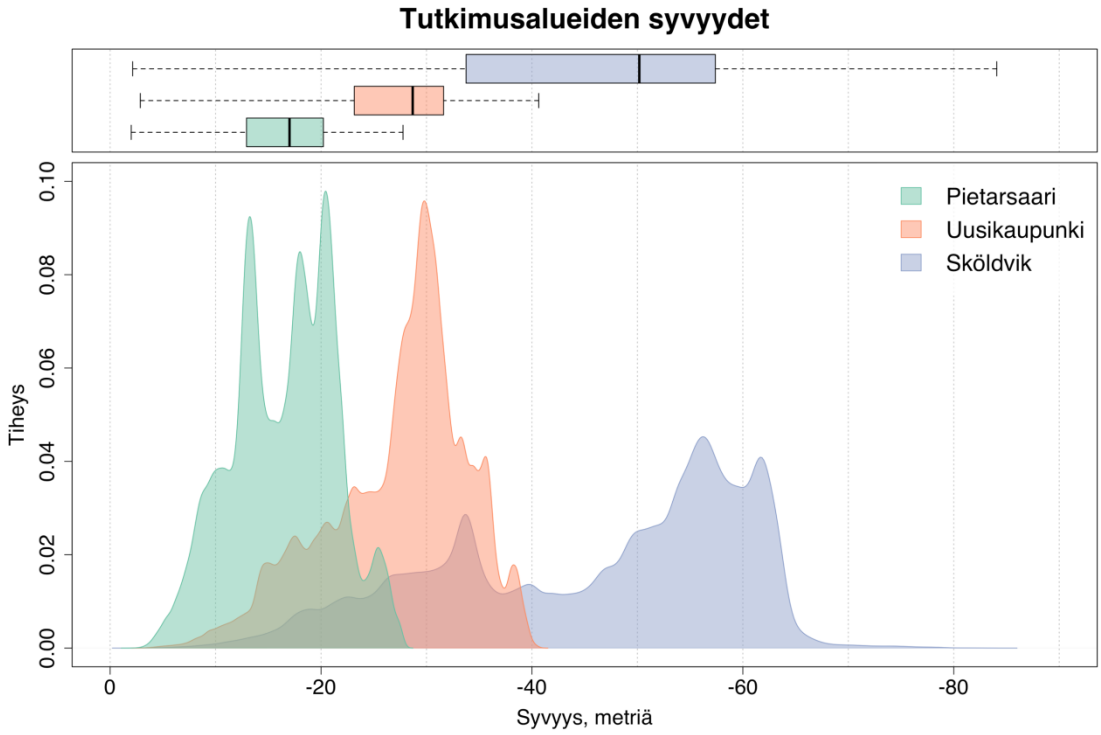
Merikarttatuotannon näkökulmasta syvyysmallipintojen pehmennyksen päätavoite on mahdollistaa teknisesti ja kartografisesti mahdollisimman laadukkaiden syvyyskäyrien määritys automaattisin menetelmin suoraan syvyysmallipinnoista. Tähän tavoitteeseen perustuen yhtenä syvyysmallipintojen pehmentämisen tulosten arviointimenetelmänä on Liikenneviraston tekemissä testeissä (Liikennevirasto 2017g) käytetty pehmennetyistä syvyysmallipinnoista automaattisesti määritettyjen syvyyskäyrien lukumäärää ja pituuksien jakaumaa.

Syvyyskäyrien lukumäärään ja pituuksien jakaumaan nojaava laadunarviointimenetelmä perustuu automaattisen käyränmäärityksen ominaisuuteen tuottaa syvyyskäyrä jokaisen käyrärajan puhkaisevan solun tai solujoukon noodien ympärille. Syvyyskäyrien vähäisempi lukumäärä ja suurempi keskipituus viittaavat näin ollen myös tasaisempiin syvyysmallipintoihin. Pehmennetyistä syvyysmallipinnoista automaattisin menetelmin määritettyjen syvyyskäyrien lukumääriä ja pituuksien jakaumia on vertailtu merikarttatuotteilla nykyisin esitettyihin, manuaalisin menetelmin tuotettuihin syvyyskäyriin.

6. TULOKSET

6.1. Tutkimusalueiden syvyydet

Tutkimusalueiden syvyyksien jakauma ja tilastolliset tunnusluvut määritettiin alkuperäisistä, pehmentämättömistä syvyysmallipinnoista. Kuvassa 27 esitetyt tutkimusalueiden syvyyksien jakaumat eroavat toisistaan huomattavasti. Sköldvik on tutkimusalueista selvästi syvin. Syvyyksien tilastolliset tunnusluvut on esitetty myös taulukossa 2.



Kuva 27. Tutkimusalueiden syvyydet esitettynä yhdistetyn ydintiheyskuvaajan ja laatikkokuvaajan keinoin. Tutkimusalueiden syvyysprofiilit eroavat toisistaan huomattavasti.

Taulukko 2. Tutkimusalueiden syvyyksien tunnuslukuja. Arvot ovat metrejä N2000 -vertaustasosta.

	Minimi	Alakvartiili	Mediaani	Keskiarvo	Yläkvartiili	Maksimi
Pietarsaari	-27,77	-20,22	-17,01	-16,46	-12,93	-2,00
Uusikaupunki	-40,64	-31,62	-28,69	-27,27	-23,15	-2,86
Sköldvik	-84,07	-57,39	-50,18	-45,90	-33,75	-2,14

6.2. Erotuspinnat

Erotuspinnat laskettiin pehmennetyistä syvyysmallipinnoista vähentämällä niistä alkuperäinen, pehmentämätön syvyysmallipinta. Erotuspintojen tilastolliset tunnusluvut on esitetty tutkimusalueittain taulukoissa 3, 4 ja 5. Erotuspintoja kuvaaviin taulukoihin on lisäksi merkitty kunkin pehmennetyn syvyysmallipinnan turvallisuus navigoinnin näkökulmasta. Syvyysmallipinta on navigoinnin kannalta turvallinen, mikäli kyseisen pinnan erotuspinnan minimiarvo(-t) eivät ole negatiivisia.

Taulukko 3. Pietarsaaren erotuspintojen tunnusluvut ja käytettyjen menetelmien turvallisuus.

	Minimi	Mediaani	Keskiarvo	Maksimi	Keskihajonta	Turvallinen
Laplace (1x)	0,00	0,00	0,03	3,30	0,07	✅
Laplace (2x)	0,00	0,02	0,05	3,42	0,09	✅
Laplace (5x)	0,00	0,04	0,08	4,22	0,14	✅
Laplace (10x)	0,00	0,06	0,12	4,60	0,18	✅
Laplace (20x)	0,00	0,10	0,18	5,08	0,24	✅
Laplace (50x)	0,00	0,16	0,27	5,55	0,35	✅
Laplace (100x)	0,00	0,24	0,38	6,22	0,45	✅
Rolling Coin (R=5)	0,00	0,04	0,12	5,38	0,19	✅
Rolling Coin (R=10)	0,00	0,10	0,21	6,86	0,28	✅

Taulukko 4. Uudenkaupungin erotuspintojen tunnusluvut ja käytettyjen menetelmien turvallisuus.

	Minimi	Mediaani	Keskiarvo	Maksimi	Keskihajonta	Turvallinen
Laplace (1x)	0,00	0,00	0,04	3,27	0,08	✅
Laplace (2x)	0,00	0,02	0,06	3,48	0,10	✅
Laplace (5x)	0,00	0,04	0,10	4,57	0,16	✅
Laplace (10x)	0,00	0,07	0,15	5,61	0,21	✅
Laplace (20x)	0,00	0,12	0,23	6,56	0,28	✅
Laplace (50x)	0,00	0,23	0,36	7,21	0,41	✅
Laplace (100x)	0,00	0,35	0,51	8,35	0,53	✅
Rolling Coin (R=5)	0,00	0,04	0,16	5,93	0,26	✅
Rolling Coin (R=10)	0,00	0,18	0,37	7,38	0,49	✅

Taulukko 5. Sköldvikin erotuspintojen tunnusluvut ja käytettyjen menetelmien turvallisuus.

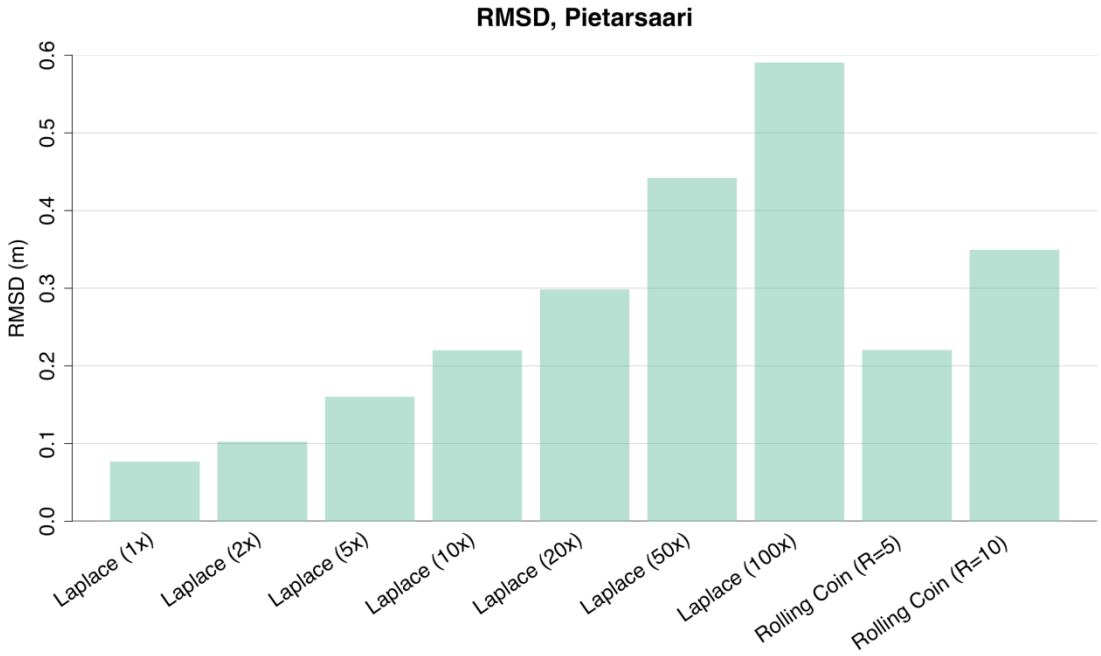
	Minimi	Mediaani	Keskiarvo	Maksimi	Keskihajonta	Turvallinen
Laplace (1x)	0,00	0,00	0,03	7,76	0,09	✅
Laplace (2x)	0,00	0,02	0,05	7,76	0,12	✅
Laplace (5x)	0,00	0,04	0,09	7,76	0,18	✅
Laplace (10x)	0,00	0,06	0,14	7,76	0,26	✅
Laplace (20x)	0,00	0,09	0,21	8,58	0,36	✅
Laplace (50x)	0,00	0,14	0,36	9,54	0,55	✅
Laplace (100x)	0,00	0,20	0,53	10,71	0,77	✅
Rolling Coin (R=5)	0,00	0,02	0,11	7,82	0,26	✅
Rolling Coin (R=10)	0,00	0,07	0,28	11,97	0,59	✅

6.3. Root Mean Square Difference, RMSD

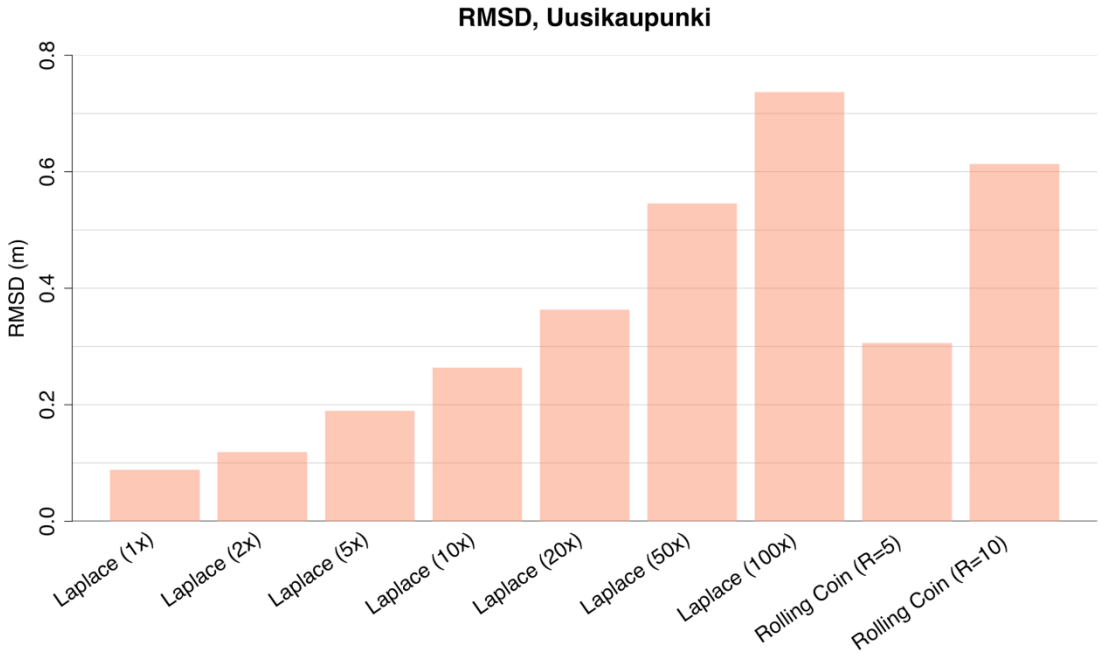
RMSD *(Root Mean Square Difference)* eli *keskimääräisen neliöpoikkeaman neliöjuuri* laskettiin erotuspintojen (kappale 6.2) pohjalta kappaleessa 5.4.5 kuvatulla tavalla. Eri menetelmin pehmennettyjen syvyysmallipintojen RMS-poikkeamat on esitetty taulukon 6 lisäksi pylväskuvaajina tutkimusalueittain kuvissa 28, 29 ja 30.

Taulukko 6. Erotuspinnoista lasketut RMS-poikkeamat tutkimusalueittain. Arvot ovat metrejä.

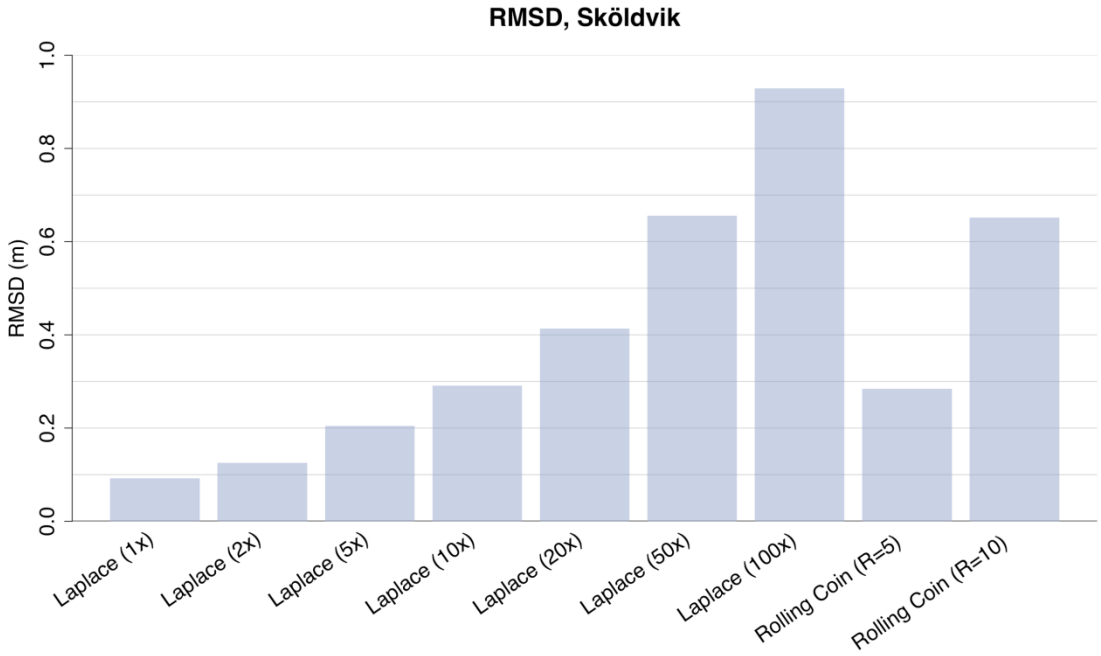
	Pietarsaari	Uusikaupunki	Sköldvik
Laplace (1x)	0,077	0,088	0,092
Laplace (2x)	0,102	0,119	0,125
Laplace (5x)	0,161	0,190	0,205
Laplace (10x)	0,220	0,264	0,291
Laplace (20x)	0,299	0,363	0,413
Laplace (50x)	0,442	0,546	0,656
Laplace (100x)	0,590	0,737	0,929
Rolling Coin (R=5)	0,220	0,306	0,284
Rolling Coin (R=10)	0,349	0,613	0,652



Kuva 28. Pehmennettyjen syvyysmallipintojen RMSD-arvot, Pietarsaaren tutkimusalue.



Kuva 29. Pehmennettyjen syvyysmallipintojen RMSD-arvot, Uudenkaupungin tutkimusalue.



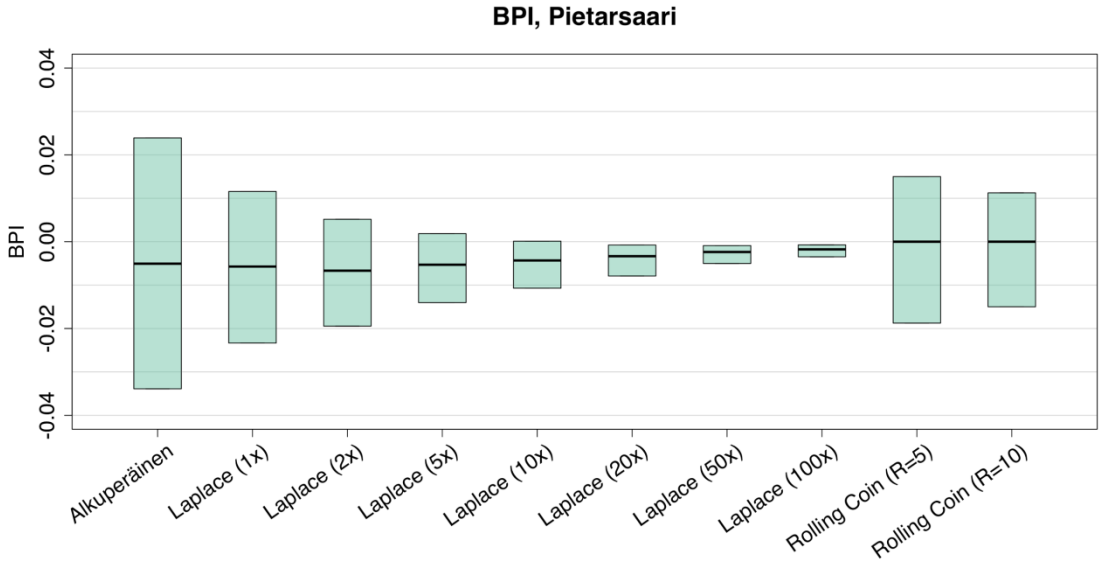
Kuva 30. Pehmennettyjen syvyysmallipintojen RMSD-arvot, Sköldvikin tutkimusalue.

6.4. Bathymetric Position Index, BPI

BPI eli *Bathymetric Position Index* laskettiin kullekin syvyysmallipinnalle kappaleessa 5.4.2 kuvatulla tavalla. Indeksin laskennassa naapurustona käytettiin 3×3 solun naapurustomatriisia. BPI-indeksin laskennan tuottamien pintojen tilastolliset tunnusluvut on esitetty tutkimusalueittain sekä taulukoin että kuvaajin. Kuvissa 31, 32 ja 33 esitetyistä laatikkokuvaajista on poistettu ylä- ja alakvartiilien ulkopuoliset havainnot kuvaajien luettavuuden parantamiseksi. Tulokset on kuitenkin koottu kokonaisuudessaan taulukoihin 7, 8 ja 9.

Taulukko 7. Pietarsaaren tutkimusalueen syvyysmallipinnoista laskettujen BPI-arvojen tunnusluvut.

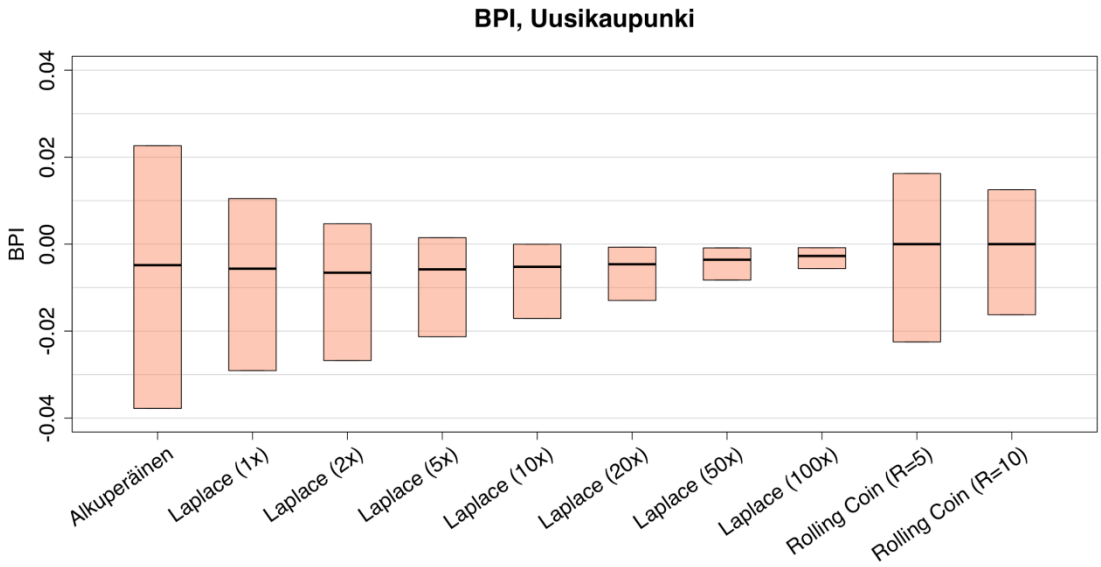
	Minimi	Alakvartiili	Mediaani	Yläkvartiili	Maksimi
Alkuperäinen	-2,829	-0,034	-0,005	0,024	5,589
Laplace (1x)	-1,608	-0,023	-0,006	0,012	4,816
Laplace (2x)	-0,928	-0,019	-0,007	0,005	4,479
Laplace (5x)	-0,515	-0,014	-0,005	0,002	3,963
Laplace (10x)	-0,332	-0,011	-0,004	0,000	3,578
Laplace (20x)	-0,253	-0,008	-0,003	-0,001	3,184
Laplace (50x)	-0,174	-0,005	-0,002	-0,001	2,625
Laplace (100x)	-0,134	-0,003	-0,002	-0,001	2,232
Rolling Coin (R=5)	-2,381	-0,019	0,000	0,015	5,248
Rolling Coin (R=10)	-2,499	-0,015	0,000	0,011	4,819



Kuva 31. Pietarsaaren tutkimusalueen BPI-arvojen jakaumat laatikkokuvaajina. Ylä- ja alakvartiilien ulkopuoliset arvot on jätetty kuvaajasta pois luettavuussyistä. Puuttuvat tiedot löytyvät taulukosta 7.

Taulukko 8. Uudenkaupungin tutkimusalueen syvyysmallipinnoista laskettujen BPI-arvojen tunnusluvut.

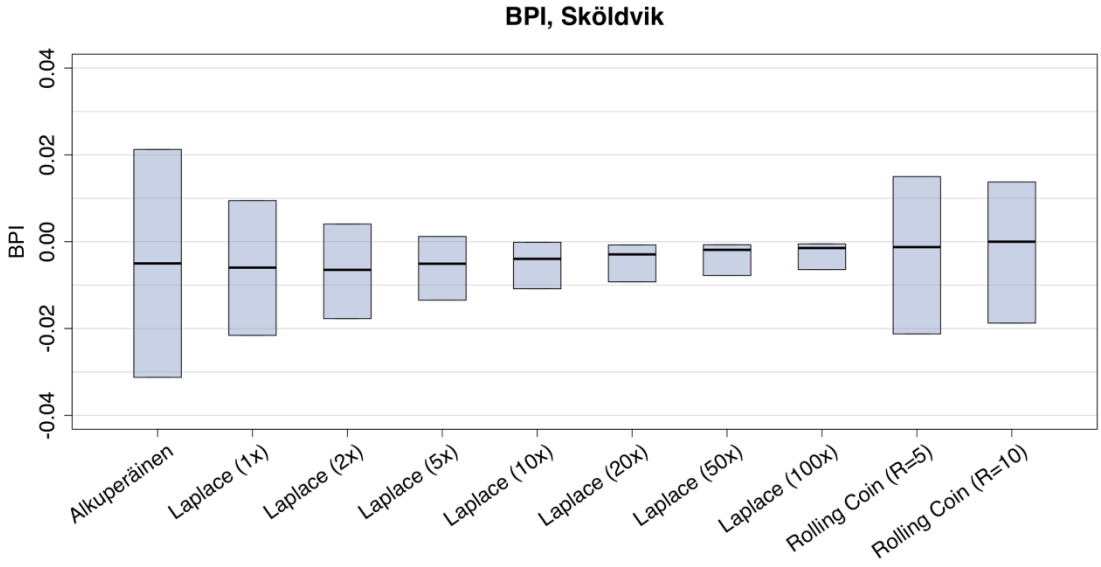
	Minimi	Alakvartiili	Mediaani	Yläkvartiili	Maksimi
Alkuperäinen	-2,833	-0,038	-0,005	0,023	4,808
Laplace (1x)	-1,699	-0,029	-0,006	0,010	4,201
Laplace (2x)	-0,935	-0,027	-0,007	0,005	3,938
Laplace (5x)	-0,639	-0,021	-0,006	0,001	3,579
Laplace (10x)	-0,340	-0,017	-0,005	0,000	3,307
Laplace (20x)	-0,208	-0,013	-0,005	-0,001	3,064
Laplace (50x)	-0,178	-0,008	-0,004	-0,001	2,809
Laplace (100x)	-0,174	-0,006	-0,003	-0,001	2,705
Rolling Coin (R=5)	-2,572	-0,023	0,000	0,016	4,659
Rolling Coin (R=10)	-2,641	-0,016	0,000	0,012	4,466



Kuva 32. Uudenkaupungin tutkimusalueen BPI-arvojen jakaumat laatikkokuvaajina. Ylä- ja alakvartiilien ulkopuoliset arvot on jätetty kuvaajasta pois luettavuussyistä. Puuttuvat tiedot löytyvät taulukosta 8.

Taulukko 9. Sköldvikin tutkimusalueen syvyysmallipinnoista laskettujen BPI-arvojen tunnusluvut.

	Minimi	Alakvartiili	Mediaani	Yläkvartiili	Maksimi
Alkuperäinen	-7,365	-0,031	-0,005	0,021	7,312
Laplace (1x)	-2,908	-0,022	-0,006	0,009	6,341
Laplace (2x)	-1,309	-0,018	-0,006	0,004	5,877
Laplace (5x)	-0,989	-0,013	-0,005	0,001	5,159
Laplace (10x)	-0,557	-0,011	-0,004	0,000	4,525
Laplace (20x)	-0,330	-0,009	-0,003	-0,001	3,821
Laplace (50x)	-0,195	-0,008	-0,002	-0,001	2,792
Laplace (100x)	-0,185	-0,006	-0,001	-0,001	2,609
Rolling Coin (R=5)	-4,934	-0,021	-0,001	0,015	5,746
Rolling Coin (R=10)	-4,934	-0,019	0,000	0,014	5,746



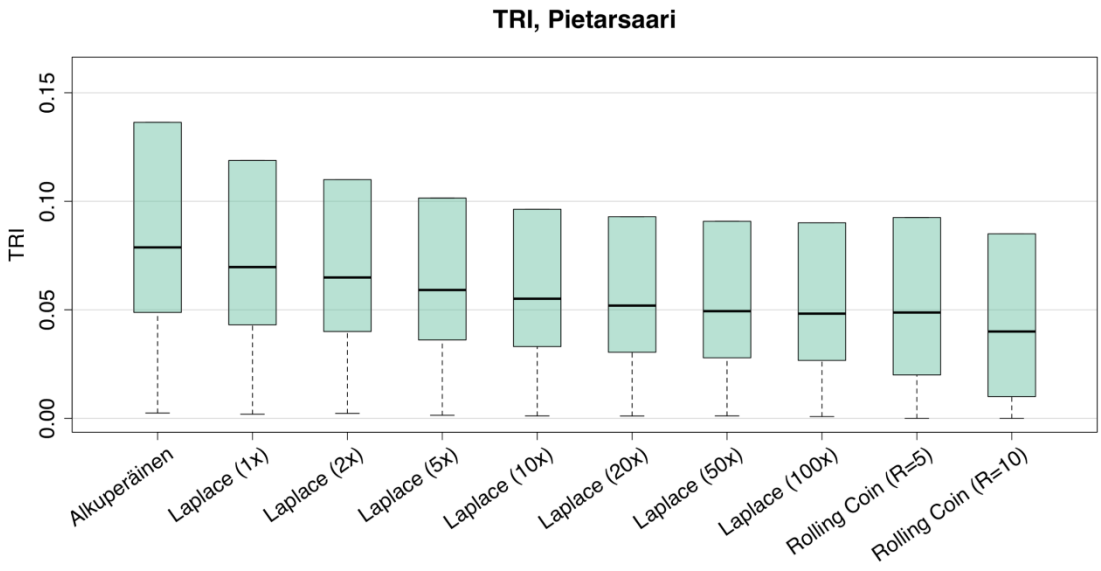
Kuva 33. Sköldvikin tutkimusalueen BPI-arvojen jakaumat laatikkokuvaajina. Ylä- ja alakvartiilien ulkopuoliset arvot on jätetty kuvaajasta pois luettavuussyistä. Puuttuvat tiedot löytyvät taulukosta 9.

6.5. Terrain Ruggedness Index, TRI

TRI eli *Terrain Ruggedness Index* laskettiin kullekin syvyysmallipinnalle kappaleessa 5.4.3 kuvatulla tavalla. Indeksin laskennassa naapurustona käytettiin 3 × 3 solun naapurustomatriisia. TRI-pintojen tilastolliset tunnusluvut on esitetty tutkimusalueittain sekä taulukoin että kuvaajin. Kuvissa 34, 35 ja 36 esitetyistä laatikkokuvaajista on poistettu yläkvartiilien ulkopuoliset havainnot kuvaajien luettavuuden parantamiseksi. Tulokset on kuitenkin koottu kokonaisuudessaan taulukoihin 10, 11 ja 12.

Taulukko 10. Pietarsaaren tutkimusalueen syvyysmallipinnoista laskettujen TRI-arvojen tunnusluvut.

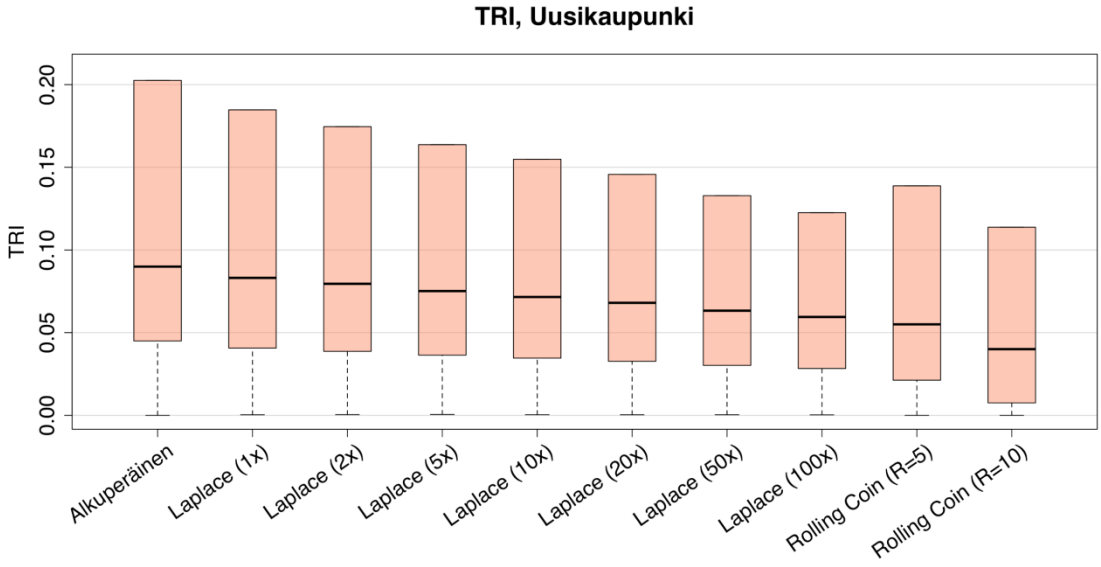
	Minimi	Alakvartiili	Mediaani	Yläkvartiili	Maksimi
Alkuperäinen	0,002	0,049	0,079	0,136	5,589
Laplace (1x)	0,002	0,043	0,070	0,119	4,816
Laplace (2x)	0,002	0,040	0,065	0,110	4,479
Laplace (5x)	0,001	0,036	0,059	0,101	3,963
Laplace (10x)	0,001	0,033	0,055	0,096	3,578
Laplace (20x)	0,001	0,030	0,052	0,093	3,184
Laplace (50x)	0,001	0,028	0,049	0,091	2,625
Laplace (100x)	0,001	0,027	0,048	0,090	2,232
Rolling Coin (R=5)	0,000	0,020	0,049	0,093	5,248
Rolling Coin (R=10)	0,000	0,010	0,040	0,085	4,819



Kuva 34. Pietarsaaren tutkimusalueen TRI-arvojen jakaumat laatikkokuvaajina. Yläkvartiilia suuremmat arvot on jätetty kuvaajasta pois luettavuussyistä. Puuttuvat maksimiarvot löytyvät taulukosta 10.

Taulukko 11. Uudenkaupungin tutkimusalueen syvyysmallipinnoista laskettujen TRI-arvojen tunnusluvut.

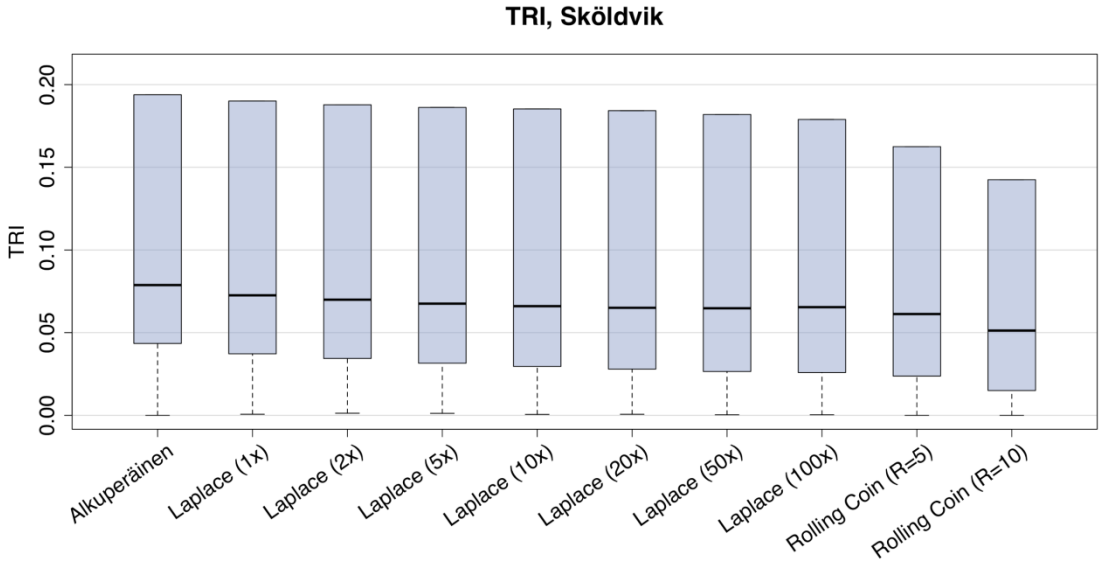
	Minimi	Alakvartiili	Mediaani	Yläkvartiili	Maksimi
Alkuperäinen	0,000	0,045	0,090	0,203	5,025
Laplace (1x)	0,000	0,041	0,083	0,185	4,201
Laplace (2x)	0,000	0,039	0,080	0,175	3,938
Laplace (5x)	0,000	0,036	0,075	0,164	3,579
Laplace (10x)	0,000	0,035	0,072	0,155	3,307
Laplace (20x)	0,000	0,033	0,068	0,146	3,064
Laplace (50x)	0,000	0,030	0,063	0,133	2,809
Laplace (100x)	0,000	0,028	0,059	0,123	2,705
Rolling Coin (R=5)	0,000	0,021	0,055	0,139	5,025
Rolling Coin (R=10)	0,000	0,008	0,040	0,114	4,909



Kuva 35. Uudenkaupungin tutkimusalueen TRI-arvojen jakaumat laatikkokuvaajina. Yläkvartiilia suuremmat arvot on jätetty kuvaajasta pois luettavuussyistä. Puuttuvat maksimiarvot löytyvät taulukosta 11.

Taulukko 12. Sköldvikin tutkimusalueen syvyysmallipinnoista laskettujen TRI-arvojen tunnusluvut.

	Minimi	Alakvartiili	Mediaani	Yläkvartiili	Maksimi
Alkuperäinen	0,000	0,043	0,079	0,194	7,365
Laplace (1x)	0,001	0,037	0,073	0,190	6,341
Laplace (2x)	0,001	0,034	0,070	0,188	5,877
Laplace (5x)	0,001	0,032	0,068	0,186	5,159
Laplace (10x)	0,001	0,030	0,066	0,185	4,525
Laplace (20x)	0,001	0,028	0,065	0,184	3,821
Laplace (50x)	0,000	0,027	0,065	0,182	3,067
Laplace (100x)	0,000	0,026	0,065	0,179	2,641
Rolling Coin (R=5)	0,000	0,024	0,061	0,163	6,834
Rolling Coin (R=10)	0,000	0,015	0,051	0,143	6,834



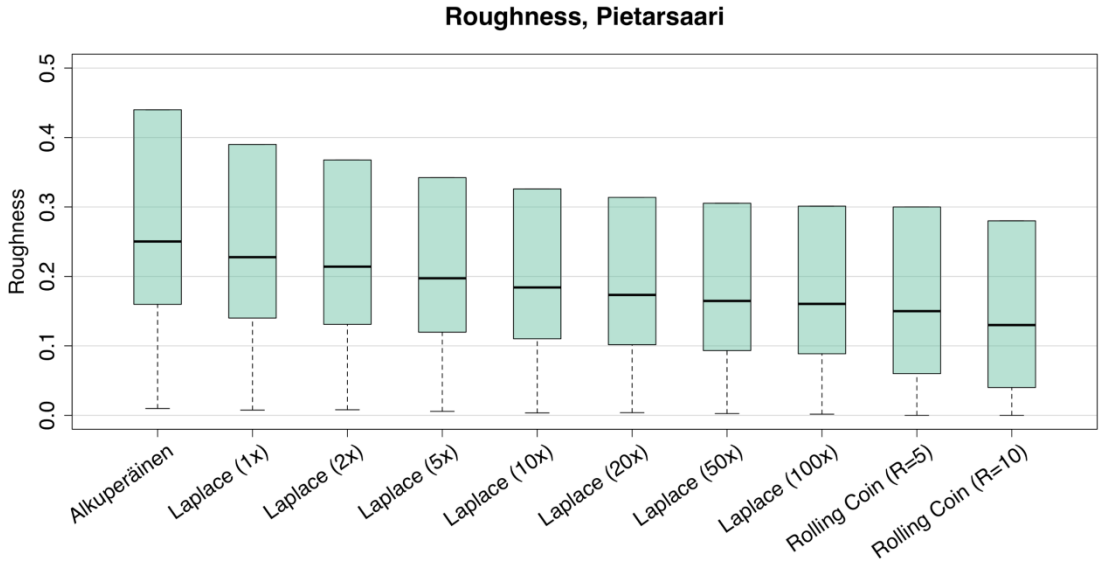
Kuva 36. Sköldvikin tutkimusalueen TRI-arvojen jakaumat laatikkokuvaajina. Yläkvartiilia suuremmat arvot on jätetty kuvaajasta pois luettavuussyistä. Puuttuvat maksimiarvot löytyvät taulukosta 12.

6.6. Roughness

Roughness laskettiin kullekin syvyysmallipinnalle kappaleessa 5.4.4 kuvatulla tavalla. Indeksin laskennassa naapurustona käytettiin 3×3 solun naapurustomatriisia. Roughness-pintojen tilastolliset tunnusluvut on esitetty tutkimusalueittain sekä taulukoin että kuvaajin. Kuvissa 37, 38 ja 39 esitetyistä laatikkokuvaajista on poistettu yläkvartiilien ulkopuoliset havainnot kuvaajien luettavuuden parantamiseksi. Tulokset on kuitenkin koottu kokonaisuudessaan taulukoihin 13, 14 ja 15.

Taulukko 13. Pietarsaaren tutkimusalueen Roughness-arvojen tunnusluvut. Lukemat ovat metrejä.

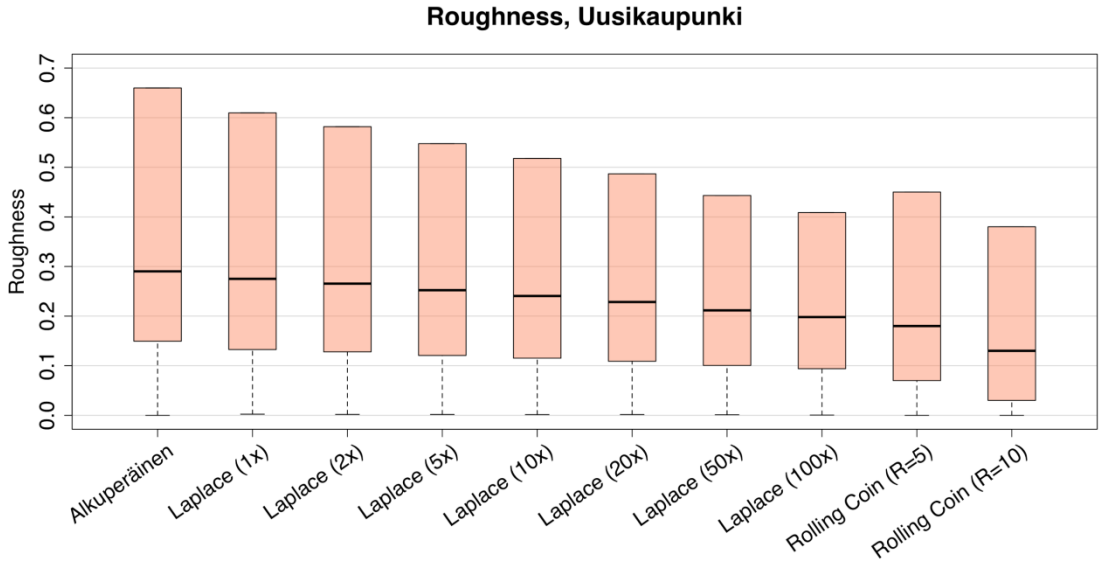
	Minimi	Alakvartiili	Mediaani	Yläkvartiili	Maksimi
Alkuperäinen	0,010	0,160	0,250	0,440	8,930
Laplace (1x)	0,008	0,140	0,228	0,390	8,560
Laplace (2x)	0,008	0,131	0,214	0,368	8,315
Laplace (5x)	0,006	0,120	0,197	0,342	7,653
Laplace (10x)	0,003	0,110	0,184	0,326	7,100
Laplace (20x)	0,004	0,102	0,173	0,314	6,310
Laplace (50x)	0,002	0,093	0,165	0,305	5,406
Laplace (100x)	0,002	0,089	0,160	0,301	4,795
Rolling Coin (R=5)	0,000	0,060	0,150	0,300	7,930
Rolling Coin (R=10)	0,000	0,040	0,130	0,280	7,840



Kuva 37. Pietarsaaren tutkimusalueen Roughness-arvojen jakaumat laatikkokuvaajina. Yläkvartiilia suuremmat arvot on jätetty kuvaajasta pois luettavuussyistä. Puuttuvat arvot löytyvät taulukosta 13.

Taulukko 14. Uudenkaupungin tutkimusalueen Roughness-arvojen tunnusluvut. Lukemat ovat metrejä.

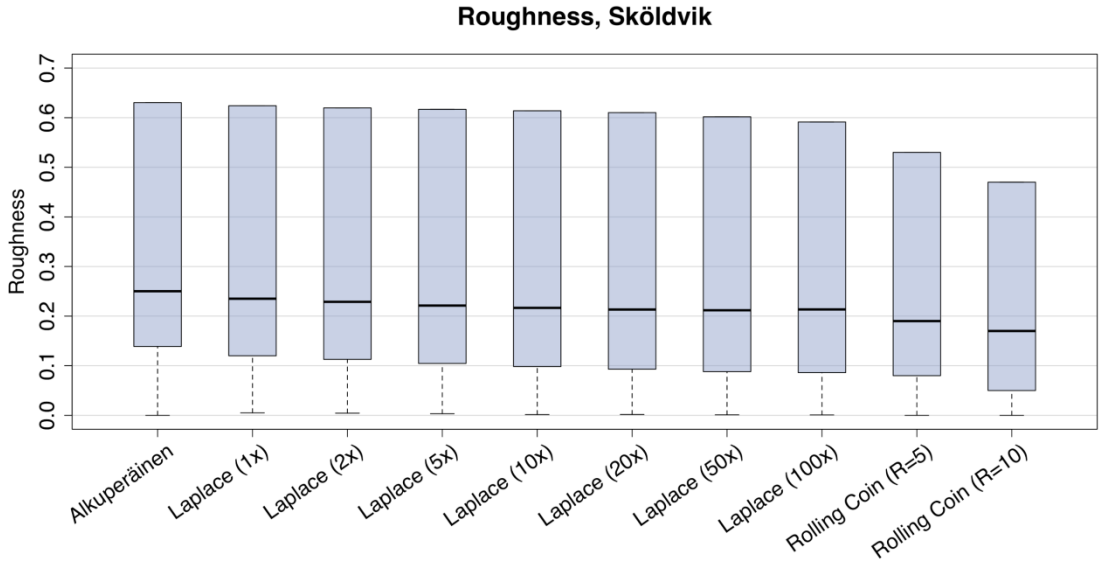
	Minimi	Alakvartiili	Mediaani	Yläkvartiili	Maksimi
Alkuperäinen	0,000	0,149	0,290	0,660	11,370
Laplace (1x)	0,002	0,133	0,275	0,610	11,370
Laplace (2x)	0,002	0,128	0,265	0,582	10,031
Laplace (5x)	0,002	0,121	0,252	0,548	9,490
Laplace (10x)	0,001	0,115	0,241	0,518	8,775
Laplace (20x)	0,001	0,109	0,228	0,487	7,869
Laplace (50x)	0,001	0,100	0,212	0,443	6,750
Laplace (100x)	0,000	0,094	0,198	0,409	5,842
Rolling Coin (R=5)	0,000	0,070	0,180	0,450	11,370
Rolling Coin (R=10)	0,000	0,030	0,130	0,380	11,180



Kuva 38. Uudenkaupungin tutkimusalueen Roughness-arvojen jakaumat laatikkokuvaajina. Yläkvartiilia suuremmat arvot on jätetty kuvaajasta pois luettavuussyistä. Puuttuvat arvot löytyvät taulukosta 14.

Taulukko 15. Sköldvikin tutkimusalueen Roughness-arvojen tunnusluvut. Lukemat ovat metrejä.

	Minimi	Alakvartiili	Mediaani	Yläkvartiili	Maksimi
Alkuperäinen	0,000	0,139	0,250	0,630	18,500
Laplace (1x)	0,005	0,120	0,235	0,624	17,310
Laplace (2x)	0,004	0,113	0,229	0,620	16,421
Laplace (5x)	0,003	0,105	0,221	0,617	15,392
Laplace (10x)	0,001	0,098	0,217	0,614	14,194
Laplace (20x)	0,002	0,093	0,213	0,610	13,332
Laplace (50x)	0,001	0,088	0,212	0,602	11,645
Laplace (100x)	0,001	0,086	0,213	0,591	10,074
Rolling Coin (R=5)	0,000	0,080	0,190	0,530	18,500
Rolling Coin (R=10)	0,000	0,050	0,170	0,470	17,310



Kuva 39. Sköldvikin tutkimusalueen Roughness-arvojen jakaumat laatikkokuvaajina. Yläkvartiilia suuremmat arvot on jätetty kuvaajasta pois luettavuussyistä. Puuttuvat arvot löytyvät taulukosta 15.

6.7. Syvyyskäyrät

Tässä työssä tutkittujen syvyysmallipintojen pehmennysmenetelmien tuloksia arvioitiin myös syvyysmallipinnoista automaattisin menetelmin määritettyjen vektorimuotoisten syvyyskäyrien lukumääriä ja pituuksia analysoimalla. Automaattisesti määritettyjen syvyyskäyrien lukumäärä ja pituudet antavat viitteitä syvyysmallipintojen pehmennyksen tasosta ja itse syvyyskäyrien yleistystasosta suhteessa nykyisillä navigointituotteilla esiintyvien syvyyskäyrien lukumääriin. Tuloksia voidaankin osaltaan hyödyntää pohdittaessa pehmennysmenetelmien käyttökelpoisuutta automatisoidussa syvyyskäyrätuotannossa.

Syvyyskäyrät määritettiin suoraan syvyysmallipinnoista käyttäen *Geospatial Data Abstraction Library* (GDAL Development Team 2018) -ohjelmakirjastoon sisältyvää käyränmääritystyökalua. Tuloksissa on huomioitu ainoastaan taulukossa 16 esitetyt syvyyskäyrät, sillä vertailuaineistona käytettyjä nykyisillä navigointituotteilla esiintyviä syvyyskäyriä ei ollut saatavilla kaikilta tutkimusalueilta muilla nimellissyvyyksillä.

Taulukko 16. Automaattisesti määritettyjen syvyyskäyrien lukumäärien ja pituuksien arvioinnissa käytettyjen syvyyskäyrien nimelliset ja todelliset syvyydet. Samoja syvyyksiä on käytetty myös nykyisillä karttatuotteilla esitettyjen syvyyskäyrien osalta.

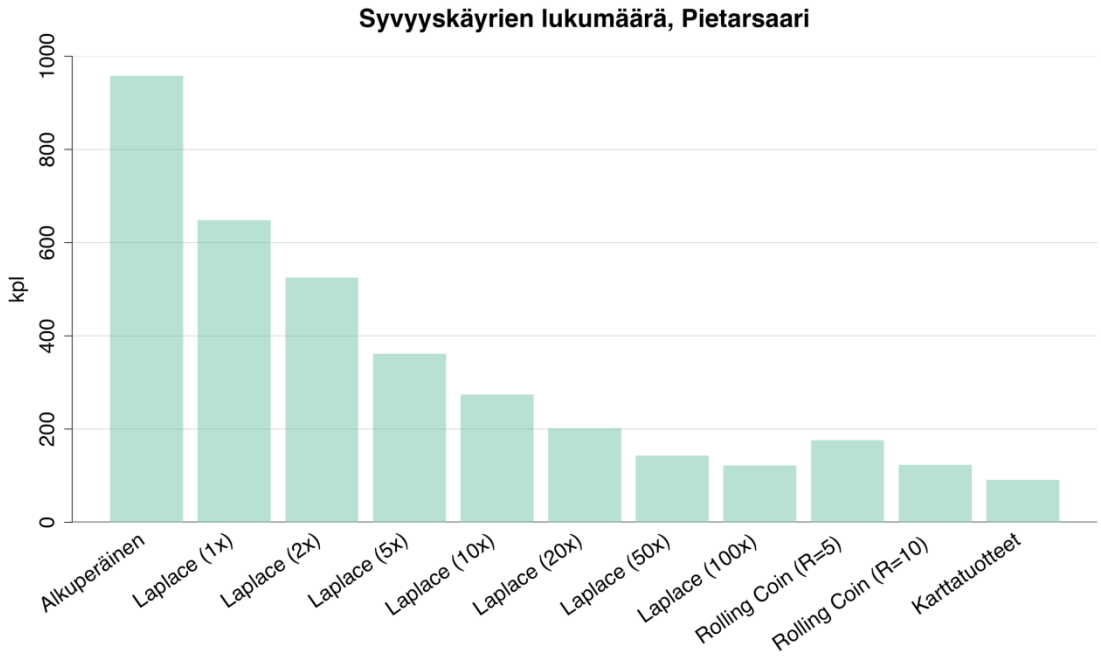
Nimellinen syvyys, m	Todellinen syvyys, m
3	3,09
6	6,09
10	10,09
20	20,09
50	50,99

6.7.1. Syvyyskäyrien lukumäärä

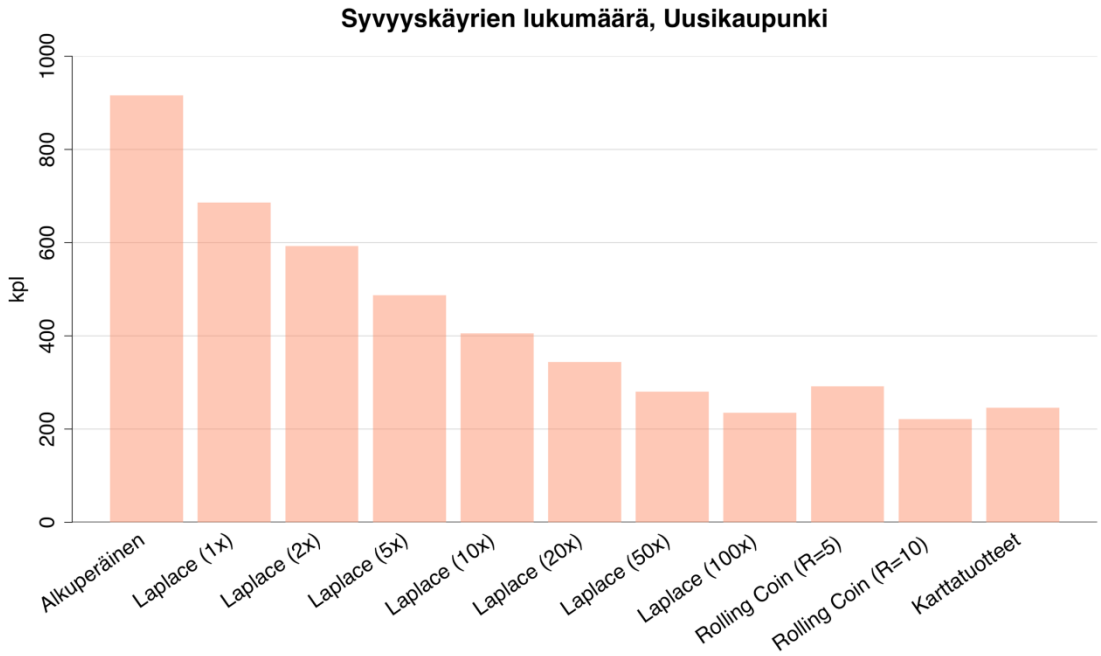
Syvyysmallipinnoista automaattisesti määritettyjen vektorimuotoisten syvyyskäyrien lukumääriä verrattiin samalla alueella nykyisillä navigointituotteilla esiintyvien syvyyskäyrien lukumäärään (taulukko 17 ja kuvat 40-42).

Taulukko 17. Erilaisista syvyysmallipinnoista määritettyjen syvyyskäyrien lukumäärät. Nykyisillä karttatuotteilla esitettyjen syvyyskäyrien (”Karttatuotteet”) lukumäärä on esitetty vertailuarvona.

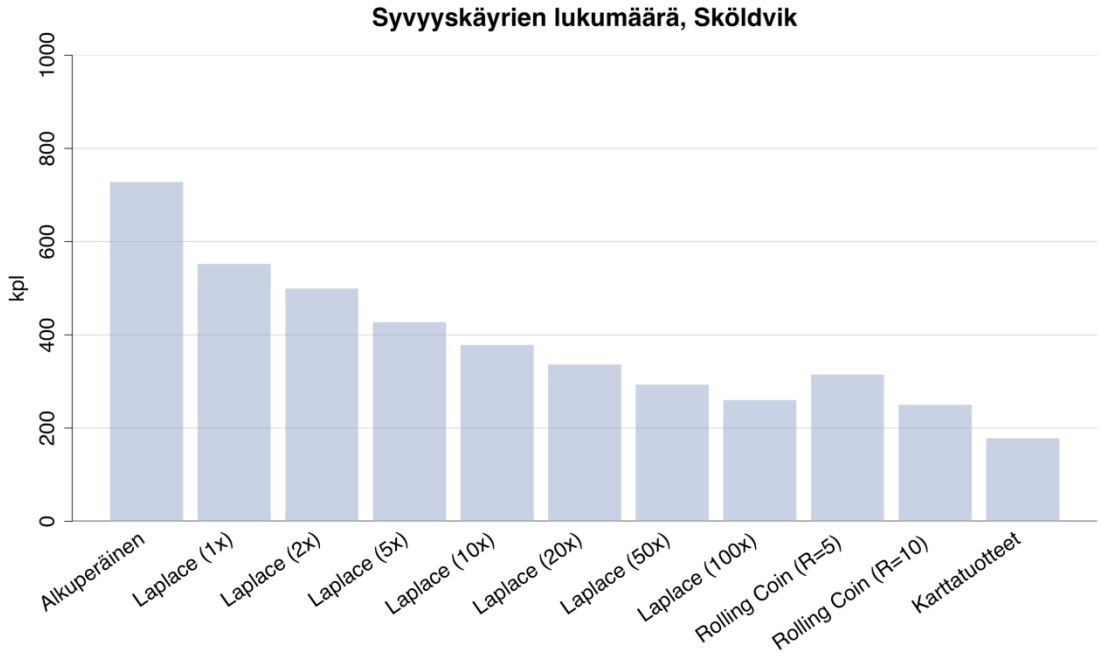
	Pietarsaari	Uusikaupunki	Sköldvik
Alkuperäinen	958	916	728
Laplace (1x)	648	686	552
Laplace (2x)	525	593	499
Laplace (5x)	361	487	427
Laplace (10x)	274	405	378
Laplace (20x)	202	344	336
Laplace (50x)	143	280	293
Laplace (100x)	122	235	260
Rolling Coin (R=5)	176	292	315
Rolling Coin (R=10)	123	221	250
Karttatuotteet	91	246	178



Kuva 40. Pietarsaaren tutkimusalueen eri tavoin pehmennetyistä syvyysmallipinnoista automaattisesti määritettyjen syvyyskäyrien lukumäärät. Nykyisillä karttatuotteilla esiintyvien syvyyskäyrien lukumäärä on otettu mukaan vertailuarvoksi.



Kuva 41. Uudenkaupungin tutkimusalueen eri tavoin pehmennetyistä syvyysmallipinnoista automaattisesti määritettyjen syvyyskäyrien lukumäärät. Nykyisillä karttatuotteilla esiintyvien syvyyskäyrien lukumäärä on otettu mukaan vertailuarvoksi.



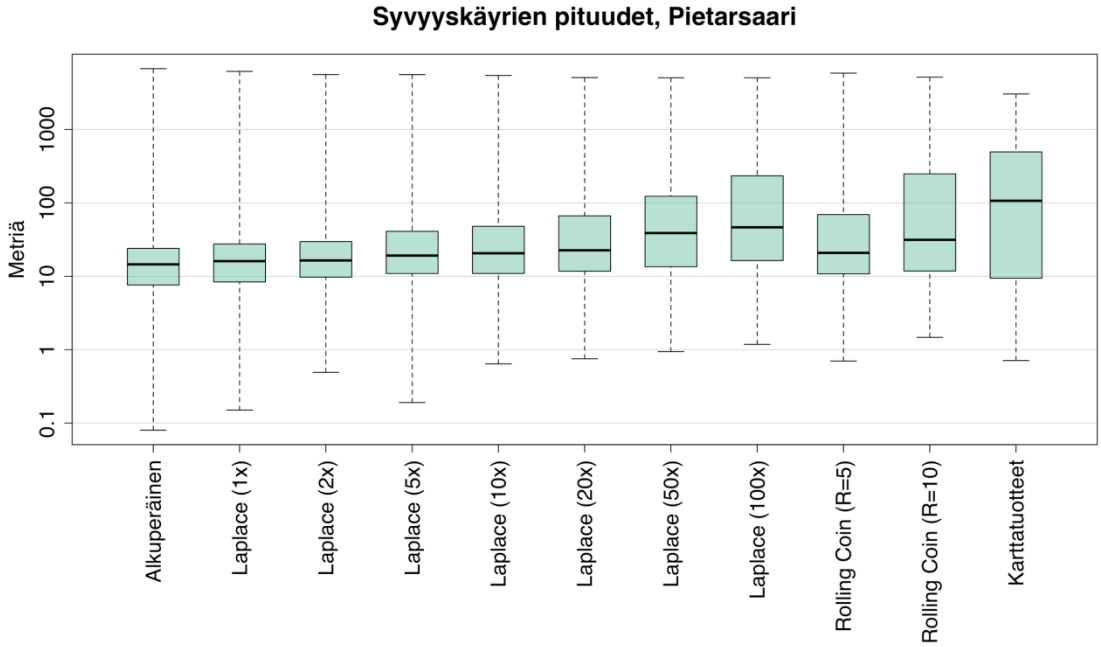
Kuva 42. Sköldvikin tutkimusalueen eri tavoin pehmennetyistä syvyysmallipinnoista automaattisesti määritettyjen syvyyskäyrien lukumäärät. Nykyisillä karttatuotteilla esiintyvien syvyyskäyrien lukumäärä on otettu mukaan vertailuarvoksi.

6.7.2. Syvyyskäyrien pituudet

Automaattisesti määritettyjen syvyyskäyrien pituuksia verrattiin samalla alueella nykyisillä navigointituotteilla esitettäviin syvyyskäyriin (taulukot 18–20 ja kuvat 43–45).

Taulukko 18. Syvyysmallipinnoista automaattisesti luotujen syvyyskäyrien pituudet Pietarsaaren testialueella. Mukana on vertailutietoina myös pehmentämättömästä pinnasta luotujen ja nykyisillä karttatuotteilla esiintyvien syvyyskäyrien pituudet. Pituudet ovat metrejä.

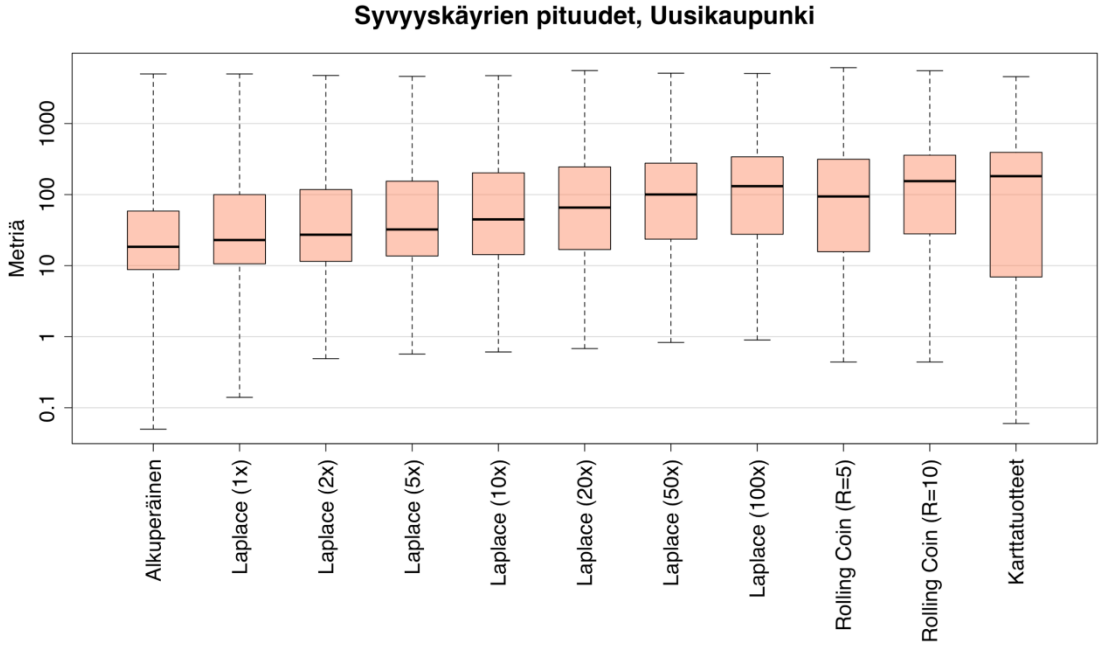
	Minimi	Alakvartiili	Mediaani	Yläkvartiili	Maksimi
Alkuperäinen	0,08	7,60	14,53	24,00	6740,54
Laplace (1x)	0,15	8,37	16,05	27,52	6198,91
Laplace (2x)	0,49	9,73	16,42	29,71	5604,37
Laplace (5x)	0,19	10,88	19,14	40,97	5589,00
Laplace (10x)	0,64	10,92	20,58	47,83	5450,14
Laplace (20x)	0,75	11,73	22,59	66,55	5095,10
Laplace (50x)	0,94	13,48	38,78	123,25	5066,72
Laplace (100x)	1,18	16,36	46,43	233,64	5059,13
Rolling Coin (R=5)	0,70	10,76	20,83	69,38	5853,10
Rolling Coin (R=10)	1,47	11,81	31,36	248,96	5180,87
Karttatuotteet	0,71	9,41	106,93	494,65	3046,68



Kuva 43. Syvyyskäyrien pituudet Pietarsaaren testialueella. Huomaa logaritminen asteikko.

Taulukko 19. Syvyysmallipinnoista automaattisesti luotujen syvyyskäyrien pituudet Uudenkaupungin testialueella. Mukana on vertailutietoina myös pehmentämättömästä pinnasta luotujen ja nykyisillä karttatuotteilla esiintyvien syvyyskäyrien pituudet. Pituudet ovat metrejä.

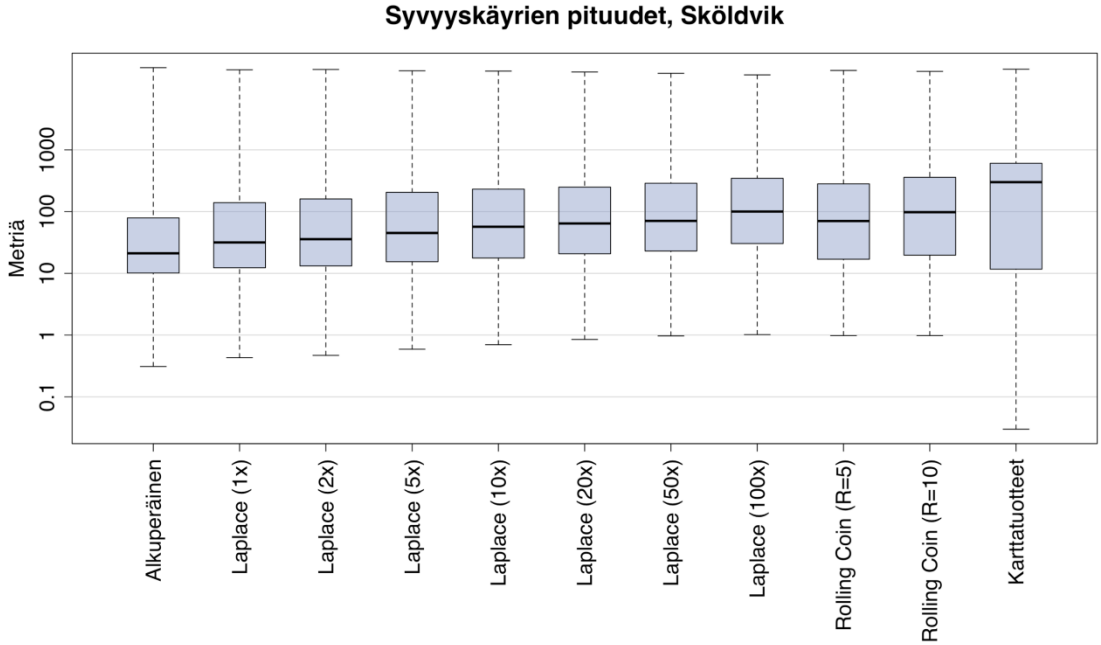
	Minimi	Alakvartiili	Mediaani	Yläkvartiili	Maksimi
Alkuperäinen	0,05	8,81	18,41	58,50	4996,65
Laplace (1x)	0,14	10,61	22,92	100,07	4993,38
Laplace (2x)	0,49	11,50	27,25	117,71	4746,53
Laplace (5x)	0,57	13,68	32,32	154,19	4613,57
Laplace (10x)	0,61	14,30	44,76	202,53	4717,41
Laplace (20x)	0,68	16,85	65,66	245,21	5557,00
Laplace (50x)	0,83	23,58	100,33	276,41	5113,26
Laplace (100x)	0,90	27,67	131,37	339,92	5065,01
Rolling Coin (R=5)	0,44	15,71	94,00	314,46	6110,36
Rolling Coin (R=10)	0,44	28,00	154,64	358,46	5544,31
Karttatuotteet	0,06	6,93	181,74	393,22	4571,28



Kuva 44. Syvyyskäyrien pituudet Uudenkaupungin testialueella. Huomaa logaritminen asteikko.

Taulukko 20. Syvyysmallipinnoista automaattisesti luotujen syvyyskäyrien pituudet Sköldvikin testialueella. Mukana on vertailutietoina myös pehmentämättömästä pinnasta luotujen ja nykyisillä karttatuotteilla esiintyvien syvyyskäyrien pituudet. Pituudet ovat metrejä.

	Minimi	Alakvartiili	Mediaani	Yläkvartiili	Maksimi
Alkuperäinen	0,31	10,13	21,12	79,02	21423,11
Laplace (1x)	0,43	12,32	31,65	139,48	19829,91
Laplace (2x)	0,47	13,19	35,78	160,45	20064,21
Laplace (5x)	0,59	15,46	45,07	205,34	19054,40
Laplace (10x)	0,70	17,72	56,81	231,00	18837,99
Laplace (20x)	0,85	20,75	64,34	249,10	18268,35
Laplace (50x)	0,97	23,04	70,79	288,01	17357,36
Laplace (100x)	1,02	30,40	100,10	346,13	16381,28
Rolling Coin (R=5)	0,98	16,96	70,30	281,63	19302,86
Rolling Coin (R=10)	0,98	19,73	97,93	358,84	18676,58
Karttatuotteet	0,03	11,69	300,31	605,99	20256,88



Kuva 45. Syvyyskäyrien pituudet Sköldvikin testialueella. Huomaa logaritminen asteikko.

7. KESKUSTELU

Työn aineistona toimivien alkuperäisten, käsittelemättömien syvyysmallien analysointi osoittaa tutkimusalueiden syvyysprofiilien eroavan toisistaan suuresti. Syvyystarkastelun tuloksista nähdään Sköldvikin olevan alueista keskimäärin selvästi syvin ja Pietarsaaren matalin. Alueiden syvyyksien suuret keskinäiset erot käyvät hyvin ilmi tarkasteltaessa kuvan 27 laatikkokuvaajia – tutkimusalueiden syvyyksien kvartiilivälit eivät osu kuvaajassa päällekkäin vaan kukin tutkimusalue on syvyysprofiililtaan selvästi kahdesta muusta alueesta erottuva.

Vaikka tutkimusalueiden syvyydet eivät itsessään olekaan tämän työn tutkimuskohteena, on tutkimusalueiden syvyysprofiilit hyvä tuntea yleispiirteissään työn muita tuloksia arvioitaessa. Liikenneviraston suorittamissa testeissä (Liikennevirasto 2017h) on ilmennyt, että ruopatut tai muutoin kapeat ja matalat väyläosuudet ovat herkkiä syvyysmallipinnan pehmennyksestä seuraavan syvyysmallipinnan mataloitumisen aiheuttamille syvyyskäyrien siirtymisille. Tämän työn tutkimusalueista Pietarsaaren meriväylä on monin paikoin kapea ja ruopattu (Liikennevirasto 2017f).

Syvyyskäyrien siirtyminen kauas todelliselta, pohjatopografian määräämältä sijainniltaan aiheuttaa kapeilla ruopatuilla väyläosuuksilla helposti syvyyskäyrien manuaalisen editoinnin tarpeen kasvamisen. Väyläalueen varmistettua vesisyvyyttä matalammat, mutta varmistetulle alueelle "pullistuvat" automaattisesti luodut syvyyskäyrät aiheuttavat aina manuaalisen syvyyskäyrän editointitarpeen syvyystietojen ristiriitaisuuksien poistamiseksi (Liikennevirasto 2012, 2017a). Täysin automaattisessa syvyyskäyräntuotannossa täydellinen lopputulos voikin käytännössä olla vaikea saavuttaa, sillä samalla kun syvyyskäyrien tulisi olla sekä kartografisesti luettavasti että navigoinnin kannalta turvallisesti yleistettyjä, tulisi niiden monin paikoin seurata todellisen pohjatopografian määrittämää syvyyskäyrärajaa syvyystietojen ristiriitaisuuden välttämiseksi.

Seuraavissa kappaleissa (7.1.-7.3.) vastataan tämän tutkielman tutkimuskysymyksiin. Kappaleessa 7.1. käsitellään tutkittujen syvyysmallipintojen pehmennysmenetelmien turvallisuutta navigointikäytön näkökulmasta. Kappaleessa 7.2. käsitellään tutkittujen pehmennysmenetelmien eri parametreilla tuottamien syvyysmallipintojen keskinäisiä eroja ja näiden pintojen eroja suhteessa alkuperäisiin syvyysmallipintoihin. Kappale 7.3.

käsittelee viimeistä tutkimuskysymystä ja pyrkii vastaamaan kysymykseen käytettyjen pehmennysmenetelmien käyttökelpoisuudesta automatisoidussa syvyyskäyrätuotannossa. Tutkimuskysymyksiin vastaamisen jälkeen (kappaleessa 7.4.) arvioidaan työstä saatuja tuloksia kokonaisuutena.

7.1. Syvyysmallipintojen pehmennysmenetelmien turvallisuus

Syvyysmallipintojen pehmentämiseen käytettyjen menetelmien navigointiturvallisuutta tutkittiin erotuspintojen avulla. Erotuspinnat laskettiin vähentämällä kustakin pehmennetystä syvyysmallipinnasta alkuperäinen syvyysmallipinta. Tarkempi kuvaus erotuspinnoista on annettu kappaleessa 5.2.

Navigointiturvallisuuden vaatimuksen voidaan tutkittavalla pehmennysmenetelmällä katsoa täyttyvän jos ja vain jos erotuspinnan minkään solun arvo ei ole negatiivinen. Tällaisessa tapauksessa kaikki syvyysmallipintaan pehmennysprosessissa tehdyt muokkaukset on tehty turvalliseen suuntaan, eli syvyysmallin soluarvoja madaltamalla. On kuitenkin muistettava, että pehmennettyjen syvyysmallipintojen turvallisuus voidaan taata vain, mikäli lähtöaineisto on navigointiturvallista.

Erotuspinnoista lasketut tilastolliset tunnusluvut (taulukot 3–5) osoittavat selvästi, että molemmat tässä työssä käytetyt syvyysmallipintojen pehmennysmenetelmät, *navigointiturvallinen Laplace-interpolointi* ja *Rolling Coin*, ovat navigointikäytön kannalta turvallisia. Menetelmien turvallisuus ilmenee taulukoiden 3–5 esittämistä erotuspintojen minimiarvoista, joista yksikään ei ole negatiivinen.

Näiden tulosten myötä voidaan katsoa, että tässä työssä sovellettujen syvyysmallipintojen pehmennysmenetelmien käyttö syvyysmallien käsittelyssä ja syvyysmalleihin perustuvassa automatisoidussa syvyyskäyrien tuotannossa on navigoinnin turvallisuuden näkökulmasta mahdollista, kunhan samalla huomioidaan, että todellinen matalan huippu voi sijaita missä vain kohdassa yksittäisen syvyysmallin solun sisällä. Kutakin syvyysmallin solua tulee siis kohdella alueena, jonka syvyys on yhtä kuin kyseisen solun syvyysarvo (Peters 2012).

7.2. Syvyysmallipintojen keskinäiset erot

Työn toisena tutkimuskysymyksenä oli selvittää miten eri pehmennysmenetelmillä tuotetut syvyysmallipinnat eroavat toisistaan ja alkuperäisistä syvyysmallipinnoista. Näitä eroja tutkittiin erotuspintojen, erotuspinnoista johdettujen *Root Mean Square Difference* eli RMS-poikkeamien ja erilaisten, tutkittavan solun naapurustoon perustuvien syvyysmallipintojen paikallista vaihtelua kuvaavien indeksien avulla. Käytetyt indeksit ja RMSD on kuvattu tarkemmin työn menetelmät-osiossa (kappale 5). Indeksit laskettiin vertailuarvojen saamiseksi myös alkuperäisistä syvyysmallipinnoista.

Pehmennetyistä syvyysmallipinnoista laskettujen erotuspintojen tilastollisista tunnusluvuista (taulukot 3-5) nähdään suoraan eri pehmennysmenetelmillä ja parametreilla tuotettujen pehmennettyjen syvyysmallipintojen keski- ja mediaanivirheet suhteessa alkuperäisiin syvyysmallipintoihin. Koska tässä työssä syvyysmallipintojen syvyyksien muokkaus on nimenomaisesti toivottua eikä virhe, käytetään syvyyssuuntaisista virheistä tästä eteenpäin nimitystä *poikkeama*.

Testatuista parametreista pienempää lanttia (R=5 solua) käytettäessä Rolling Coin -menetelmä näyttää tuottavan samansuuruisia keskipoikkeamia kuin navigointiturvallinen Laplace-interpolointi noin 10 pehmennyskierroksella. Suuremmalla lantilla (R=10 solua) käytettynä Rolling Coin -menetelmän keskipoikkeamat kasvavat pysyen kuitenkin keskimäärin hieman pienempinä kuin navigointiturvallisen Laplace-interpoloinnin 50 pehmennyskierroksella tuottamat keskipoikkeamat. Testatuista parametreista kautta linjan suurimmat keskipoikkeamat tuotti navigointiturvallinen Laplace-interpolointi 100 pehmennyskierroksella.

Pehmennettyjen ja alkuperäisten syvyysmallipintojen välisiä mediaanipoikkeamia (taulukot 3–5) tarkasteltaessa havaitaan, että tulokset ovat samansuuntaiset kuin keskipoikkeamien tapauksessa. Suuremmalla lantilla (R=10 solua) Rolling Coin -menetelmän tuottamat mediaanipoikkeamat näyttävät olevan samaa tasoa navigointiturvallisen Laplace-interpoloinnin kanssa, kun interpoloinnissa käytetään noin 20 pehmennyskierrosta. Pienemmällä lantilla (R=5 solua) Rolling Coin -menetelmän tuottamien pintojen mediaanipoikkeamat ovat matalia ja vastaavat tasoltaan navigointiturvallisen Laplace-interpoloinnin 2–5 pehmennyskierroksella tuottamia mediaanipoikkeamia.

RMS-poikkeamista saadut tulokset osoittavat, että pienempää (R=5 solua) lanttia käytettäessä Rolling Coin -menetelmällä pehmennettyjen syvyysmallipintojen RMS-poikkeamat ovat keskimäärin samaa tasoa navigointiturvallisella Laplace-interpoloinnilla pehmennettyjen syvyysmallipintojen saamien arvojen kanssa, kun interpoloinnissa käytetään noin 10 pehmennyskierrosta. Suurempaa lanttia (R=10 solua) käytettäessä Rolling Coin -menetelmällä pehmennettyjen syvyysmallipintojen RMSD-arvot vastaavat keskimäärin Laplace-interpolointimenetelmän käyttöä 50 pehmennyskierroksella. Testatuista pehmennysmenetelmistä ja parametreista selvästi suurimmat RMS-poikkeamat tuottaa navigointiturvallinen Laplace-interpolointi 100 pehmennyskierroksella.

Kappaleessa 5.4. tarkemmin esitelty *Bathymetric Position Index* eli BPI kuvaa tarkasteltavan syvyysmallin solun syvyyssuuntaista sijaintia suhteessa naapurustonsa keskiarvoon. Positiiviset arvot kertovat tutkittavan solun sijaitsevan ympäristöään matalammalla, negatiiviset puolestaan syvemmällä. Mikäli solun BPI-indeksin arvo on 0, viittaa tulos kyseisen solun naapuruston osalta joko tasaiseen pohjaan tai vaihtoehtoisesti tasaisesti viettävään rinteeseen (Lundblad et al. 2006; Kaskela et al. 2012).

BPI-indeksin tuloksista (taulukot 7–9 ja kuvat 31–33) nähdään selvästi, miten indeksin arvot pienenevät nopeasti Laplace-interpolointimenetelmän pehmennyskierroksia lisättäessä. Rolling Coin -menetelmällä pehmennettyjen pintojen BPI-arvot eivät muutu kovinkaan paljon alkuperäisten syvyysmallipintojen saamista arvoista ja ovat kautta linjan huomattavan suuria verrattuna Laplace-interpoloinnilla pehmennettyihin pintoihin. Tulosten valossa vaikuttaisikin siltä, että BPI kuvaa melko hyvin pehmennettyjen syvyysmallipintojen *sileyttä.* Käytetyistä syvyysmallien pehmennysmenetelmistä vain Laplace-interpolointi voi varmuudella tuottaa sileitä tai lähes sileitä pintoja (Peters 2012). Tämä voisi selittää Laplace-interpoloinnilla pehmennettyjen syvyysmallipintojen saamat, huomattavasti Rolling Coin -menetelmällä pehmennettyjä syvyysmallipintoja matalammat BPI-arvot.

TRI-indeksi (*Terrain Ruggedness Index,* esitelty tarkemmin kappaleessa 5.5.) kuvaa tutkittavan solun keskimääräistä syvyyssuuntaista etäisyyttä naapurustonsa soluihin. Indeksin laskennasta saadut tulokset osoittautuivat mielenkiintoisiksi. Tuloksista (taulu-

kot 10–12 ja kuvat 34–36) nähdään molempien syvyysmallipintojen pehmentämiseen käytettyjen menetelmien pienentävän keskimääräisiä TRI-arvoja selvästi alkuperäisten syvyysmallien arvoista. Tämä viittaa tutkittujen pehmennysmenetelmien odotetun kaltaiseen toimintaan – menetelmät näyttäisivät pienentävän syvyyksien keskimääräistä paikallista vaihtelua.

Tutkittujen pehmennysmenetelmien väliset erot ovat selviä. Rolling Coin -menetelmällä pehmennettyjen syvyysmallipintojen saamat TRI-arvot ovat kaikilla tutkimusalueilla ja kaikilla kvartiileilla lähes poikkeuksetta navigointiturvallisella Laplace-interpoloinnilla pehmennettyjä syvyysmallipintoja matalampia. Ainoat poikkeukset ovat Pietarsaaren tutkimusalueella pienemmällä lantilla (R=5 solua) tehty syvyysmallipinnan pehmennys, jonka TRI:n mediaani ja yläkvartiili ovat hieman korkeammat kuin navigointiturvallisella Laplace-interpoloinnilla korkeimmilla pehmennyskierroksilla (50 ja 100 kierrosta) käsiteltyjen syvyysmallipintojen arvot. Toinen poikkeus löytyy Uudenkaupungin tutkimusalueelta, jossa Rolling Coin tuotti pienemmällä lantilla (R=5 solua) hieman suuremman TRI-indeksin yläkvartiiliarvon kuin navigointiturvallinen Laplace-interpolointi korkeimmilla (50 ja 100 kierrosta) pehmennyskierroksilla.

Tuloksissa huomionarvoisia ovat myös TRI-indeksin saamat maksimiarvot. Maksimiarvot osoittavat Rolling Coin -menetelmän säilyttävän alkuperäisten syvyysmalliaineistojen maksimaaliset TRI-arvot navigointiturvallista Laplace-interpolointi-menetelmää paremmin. Tulos voi viitata Rolling Coin -menetelmän parempaan kykyyn säilyttää syvyysmalliaineistoon sisältyviä jyrkkiä ja pystysuoria pintoja myös pehmennetyssä syvyysmallissa.

Tarkasteltavan syvyysmallin solun naapurustoon kuuluvien solujen syvyyksien vaihteluväliä kuvaavasta *Roughness-indeksistä* (esitelty kappaleessa 5.6.) saadut tulokset ovat samansuuntaisia kuin TRI-indeksistä saadut tulokset. Roughness-indeksin laskennan tulokset (taulukot 13-15 ja kuvat 37-39) osoittavat molempien tutkittujen syvyysmallipintojen pehmennysmenetelmien pienentävän syvyysmallien soluarvojen paikallista vaihteluväliä. Käytetyistä syvyysmallipintojen pehmennyksen menetelmistä Rolling Coin näyttää tuottavan kaikilla tutkimusalueilla selvästi pienimmät keskimääräiset Roughness-arvot.

Tutkittujen pehmennysmenetelmien väliset erot näkyvät selvästi myös Roughness-indekseissä. Rolling Coin -menetelmä näyttää säilyttävän alkuperäisten syvyysmalliaineistojen maksimaaliset Roughness-arvot huomattavasti navigointiturvallista Laplace-interpolointimenetelmää paremmin. Kun huomioidaan Rolling Coin -menetelmän kautta linjan navigointiturvallista Laplace-interpolointimenetelmää matalammat keskimääräiset Roughness-arvot ja toisaalta maksimaalisten Roughness-arvojen parempi säilyminen alkuperäisten syvyysmallipintojen Roughness-maksimien tasolla, näyttäisi Rolling Coin -menetelmä tasoittavan syvyysmallipintoja tehokkaammin, säilyttäen samalla alkuperäisten syvyysmallipintojen jyrkät ja pystysuorat pinnat navigointiturvallista Laplace-interpolointimenetelmää paremmin.

Erotuspintojen tilastollisten tunnuslukujen, RMS-poikkeamien ja pehmennettyjen syvyysmallipintojen paikallisesta syvyysvaihtelusta kertovien indeksien tulosten tarkastelu osoitti molempien tutkittujen syvyysmallipintojen pehmennysmenetelmien toivovan odotetulla tavalla ja vähentävän syvyysmallien keskimääräistä paikallista syvyysvaihtelua. Myös tutkittujen menetelmien välisiin eroihin saatiin tuloksista uutta ymmärrystä. Tutkituista menetelmistä navigointituvallinen Laplace-interpolointi tuottaa mahdollisesti sileämpiä syvyysmallipintoja, kun taas Rolling Coin näyttäisi säilyttävän alkuperäiseen syvyysmalliaineistoon sisältyviä jyrkkiä ja pystysuoria pintoja navigointiturvallista Laplace-interpolointimenetelmää paremmin. Tulokset antavat myös viitteitä Rolling Coin -menetelmän kyvystä tuottaa navigointiturvallista Laplace-interpolointia tasaisempia syvyysmallipintoja, etenkin kun vertailussa huomioidaan menetelmien tuottamien syvyysmallipintojen erot alkuperäisiin syvyysmallipintoihin.

Pehmennettyjen ja alkuperäisten syvyysmallipintojen välisistä syvyyseroista kertovien keski- ja mediaanipoikkeamien ja RMSD-arvojen suora vertailu on kuitenkin hankalaa, sillä eri menetelmillä ja parametreilla pehmennettyjen syvyysmallipintojen vastaavuutta on vaikea arvioida tuntematta pehmennetyistä syvyysmallipinnoista laskettujen syvyyskäyrien ominaisuuksia. Seuraavassa kappaleessa esiteltävien automaattisesti määritettyjen syvyyskäyrien tulosten perusteella voidaan kuitenkin olettaa, että Rolling Coin -menetelmällä pehmennettyjä syvyysmallipintoja tulisi ensisijaisesti verrata vain korkeammilla (N=20, N=50, N=100) pehmennyskierrosten lukumäärillä pehmennettyihin navigointiturvallisella Laplace-interpolointimenetelmällä tuotettuihin syvyysmallipintoihin.

7.3. Automaattisesti määritetyt syvyyskäyrät

Erotuspinnat, RMSD ja naapuruston syvyysvaihtelua kuvaavat indeksit eivät vielä anna riittävää ymmärrystä tutkittujen syvyysmallipintojen pehmennysmenetelmien soveltuvuudesta käytännön automatisoituun syvyyskäyrätuotantoon. Jotta menetelmien ja parametrien soveltumista todelliseen automaattiseen syvyyskäyräntuotantoon voitaisiin arvioida, tulee käytettyjen indeksien, erotuspintojen ja niistä johdettujen poikkeamien lisäksi tarkastella syvyysmallipinnoista automaattisin menetelmin määritettyjen syvyyskäyrien ominaisuuksia.

Syvyysmallipinnoista automaattisesti määritetyt syvyyskäyrät tuotettiin käyttämällä GDAL -ohjelmakirjastoon (GDAL Development Team 2018) sisältyvää gdal_contour -sovellusta (GDAL 2018), jonka dokumentaatiosta selviää varsinaisen käyränmääritysalgoritmin toiminta. Sovellus tulkitsee syvyysmallin solut solujen keskellä sijaitseviksi noodeiksi, joiden perusteella syvyyskäyrät määritetään bilineaarisesti interpoloimalla. Tuotettujen syvyyskäyrien kulkusuunta on vakio ja ne kiertävät matalikkoja myötäpäivään – matalan huippu on siis aina syvyyskäyrän kulkusuuntaan nähden oikealla puolella.

Vaikka syvyyskäyränmääritykseen käytetyt pehmennetyt syvyysmallipinnat olisivatkin navigoinnin näkökulmasta turvallisia, ei nyt valittua käyränmääritystyökalua voi sen toiminnan vuoksi suoraan käyttää navigointiturvallisten syvyyskäyrien tuottamiseen, sillä käyränmääritystyökalun tuottamat syvyyskäyrät voivat päätyä kulkemaan syvyyskäyrän arvoa matalamman solun kulman tai kulmien yli (Peters et al. 2014). Ongelma voidaan luonnollisesti ratkaista melko helpostikin – tämän työn liitteenä löytyvässä lähdekoodissa (liite 1) on esimerkin vuoksi sisällytettynä myös syvyyskäyrien navigointiturvallisuuden varmistava toiminnallisuus.

Yhtenä tämän työn tutkimuskysymyksistä on selvittää vaikuttavatko työssä käytetyt syvyysmallipintojen pehmennysmenetelmät käyttökelpoisilta automatisoidun syvyyskäyrien tuotannon näkökulmasta. Koska menetelmien käyttökelpoisuutta selvitettiin osaltaan automaattisesti luotujen syvyyskäyrien lukumääriä ja pituuksia arvioimalla, ei tavoitteen toteutuminen edellytä tuotettujen syvyyskäyrien ehdotonta navigointiturvallisuutta. Pehmennetyistä syvyysmallipinnoista automaattisesti luotujen syvyyskäyrien

pituuksia ja lukumääriä on verrattu Liikenneviraston merikartoituspalvelut-yksikön SYVÄ-tuotantotietokannasta saatuihin, nykyisillä merikarttatuotteilla esitettyihin syvyyskäyriin.

Syvyyskäyrien lukumääristä ja pituuksista saatuja tuloksia on ajateltava vain suuntaa-antavina, sillä absoluuttista totuutta esimerkiksi "oikeasta" syvyyskäyrien yleistystasosta on asian subjektiivisuudesta johtuen käytännössä mahdotonta antaa. Oksanen ja Sarjakoski (2005) pitävät korkeuskäyrien kartografisen laadun arviointia haastavana ja epävarmana erityisesti juuri siksi, että mitään objektiivisesti määritettyä korkeuskäyrien kartografisen laadun referenssiä ei ole olemassa.

Tuloksien tarkastelussa tulee huomioida myös aineistojen vertaustasojen ero. Nykyisillä merikarttatuotteilla esitettävien syvyyskäyrien määrityksen vertaustaso ei ole sama kuin tässä työssä käytettyjen syvyysmalliaineistojen vertaustaso. Nykyisillä merikartoilla käytetään syvyystietojen vertaustasona vielä toistaiseksi teoreettisen keskiveden tasoja (Ilmatieteen laitos 2018), kun taas tämän työn aineistona käytetyt syvyysmalliaineistot on sidottu N2000-vertaustasoon (Julkisen hallinnon tietohallinnon neuvottelukunta 2007; Liikennevirasto 2018c). Tulevaisuudessa myös kaikki merikartta-aineistot tuotetaan N2000-vertaustasossa (Liikennevirasto 2018c).

Syvyyskäyrien automaattisesta määrityksestä saatuja tuloksia tarkasteltaessa on huomioitava myös tässä työssä syvyyskäyrien ominaisuuksien tarkasteluun käytettyjen tunnuslukujen puutteet ja heikkoudet. Syvyyskäyrien lukumäärien osalta on muistettava, että vaikka syvyyskäyrien matalampi lukumäärä antaakin viitteitä lähekkäisten matalikkojen paremmasta yleistymisestä yhden yhteisen syvyyskäyrän sisään, se ei kerro mitään lopputuloksen kartografisesta laadusta tai syvyyskäyrien sijainnista todellisen käyrärajan suhteen. Myöskään automaattisesti luotujen syvyyskäyrien pituuksia ei voida pitää yksiselitteisen luotettavana syvyyskäyrien laadun mittarina, sillä käyrien suurempi pituus voi matalikkojen yhteen yleistämisen sijaan johtua myös syvyyskäyrien huomattavasta mutkikkuudesta. Syvyyskäyrien lukumäärien ja pituuksien tuloksia voidaan kuitenkin pitää suuntaa antavina – etenkin kun niitä samalla verrataan nykyisillä tuotteilla esitettyjen syvyyskäyrien vastaaviin ominaisuuksiin.

Kenties tärkein ja käyttökelpoisin tämän työn melko suppeasta syvyyskäyrien ominaisuuksien tarkastelusta saatava tietämys kohdistuu tutkituille syvyysmallipintojen pehmennysmenetelmille sopivien parametrien valintaan – pehmennetyistä syvyysmallipinnoista automaattisin menetelmin määritettyjen syvyyskäyrien lukumääriä ja pituuksia tarkastelemalla ja näitä tietoja nykyisillä navigointituotteilla esitettyihin syvyyskäyriin vertaamalla voidaan mahdollisesti helpottaa sopivimpien syvyysmallipintojen pehmennyksessä käytettyjen parametrien valintaa. Saatuja tuloksia hyödynnetään myös seuraavassa kappaleessa käsiteltävän tulosten kokonaistarkastelun yhteydessä.

Pehmennetyistä syvyysmallipinnoista automaattisesti luotujen syvyyskäyrien lukumääristä ja pituuksista saadut tulokset (taulukot 17–20 ja kuvat 40–45) osoittavat tässä työssä käytettyjen syvyysmallipintojen pehmennysmenetelmien laskevan syvyysmallipintojen pohjalta automaattisin menetelmin tuotettujen syvyyskäyrien lukumäärää ja kasvattavan syvyyskäyrien pituutta, kun vertailukohtana pidetään käsittelemättömiä, matalia suosivalla Shoal Bias -periaatteella tuotettuja syvyysmalleja.

Taulukossa 17 esitetyistä automaattisesti luotujen syvyyskäyrien lukumääristä nähdään alkuperäisten syvyysmallipintojen synnyttävän erittäin suuren määrän syvyyskäyriä. Automaattisesti tuotettujen syvyyskäyrien lukumäärä laskee kuitenkin huomattavasti jo pienilläkin syvyysmallipintoihin kohdistuvilla pehmennystoimenpiteillä. Tuloksista selviää syvyyskäyrien lukumäärän putoavan alle puoleen viimeistään kahdellakymmenellä pehmennyskierroksella navigointiturvallista Laplace-interpolointia käytettäessä. Myös Rolling Coin -menetelmä alensi syvyysmallipinnoista automaattisesti määritettyjen syvyyskäyrien lukumääriä huomattavasti. Jo pienemmän lantin (R=5 solua) käyttö pudotti syvyyskäyrien lukumäärän alle puoleen kaikilla tutkimusalueilla.

Rolling Coin -menetelmän suuremmalla lantilla (R=10 solua) tuottamien syvyyskäyrien lukumäärää koskevat tulokset ovat Pietarsaaren tutkimusaluetta lukuun ottamatta testattujen syvyysmallipintojen pehmennysmenetelmien pienimpiä. Pietarsaaren tutkimusalueella tulos on miltei identtinen navigointiturvallisen Laplace-interpoloinnin tuloksen kanssa – navigointiturvallinen Laplace-interpolointi tuotti sadalla pehmennyskierroksella 122 syvyyskäyrää ja Rolling Coin -menetelmä suuremmalla lantilla (R=10 solua) 123 syvyyskäyrää. Syvyyskäyrien lukumääriä arvioitaessa voidaan todeta, että tutkituista parametreista lähimmäksi nykyisillä merikarttatuotteilla esitettävien syvyyskäyrien lu-

kumäärää päästiin käyttämällä navigointiturvallista Laplace-interpolointia sadalla pehmennyskierroksella tai vaihtoehtoisesti Rolling Coin -menetelmää suuremmalla lantilla (R=10 solua).

Syvyysmallipinnoista automaattisin menetelmin luotujen syvyyskäyrien pituuteen vaikuttaa syvyysmallipinnan pehmennyksessä syntyvän syvyyskäyrärajan yleistyksen lisäksi syvyyskäyrien mutkikkuus ja aineiston vertaustaso. Koska automaattisesti tuotettujen syvyyskäyrien pituuksien perusteella tehtävä syvyysmallipintojen pehmennysmenetelmien tuotantokäyttöön soveltumisen arviointi sisältää hyvin suuria epävarmuuksia, keskitytään tämän kappaleen syvyyskäyrien pituuksien tarkastelussa vain kahteen syvyyskäyrien pituuksien tunnuslukuun, syvyyskäyrien mediaanipituuteen ja pituuksien yläkvartiiliin. Näiden tunnuslukujen voidaan katsoa kuvaavan syvyyskäyrien ominaisuuksia tässä tehtävän arvioinnin kannalta riittävällä tasolla.

Pehmennetyistä syvyysmallipinnoista automaattisesti luotujen syvyyskäyrien pituuksien tilastollisten tunnuslukujen tarkastelu osoittaa saatujen tulosten vaihtelevan suuresti eri tutkimusalueiden välillä. Uudenkaupungin tutkimusalueella (taulukko 19 ja kuva 44) automaattisesti tuotetuilla syvyyskäyrillä päästiin sekä mediaani- että yläkvartiilipituuksien osalta lähimmäksi nykyisillä karttatuotteilla esitettäviä syvyyskäyriä, kun taas Pietarsaaren (taulukko 18 ja kuva 43) ja etenkin Sköldvikin (taulukko 20 ja kuva 45) tutkimusalueilla syvyyskäyrien pituuksien tunnusluvut eroavat toisistaan huomattavasti enemmän. Löydös on mielenkiintoinen ja liittyy mahdollisesti nykyisillä merikarttatuotteilla esitettävien syvyyskäyrien manuaalisessa tuotantoprosessissa luonnostaan esiintyvään yleistystason vaihteluun. Tämä nykyiseen tuotantoprosessiin sisältyvä yleistystason vaihtelu selittyy prosessin manuaalisella luonteella – eri henkilöiden tuottama "kädenjälki" ei ole identtistä, vaikka syvyystiedon uusimisen prosessiohjeissa (Liikennevirasto 2012, 2017a) pyritäänkin antamaan yleisiä ohjeita kartografisesta yleistystasosta. Toinen vertailuaineiston ja automaattisesti luotujen syvyyskäyrien välisiä eroja selittävä tekijä on aineistojen toisistaan eroava vertaustaso. Vertaustason vaikutusta saatuihin tuloksiin on vaikea arvioida ilman uutta, samassa vertaustasossa tehtyä vertailututkimusta.

Rolling Coin -menetelmällä tuotettujen syvyyskäyrien mediaanipituus vastasi parhaiten vertailuaineiston syvyyskäyrien mediaanipituutta Uudenkaupungin testialueella. Navi-

gointiturvallisella Laplace-interpolointimenetelmällä pehmennetyistä syvyysmallipinnoista luotujen syvyyskäyrien mediaanipituus puolestaan vastasi paremmin vertailuaineiston syvyyskäyriä Pietarsaaren ja Sköldvikin tutkimusalueilla. Sköldvikin osalta tutkittujen pehmennysmenetelmien tulokset olivat lähellä toisiaan, kun taas Pietarsaaren tutkimusalueella erot olivat selvempiä. Kun tarkastelussa huomioidaan syvyyskäyrien mediaanipituuksien lisäksi myös yläkvartiilipituudet, huomataan Rolling Coin -menetelmän tuottavan kaikilla tutkimusalueilla suuremman ja paremmin vertailuaineistoa vastaavan syvyyskäyrien yläkvartiilipituuden.

Syvyyskäyrien lukumääristä ja pituuksista saatujen tulosten kokonaismerkittävyyttä on hankala arvioida. Syvyyskäyrien pituuksien tilastollisista tunnusluvuista voidaan lähinnä päätellä itse syvyysmallipintojen pehmennykseen käytettävien menetelmien todella toimivan tarkoituksessaan ja sekä vähentävän syvyysmallipinnoista automaattisin menetelmin tuotettujen syvyyskäyrien lukumääriä että kasvattavan niiden pituuksia. Tältä osin tutkittujen syvyysmallipintojen pehmennysmenetelmien voidaan siis arvioida vaikuttavan automatisoituun syvyyskäyrien tuotantoon soveltuvilta.

7.4. Tulosten kokonaistarkastelu

Vaikka jo melko pienetkin syvyysmallipintojen pehmennystoimenpiteet näyttävät vaikuttavan merkittävästi pehmennetyistä syvyysmallipinnoista vektoroitujen syvyyskäyrien lukumääriin ja pituuksiin (taulukot 17–20 ja kuvat 40–45), voidaan saatujen tulosten perusteella katsoa ainoastaan syvyysmallipintoja voimakkaimmin muokkaavien menetelmä-parametriyhdistelmien yltävän automatisoidun syvyyskäyrätuotannon kannalta lupaaviin tuloksiin.

Navigointiturvallisessa Laplace-interpoloinnissa parametrina annettavan pehmennyskierrosten lukumäärän kasvattaminen paransi selvästi tällä menetelmällä pehmennetyistä syvyysmallipinnoista tuotettujen syvyyskäyrien lukumäärän ja pituuksien vastaavuutta nykyisillä merikarttatuotteilla esitettäviin syvyyskäyriin. Testatuilla parametreilla ei saatujen tulosten valossa kuitenkaan vielä ylletty nykyisten merikarttatuotteiden syvyyskäyrien yleistystasolle. Myös Rolling Coin -menetelmästä saatiin samansuuntaisia tuloksia – pehmennetyistä syvyysmallipinnoista vektoroidut syvyyskäyrät ovat tulosten

perusteella selvästi yleistettyjä, mutta eivät kuitenkaan yleistystasoltaan nykyisillä merikarttatuotteilla esitettäviä syvyyskäyriä vastaavia.

Tutkittujen syvyysmallipintojen pehmennysmenetelmien keskinäinen suora vertailu on hankalaa, sillä menetelmien ja parametrien vastaavuutta joudutaan arvioimaan ainoastaan lopputuloksena saatavien, pehmennetyistä syvyysmallipinnoista vektoroitujen, syvyyskäyrien lukumääriä ja pituuksia vertailemalla. Tulosten perusteella on kuitenkin selvää, että Rolling Coin -menetelmän tutkituilla parametreilla tuottamia tuloksia parhaiten vastaavat navigointiturvallisen Laplace-interpoloinnin tuottamat tulokset, kun käytetty pehmennyskierrosten lukumäärä on melko korkea. Pehmennetyistä syvyysmallipinnoista automaattisesti tuotettujen syvyyskäyrien tulosten perusteella menetelmien ja parametrien paras vastinpari näyttäisikin olevan Rolling Coin suuremmalla lantilla (R=10 solua) ja Laplace-interpolointi 100 pehmennyskierroksella. Pienemmällä lantilla (R=5 solua) käytettynä Rolling Coin -menetelmän tulokset vaihtelivat suhteessa Laplace-interpoloinnin tuottamiin tuloksiin hieman enemmän, mutta karkeahkona arviona voidaan todeta Rolling Coin -menetelmän pienemmällä lantilla tuottamien tuloksien vastaavan navigointiturvallisen Laplace-interpoloinnin tuloksia, kun interpoloinnin pehmennyskierrosten lukumäärä on noin 50. Kun tässä työssä saatujen tulosten tarkastelussa huomioidaan nämä automaattisesti tuotettujen syvyyskäyrien tuloksien tarkastelun perusteella arvioidut syvyysmallipintojen pehmennysmenetelmien ja parametrien vastinparit, voidaan saatuja tuloksia arvioida paremmin myös kokonaisuutena.

Kun tarkastellaan erotuspintojen tilastollisista tunnusluvuista (taulukot 3–5) saatuja pehmennettyjen ja alkuperäisten syvyysmallipintojen välisiä syvyyseroja kuvaavia keski- ja mediaanipoikkeamia huomioiden samalla tutkittujen syvyysmallipintojen pehmennysmenetelmien menetelmä-parametri -vastinparit, huomataan tutkittujen menetelmien eroavan toisistaan huomattavasti. Lopullisten vektorimuotoisten syvyyskäyrien osalta likimain saman lopputuloksen tuottavista vastinpareista navigointiturvallinen Laplace-interpolointi tuotti kaikilla tutkimusalueilla suuremmat syvyyssuuntaiset keski- ja mediaanipoikkeamat. Myös poikkeamien keskihajonta on navigointiturvallisella Laplace-interpolointimenetelmällä kaikilla tutkimusalueilla vastinpariaan suurempi. Vastaavat havainnot voidaan tehdä myös tarkasteltaessa pehmennettyjen syvyysmallipintojen RMS-poikkeamia (taulukko 6 ja kuvat 28–30). Navigointiturvallisen Laplace-

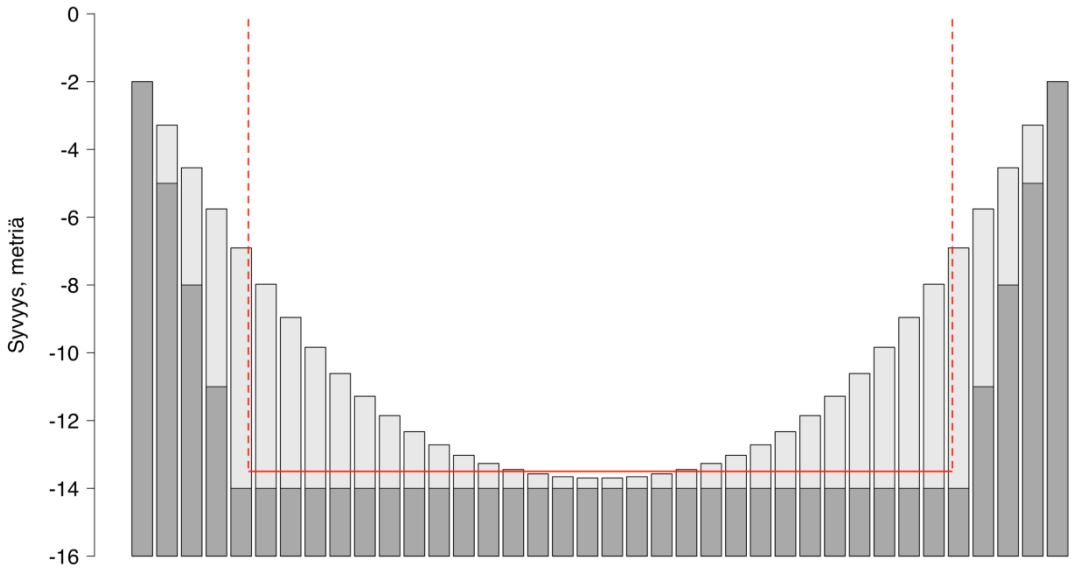
interpolointimenetelmän tuottamien syvyysmallipintojen RMS-poikkeama on kaikilla tutkimusalueilla suurempi kuin vastinpariensa.

Erotuspinnoista laskettujen, alkuperäisten ja pehmennettyjen syvyysmallipintojen välisiä syvyyspoikkeamia kuvaavien tunnuslukujen perusteella voidaan todeta Rolling Coin -menetelmällä pehmennettyjen syvyysmallipintojen vastaavan keskimäärin paremmin alkuperäistä syvyysmallipintaa kuin lopputuloksena syntyvien syvyyskäyrien osalta likimain saman lopputuloksen tuottavalla navigointiturvallisella Laplace-interpolointimenetelmällä pehmennettyjen syvyysmallipintojen.

Syvyysmallipintojen paikallista syvyysvaihtelua kuvaavista BPI-, TRI- ja Roughness-indekseistä saadut tulokset (taulukot 7–15 ja kuvat 31–39) osoittavat eri menetelmillä pehmennettyjen syvyysmallipintojen eroavan toisistaan huomattavasti. Kun tarkastellaan ylempänä määriteltyjä menetelmä-parametri -vastinpareja, huomataan että navigointiturvallinen Laplace-interpolointi tuottaa kaikilla tutkimusalueilla selvästi vastinparejaan matalammat BPI-arvot. Koska BPI kuvaa tarkasteltavan solun sijaintia suhteessa naapurustonsa syvyyksien keskiarvoon (Weiss 2001; Lundblad et al. 2006), voidaan tuloksen tulkita viittaavan Laplace-interpoloinnin kykyyn tuottaa vastinpariaan *sileämpiä* syvyysmallipintoja. Kuitenkin kun tarkastellaan TRI- ja Roughness -indekseistä saatuja tuloksia, huomataan Rolling Coin -menetelmän tuottamien syvyysmallipintojen olevan keskimäärin jopa tasaisempia kuin navigointiturvallisella Laplace-interpolointimenetelmällä tuotetut vastinparinsa.

Kappaleessa 7.2. arvioitiin TRI- ja Roughness-indekseistä saatujen tulosten maksimiarvojen viittaavan Rolling Coin -menetelmän vastinpariaan parempaan kykyyn säilyttää alkuperäisen syvyysmallipinnan jyrkät ja pystysuorat rinteet. Kappaleessa 2.3. esitetystä navigointiturvallisen Laplace-interpolointimenetelmän toiminnan kuvauksesta voidaan päätellä menetelmän ominaisuuksiin todella kuuluvan jyrkkien ja pystysuorien rinteiden häivyttämisen rinteiden alaosia ja pohjia korottamalla. Koska pehmennysmenetelmä on luonteeltaan iteratiivinen, aiheuttaa jokainen pehmennyskierros alun perin pystysuoran rinteen loivenemisen entisestään. Samalla pohjan madaltuminen myös ulottuu kierros kierrokselta kauemmas alkuperäisestä rinteestä.

Kuvassa 46 on esitetty kuvitteellinen laskennallinen poikkileikkausesimerkki navigointiturvallisen Laplace-interpolointimenetelmän jyrkkäreunaiselle, kanavamaiseksi ruopatulle väylälle aiheuttamasta syvyysmallipinnan muutoksesta. Kuvassa todellista pohjatopografiaa edustaa tummanharmailla pylväillä kuvattu poikkileikkausprofiili. Kuvitteellisen väyläalueen yhdessä sille määritetyn *haraussyvyyden* kanssa muodostama syvyydeltään varmistettu *väylätila* on kuvattu punaisin viivoin. Kun kyseistä poikkileikkauskohtaa pehmennetään navigointiturvallisella Laplace-interpolointimenetelmällä 100 pehmennyskierroksen verran, päädytään kuvassa vaaleanharmailla pylväillä kuvattuun poikkileikkausprofiiliin. Tuloksesta nähdään menetelmän aiheuttavan syvyydeltään varmistetun haraustason rikkoontumisen. Tämä puolestaan johtaa syvyystietojen loogiseen ristiriitaan ja pehmennetystä syvyysmallipinnasta automaattisesti luotujen syvyyskäyrien mahdolliseen pullistumiseen syvyydeltään varmistetulle väyläalueelle. Kuvan 46 esimerkissä navigointiturvallisella Laplace-interpolointimenetelmällä pehmennetystä syvyysmallipinnasta automaattisesti luodut 10 metrin syvyyskäyrät rikkoisivat syvyystietojen ristiriidattomuusvaatimusta pullistumalla syvyydeltään varmistetulle, syvyyskäyrän määrityssyvyyttä syvemmälle väyläalueelle ja ne tulisi myöhemmin korjata manuaalisesti editoimalla. Tämä syvyyskäyrien siirtyminen on menetelmään sisältyvä ominaisuus, joka voi käytännössä rajoittaa käyttökelpoista pehmennyskierrosten lukumäärää muuallakin kuin ruopatuilla väyläosuuksilla.



Kuva 46. Kuvitteellinen esimerkki ruopatun väylän poikkileikkauksesta (tumma harmaa) ja saman kohdan poikkileikkauksesta, kun pintaa on pehmennetty Laplace-interpoloinnilla 100 pehmennyskierrosta (vaalea harmaa).

Mikäli kuvan 46 tummanharmailla pylväillä kuvatulla väyläalueella syvyysmallipinnan pehmennykseen käytettäisiin navigointiturvallisen Laplace-interpolointimenetelmän sijaan Rolling Coin -menetelmää ja sen parametrina lanttia, jonka halkaisija on pienempi kuin esitetyn väyläalueen leveys, ei tällaista ristiriitatilannetta syntyisi vaan Rolling Coin -menetelmällä pehmennetyn syvyysmallipinnan poikkileikkausprofiili vastaisi kuvassa 46 tummanharmaalla esitettyä poikkileikkausta. On toki muistettava, että syvyyskäyrien ehdottomasta turvallisuusvaatimuksesta johtuen automaattisesti määritetyt syvyyskäyrät voivat joissain tilanteissa pullistua varmistetulta syvyydeltään syvyyskäyrää syvemmälle väyläalueelle, vaikka itse syvyysmallipinnan ja varmistetun väyläalueen välillä ei olisikaan ristiriitaa.

Kappaleessa 2.4.2. esitellystä 2,5-ulotteisen Rolling Coin -menetelmän toimintaperiaatteesta voidaan päätellä menetelmällä pehmennetyistä syvyysmallipinnoista vektoroitujen syvyyskäyrien pyrkivän noudattamaan alkuperäisen syvyysmallipinnan osoittamaa syvyyskäyrärajaa aina, kun se on mahdollista. Rolling Coin -menetelmä ei siis ole yhtä herkkä syvyyskäyrärajojen siirtymiseen kuin Laplace-interpolointi. Laplace-interpolointimenetelmällä luotujen syvyyskäyrien mahdollisen siirtymän suuruuden arviointi on hankalaa, sillä siirtymään vaikuttaa pohjatopografia syvyyskäyrärajan välittömässä läheisyydessä – mitä lähempänä syvyysmallin esittämä pohja on syvyyskäyrärajaa, sitä herkemmin ja kauemmas syvyyskäyrät siirtyvät Laplace-interpolointimenetelmää käytettäessä paikaltaan.

Tuloksien kokonaisarvioinnin lopuksi voidaan todeta tätä työtä varten kehitetyn Rolling Coin -menetelmän 2,5 -ulotteisen version osoittautuneen automatisoidun syvyyskäyrien tuotannon näkökulmasta hyvin lupaavaksi syvyysmallipintojen pehmennysmenetelmäksi. Myös navigointiturvalliseksi muokattu Laplace-interpolointi osoittautui tehokkaaksi syvyysmallipintojen pehmennyksessä. Tutkituilla menetelmillä on keskenään suuria eroja, joista johtuen myös niiden tuottamat syvyysmallipinnat eroavat ominaisuuksiltaan toisistaan. Menetelmien eroista huolimatta molemmat menetelmät näyttävät tuottavan navigoinnin kannalta turvallisia ja alkuperäisiä syvyysmallipintoja paremmin automaattisen syvyyskäyränmääritykseen soveltuvia pintoja. Menetelmien jatkotestaus parhaiden parametrien ja käytäntöjen löytämiseksi olisikin todennäköisesti hyvin mielenkiintoinen jatkotutkimuksen aihe.

8. JOHTOPÄÄTÖKSET

Tutkituista syvyysmallipintojen pehmennysmenetelmistä molemmat osoittautuivat navigoinnin kannalta turvallisiksi ja automatisoidun syvyyskäyrätuotannon osalta lupaaviksi. Saatujen tulosten valossa vaikuttaakin siltä, että yleistettyjen vektorimuotoisten syvyyskäyrien automaattista tuottamista kannattaa lähestyä nimenomaan syvyysmallipintojen muokkauksen kautta – kun varsinaiset syvyyskäyrät määritetään suoraan pehmennetyistä syvyysmallipinnoista, vältytään hankalilta topologiaongelmilta kuten vektorimuotoisten käyrien yleistämisessä usein kohdattavilta syvyyskäyrien risteämisiltä.

Nyt tutkittujen menetelmien käyttöönottamiseen liittyy kuitenkin sopivien parametrien valinnan haasteet eikä tässä työssä edes pyritty löytämään menetelmille parhaita parametreja. Kaikesta syvyyskäyrien tuottamiseen liittyvästä manuaalisesta editoinnista ei todennäköisesti myöskään voida näiden menetelmien käytön myötä luopua. Kartografisen asiantuntijan paneutumista tarvittaneen edelleen joidenkin syvyyskäyriä koskevien kartografisten päätösten ja pienten editointien tekemiseen sekä menetelmien ehdottoman turvallisuuden mahdollisesti aiheuttamien ristiriitojen korjaamiseen.

Tutkittujen menetelmien käytöllä voitaneen kuitenkin tehostaa nykyisessä merikartantuotannossa pullonkaulaksi muodostunutta manuaaliseen työhön nojaavaa syvyystietojen uusimistyötä merkittävästi. Koska oikein käytettynä nyt tutkituilla menetelmillä tuotetut syvyyskäyrät ovat varmuudella navigointiturvallisia ja samalla yleistettyjä, ei syvyyskäyrien manuaalinen editointi ole täysin välttämätöntä kuin mahdollisten syvyydeltään varmistetuilla alueilla esiintyvien ristiriitojen korjaamiseksi. Selvästi suurin osa kaikista nykyaikaisilla täyden peittävyyden (MBES) merenmittauksilla katetuille alueille määritettävistä syvyyskäyristä voitaisiin siis tuottaa joko täysin tai vähintäänkin suurin osin automaattisesti. Selvän ajansäästön myötä tehostuneen prosessin lisäksi automaattisesti tuotettujen syvyyskäyrien eduiksi voidaan laskea syvyyskäyrien navigointiturvallisuuden varmistuminen ja etenkin Rolling Coin -menetelmää käytettäessä syvyyskäyrien yleistystason vakiointi ja ennakoitavuus. Yleistystason vakiointi on aiemmin mainituista asiantuntijoiden yksilöllisistä eroista johtuen käytännössä mahdotonta saavuttaa ilman syvyyskäyrien tuotannon osittaista tai täyttä automatisointia.

Menetelmien parametrien valinnan osalta Rolling Coin saattaa osoittautua käytännössä iteratiivista Laplace-interpolointia helpommaksi. Rolling Coin -menetelmässä lantin kokoa ja muotoa voidaan haarukoida suoraan toivotun kartografisen yleistystason mukaan, eikä menetelmä ole Laplace-interpoloinnin tavoin luonteeltaan iteratiivinen.

Laplace-interpoloinnin pehmennyskierrosten lukumäärän valinta voi menetelmän iteratiivisen luonteen ja Suomen merialueiden geomorfologian suuren alueellisen vaihtelun (mm. Kaskela et al. 2012; Kaskela 2017) johdosta osoittautua käytännössä melko hankalaksi. Jollain alueella hyvän lopputuloksen tuottava pehmennyskierrosten lukumäärä saattaa toisenlaisen geomorfologian alueilla tuottaa huomattavasti kehnompia lopputuloksia. Navigointiturvallisen Laplace-interpoloinnin osalta yksi mielenkiintoinen jatkotutkimus- ja kehitysmahdollisuus voisikin olla pehmennyskierrosten lukumäärän automaattinen asettaminen kunkin alueen syvyysmallipinnan ominaisuuksien mukaan. Tällaista adaptiivista, maaston paikalliseen korkeusvaihteluun perustuvaa korkeusmallipintojen pehmennystä ovat tutkineet mm. Kettunen et al. (2017) kehittäessään korkeuskäyrien automaattista mallipohjaista tuottamista.

Nyt saatujen tulosten perusteella Rolling Coin -menetelmä vaikuttaa automaattisen syvyyskäyrätuotannon kannalta hieman Laplace-interpolointia lupaavammalta. Menetelmän paremmuus näkyy ensinnäkin siinä, ettei Rolling Coin Laplace-interpoloinnin tavoin madalla syvyysmallipintoja siellä, missä madaltaminen ei ole toivottua, kuten ruopatuilla tai muutoin kapeilla väyläosuuksilla. Toiseksi Rolling Coin -menetelmän eduksi voidaan lukea sillä tuotettujen syvyyskäyrien parempi pysyminen alkuperäisen syvyysmallipinnan mukaisissa käyrärajoissa – paikallinen pohjatopografia ei siis vaikuta Rolling Coin -menetelmällä tuotettujen syvyyskäyrien siirtymiseen niin voimakkaasti kuin Laplace-interpoloinnissa. Lisäksi vaikuttaa siltä, että keskenään melko samanlaisen yleistystason tuottavista menetelmä-parametri -vastinpareista Rolling Coin -menetelmällä pehmennetyt pinnat eroavat alkuperäisistä syvyysmallipinnoista keskimäärin selvästi Laplace-interpolointia vähemmän.

Tässä työssä saatujen tulosten ja kokemusten valossa kiinnostaviksi lisätutkimuksen aiheiksi ovat nousseet nyt tutkittujen syvyysmallipintojen pehmennysmenetelmien edelleen kehittämisen lisäksi sopivien parametrien valintaan ja menetelmien tuottamien syvyyskäyrien kartografisen laadun arviointiin liittyvät kysymykset. Etenkin tämän työn

rajauksen ulkopuolelle jäänyt automaattisesti tuotettujen syvyyskäyrien kartografisen laadun arviointi ja siihen perustuva syvyyskäyrien kartografisen laadun maksimointi vaikuttavat mielenkiintoisilta ja varmuudella hyödyllisiltä tutkimuskysymyksiltä. Lopputuloksena saatavien syvyyskäyrien kartografinen arviointi tuottaisi arvokasta tietoa menetelmien nykyisen käyttökelpoisuuden lisäksi menetelmien mahdolliselle jatkokehitykselle.

Rolling Coin -menetelmästä saatujen lupaavien tulosten myötä menetelmä ollaan ottamassa Liikenneviraston merikartoituksessa tuotantokäyttöön. Liikennevirastossa parhaillaan käynnissä olevan uuden merikarttatietojärjestelmän hankinnan yhteydessä menetelmä on päätetty sisällyttää osaksi hankittavaa AHTI-järjestelmää. Uuden tuotantojärjestelmän ja prosessimallin myötä Liikennevirasto toivoo pystyvänsä tehostamaan syvyystietojen tuottamisen prosessia merkittävästi. Uuden merikarttatietojärjestelmän toimittava CARIS aikoo sisällyttää Rolling Coin -menetelmän samalla osaksi kaikille asiakkailleen tarjoamaansa perustoiminnallisuutta. Onkin mahdollista, että tätä työtä varten kehitetty Rolling Coin -menetelmä päätyy vielä melko laajaan ja vaativaan käyttöön – se jos mikä olisi hieno lopputulos tälle tutkielmalle!

9. KIITOKSET

Haluan kiittää työtoveriani, syvyysmalliprojektissa projektipäällikkönä toiminutta merikapteeni Stefan Engströmiä lukuisista hyvistä keskusteluista, monipuolisesta sparrauksesta ja etenkin Rolling Coin -menetelmän kehittämisessä saadusta avusta ja ideoista. Työtovereistani kiitokset ansaitsevat myös kartta- ja paikkatietoasiantuntija Mikko Hovi, merikarttaryhmän päällikkö Juha Tiihonen ja merikartoituspalvelut-yksikön päällikkö Rainer Mustaniemi.

Erityisen suuret ja lämpimät kiitokset haluan osoittaa kumppanilleni Rosalle – joka sieti tämän työn tekemiseen liittynyttä kiukutteluani ja auttoi minua jaksamaan kokopäiväisen työn ja gradun kirjoittamisen paineessa. Rosa – ilman sinun tukeasi tämä työ olisi taatusti jäänyt tekemättä!

10. KIRJALLISUUS

Aluevalvontalaki (2000). Suomen säädöskokoelma 755. Helsinki.

Belikov, V., V. Ivanov, V. Kontorovich, S. Korytnik & A. Semenov (1997). The non-sibsonian interpolation: a new method of interpolation of the values of a function on an arbitrary set of points. *Computational mathematics and mathematical physics* 37: 1, 9–15.

Calder, B. R. & L. A. Mayer (2003). Automatic processing of high-rate, high-density multibeam echosounder data. *Geochemistry, Geophysics, Geosystems* 4: 6, 1–22.

Calder, B. R. & D. E. Wells (2007). CUBE User's manual. *Center for Coastal and Ocean Mapping* 1217.

CARIS (2017). Caris Base editor, version 4.4. CARIS, Fredericton, NB, Canada.

Christ, N. H., R. Friedberg & T. D. Lee (1982). Weights of links and plaquettes in a random lattice. *Nuclear Physics B* 210: 3, 337–346.

GDAL (2018). gdal_contour. 20.04.2018. <*www.gdal.org/gdal_contour.html*>

GDAL Development Team (2018). GDAL - Geospatial data abstraction library, version 2.2.4. Open Source Geospatial Foundation.

Hashim, M. (1996). New texural extraction method using rolling ball and ripping membrane transforms. *In Asian Conference on Remote Sensing 1996 (ACRS 1996)*. 20.04.2018. <*http://a-a-r-s.org/aars/proceeding/ACRS1996/Papers/DIP96-1.htm*>

Heiskanen, T. (2008). Merikarttojen syvyyskäyrien tuottamisen automatisointi - turvallisen navigointikäytön ehdoilla. Julkaisematon diplomityö. 71 s. Tietotekniikan osasto, Teknillinen korkeakoulu.

Hiyoshi, H. & K. Sugihara (1999). Two generalizations of an interpolant based on voronoi diagrams. *International journal of shape modeling* 05: 2, 219–231.

Hyndman, R. J. & A. B. Koehler (2006). Another look at measures of forecast accuracy. *International Journal of Forecasting* 22: 4, 679–688.

IHO (2000). S-57, IHO Transfer standard for digital hydrographic data. *International Hydrographic Organization Special Publication 57 (Edition 3.1)*. International Hydrographic Bureau, Monaco.

IHO (2005). *Manual on hydrography. International Hydrographic Organization Publication C-13 (1st Edition).* International Hydrographic Bureau, Monaco.

IHO (2008). S-44, IHO Standards for hydrographic surveys. *International Hydrographic Organization Special Publication 44 (5th Edition)*. International Hydrographic Bureau, Monaco.

IHO (2012). S-102, Bathymetric surface product specification. *International Hydrographic Organization Publication S-102 (Edition 1.0.0)*. International Hydrographic Bureau, Monaco.

IHO (2013). List of S-100 based product specifications. International Hydrographic Organization. 5.3.2018. *<www.iho.int/mtg_docs/com_wg/HSSC/HSSC_Misc/List_of_S-100_Product_Specifications.pdf>*

IHO (2014). S-52, Specifications for chart content and display aspects of ECDIS. *International Hydrographic Organization Publication S-52 (Edition 6.1)*. International Hydrographic Organization, Monaco.

IHO (2017a). S-4, Regulations of the IHO for international (INT) charts and chart specations of the IHO. *International Hydrographic Organization Publication S-4 (Edition 4.7.0)*. International Hydrographic Organization, Monaco.

IHO (2017b). S-100 - Universal hydrographic data model. *International Hydrographic Organization Publication S-100 (Edition 3.0.0)*. International Hydrographic Organization, Monaco.

IHO (2018). International Hydrographic Organization. 15.5.2018. *<www.iho.int>*

Ilmatieteen laitos (2018). Keskivesitaulukot. 29.4.2018. *<http://ilmatieteenlaitos.fi/keskivesitaulukot>*

IMO (1974). International convention for the safety of life at sea (SOLAS).

Julkisen hallinnon tietohallinnon neuvottelukunta (2007). JHS 163 Suomen korkeusjärjestelmä N2000. *JHS-Suositukset* 163, 1–13.

Kaskela, A. (2017). Seabed landscapes of the Baltic Sea: Geological characterization of the seabed environment with spatial analysis techniques. *Erikoisjulkaisut - Special Publications* 100, 1–43. Geologian tutkimuskeskus - Geological survey of Finland, Espoo.

Kaskela, A. M., A. T. Kotilainen, Z. Al-Hamdani, J. O. Leth & J. Reker (2012). Seabed geomorphic features in a glaciated shelf of the Baltic Sea. *Estuarine, Coastal and Shelf Science* 100, 150–161.

Kernighan, B. W. & D. M. Ritchie (1988). *The C programming language*. 2. p. 1019 s. Prentice-Hall, Upper Saddle River, NJ, USA.

Kettunen, P., C. Koski & J. Oksanen (2017). A design of contour generation for topographic maps with adaptive DEM smoothing. *International Journal of Cartography* 3: 1, 19–30.

Kotulak, K., A. Froń, A. Krankowski, G. O. Pulido & M. Henrandez-Pajares (2017). Sibsonian and non-Sibsonian natural neighbour interpolation of the total electron content value. *Acta Geophysica* 65: 1, 13–28.

Kujala, P., M. Hänninen, T. Arola & J. Ylitalo (2009). Analysis of the marine traffic safety in the Gulf of Finland. *Reliability Engineering and System Safety* 94: 8, 1349–1357.

Laki Liikennevirastosta (2009). Suomen säädöskokoelma 862. Helsinki.

Lecordix, F., L. Gondol, P. Julien & K. Jaara (2009). Representation du relief en zone de montagne. *Proceedings of 24th International Cartographic Conference*, 1–13. International Cartographic Association.

Lecours, V., M. F. J. Dolan, A. Micallef & V. L. Lucieer (2016). A review of marine geomorphometry, the quantitative study of the seafloor. *Hydrology and Earth System Sciences* 20: 8, 3207–3244.

Li, X. & M. E. Hodgson (2004). Vector field data model and operations. *GIScience and Remote Sensing* 41: 1, 1–24.

Liikennevirasto (2010). *Kartta 1. Merikarttamerkit. Karttamerkit, lyhenteet ja käsitteet suomalaisilla ja INT-kartoilla*. 5. painos. 124 s. Liikennevirasto, Helsinki.

Liikennevirasto (2011). Väyläkortti, Sköldvikin 15,3 m väylä. Liikennevirasto, Helsinki. 12.12.2017. *<http://www.liikennevirasto.fi/ammattimerenkulku/liikkuminen-vesivaylilla/vaylakortit>*

Liikennevirasto (2012). SYVÄ-kohdeluokkien ohjeet (versio 0.1). Liikenneviraston merikartoituksen sisäinen ohje. Liikennevirasto, Helsinki.

Liikennevirasto (2015). Väyläkortti, Uudenkaupungin 12,5 m väylä. Liikennevirasto, Helsinki. 12.12.2017. *<http://www.liikennevirasto.fi/ammattimerenkulku/liikkuminen-vesivaylilla/vaylakortit>*

Liikennevirasto (2017a). Syvyystiedon uusimisen pelisäännöt (versio 1.4). Liikenneviraston merikartoituksen sisäinen ohje. Liikennevirasto, Helsinki.

Liikennevirasto (2017b). Merenkulun älyväylä. 12.12.2017. *<https://www.liikennevirasto.fi/hankkeet/digitalisaatiohanke/merenkulun-alyvayla>*

Liikennevirasto (2017c). Selvitys merikarttatuotteiden, -palvelujen ja -toiminnan kehittämisestä, loppuraportti. Liikenneviraston merikartoituksen sisäinen raportti. Liikennevirasto, Helsinki.

Liikennevirasto (2017d). Digitalisaatiohanke. 12.12.2017. *<https://www.liikennevirasto.fi/hankkeet/digitalisaatiohanke>*

Liikenneviraston (2017e). Rauman meriväylän ja sataman syventäminen. 12.12.2017. *<https://www.liikennevirasto.fi/rauman_vayla>*

Liikennevirasto (2017f). Väyläkortti, Pietarsaaren 11,0 m väylä. Liikennevirasto, Helsinki. 12.12.2017. *<http://www.liikennevirasto.fi/ammattimerenkulku/liikkuminen-vesivaylilla/vaylakortit>*

Liikennevirasto (2017g). Caris Base editor -sovelluksen automaattisen syvyyskäyränluonnin työkalujen testitulokset. Liikenneviraston merikartoituksen sisäinen raportti. Liikennevirasto, Helsinki.

Liikennevirasto (2017h). CARIS Base Editor -sovelluksella tehtyjen syvyysmallipintojen pehmennystestien tulokset. Liikenneviraston merikartoituksen sisäinen raportti. Liikennevirasto, Helsinki.

Liikennevirasto (2018a). Merikartat. 12.12.2017. *<https://www.liikennevirasto.fi/ammattimerenkulku/merikartat>*

Liikennevirasto (2018b). RollingCoin. 20.4.2018. *<https://github.com/finnishtransportagency/RollingCoin>*

Liikennevirasto (2018c). Korkeusjärjestelmä N2000 - Liikennevirasto. 20.5.2018. *<https://www.liikennevirasto.fi/ammattimerenkulku/merikartat/korkeusjarjestelma-n2000>*

Lundblad, E. R., D. J. Wright, J. Miller, E. M. Larkin, R. Rinehart, D. F. Naar, B. T. Donahue, S. M. Anderson & T. Battista (2006). A benthic terrain classification scheme for American Samoa. *Marine Geodesy* 29: 2, 89–111.

Oksanen, J. & T. Sarjakoski (2005). The EVRS and the need for contour updating in national topographic maps. *Proceedings of the XXII International Cartographic Conference - Mapping Approaches into a Changing World*. International Cartographic Association.

Open Navigation Surface Working Group (2004). Format specification document: description of bathymetric attributed grid object (BAG), version 1.5.1. Open Navigation Surface Working Group.

Open Navigation Surface Working Group (2017). The open navigation surface project - Esperanto for hydrographers! 10.2.2018. <*http://www.opennavsurf.org*>

Peters, R. (2012). A Voronoi- and surface-based approach for the automatic generation of depth-contours for hydrographic charts. Master's thesis. 124 p. Department of GIS Technology, Delft University of Technology.

Peters, R., H. Ledoux & M. Meijers (2014). A Voronoi-based approach to generating depth-contours for hydrographic charts. *Marine Geodesy* 37: 2, 145–166.

QPS (2010). Fledermaus, version 7.4.2a. Portsmouth, NH, USA.

R Core Team (2017). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria.

Rannikkokartta 1:50 000, lehti 17 Tallörn-Glosholm. 8. p. Liikennevirasto, Helsinki 2017.

Rashed, M. (2016). Rolling ball algorithm as a multitask filter for terrain conductivity measurements. *Journal of Applied Geophysics* 132, 17–24.

Riley, S. J., S. D. DeGloria & R. Elliot (1999). A terrain ruggedness index that quantifies topographic heterogeneity. *Intermountain Journal of Sciences* 5: 1–4, 23–27.

Ritter, G. X., J. N. Wilson & J. L. Davidson (1990). Image algebra: An overview. *Computer Vision, Graphics and Image Processing* 49: 3, 297–331.

Ritter, N. & M. Ruth (1997). The GeoTiff data interchange standard for raster geographic images. *International Journal of Remote Sensing* 18: 7, 1637–1647.

Rosenkranz, B., T. Knudsen, H. E. Mortensen & P. B. Michealsen (2012). Automatic generation of contour lines based on DK-DEM. *Technical report series* 13, 1–32. National Survey and Cadastre, Denmark.

Sanastokeskus TSK (2018). *Geoinformatiikan sanasto*. 4. p. 128 s. Maanmittauslaitos, Helsinki.

Sibson, R. (1981). A brief description of natural neighbour interpolation. *Teoksessa* Barnett, V. (toim.): *Interpreting Multivariate Data*, 21–36. John Wiley & Sons Ltd, Hoboken, NJ, USA.

Smith, S. M. (2003). The navigation surface: a multipurpose bathymetric database. Master's thesis. 86 p. Center for Coastal and Ocean Mapping, University of New Hampshire.

Sternberg, S. R. (1983). Biomedical image processing. *Computer* 16: 1, 22–34.

Sukumar, N., B. Moran, A. Yu Semenov & V. V Belikov (2000). Natural neighbor galerkin methods. *International Journal for Numerical Methods in Engineering* 50: 1, 1–27.

Valentine, P. C., S. J. Fuller & L. A. Scully (2004). Terrain ruggedness analysis and distribution of boulder ridges in the Stellwagen Bank national marine sanctuary region. *Posteriesitys*. Galway, Ireland.

Weiss, A. (2001). Topographic position and landforms analysis. *Posteriesitys, ESRI User Conference*. San Diego, CA, USA.

Wilson, M. F. J., B. O'Connell, C. Brown, J. C. Guinan & A. J. Grehan (2007). Multiscale terrain analysis of multibeam bathymetry data for habitat mapping on the continental slope. *Marine Geodesy* 30: 1–2, 3–35.

Zhang, X. & E. Guilbert (2011). A multi-agent system approach for feature-driven generalization of isobathymetric line. *Teoksessa* Ruas, A. (toim.): *Advances in Cartography and GIScience Volume 1: Selection from ICC 2011, Paris*, 477–495. Springer, Berlin, Heidelberg.

LIITE 1

```
// The following source code was written by Topi Filppula for his Master's thesis.
// Check the header file for more information.
// ALL RIGHTS RESERVED

//
//          - - - Header file (bathymetrictools.h) - - -
//


#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <unistd.h>
#include "gdal.h"
#include "cpl_conv.h"
#include "cpl_string.h"

/*
*   Header file:
*   - Includes
*   - General information
*   - Compilation instructions
*   - Constants
*   - Structured datatype definitions
*   - Function prototypes / forward declarations
*/

/*
*   Bathymetric surface tools (written in 2017/2018 to be a part of my master's thesis)
*
*   Author:                 Topi Filppula
*   Latest update:          20.05.2018
*   Latest version:         0.9b
*
*
*           ALL RIGHTS RESERVED.
*
*
*   Originally written for Unix (Mac OS), possible later versions might or might not be
cross-platform.
*
*   Depends on GDAL (Geospatial Data Abstraction Library, see www.gdal.org). GDAL uses
MIT/X -type license.
*
*   Compilation instructions:
*   1. Make sure GDAL is installed
*   2. Link GDAL on compilation
*   3. Optimize and check on compilation
*
*   Compiling:
*   1. Use either Make and the included makefile or
*   2. Compile manually for example like:
*   gcc -g -O3 -march=native -Wall -Wextra -Wfloat-equal -Werror -pedantic -pedantic-
errors -std=gnu11 -o bathytools *.c -lgdal
*/


// Constants:
#define GO      200000
#define NOGO    100000
#define NODATA  -9999
#define EPSILON 0.00001
```

```
// Structured datatype to hold bathymetric surface:
struct FloatSurface {
    char *inputfp;          // Original file path
    char *projection;       // CRS information in WKT
    double *geotransform;   // Georeferencing parameters
    float **array;          // Data array (2D, float**)
    double nodata;          // Source file nodata value
    int rows;               // Number of rows
    int cols;               // Number of columns
};


// Structured datatype to hold abstract process surfaces:
struct LimitSurface {
    char *inputfp;          // Original file path
    char *projection;       // CRS information in WKT
    double *geotransform;   // Georeferencing parameters
    int **array;            // Data array (2D, int**)
    int rows;               // Number of rows
    int cols;               // Number of columns
};


// Structured datatype to hold the coin:
struct Coin {
    int originalRadius;     // Radius
    int originalDiameter;   // Diameter
    int trimmedDiameter;    // Trimmed diameter
    int maxIndex;           // Max indexes
    int **array;            // Array (2D, int**)
};


// Control functions: (main.c)
int main(void);
void rollingCoinSmoothing(void);
void laplacianSmoothing(void);
void createContours(void);
void testCoins(void);
void clearScreen(void);
int intInput(const int lower, const int upper, const char *text);
double doubleInput(const double lower, const double upper, const char *text);

// File input and memory management functions: (inputandmemory.c)
struct FloatSurface *inputDepthModel(void);
struct Coin *createCoin(const int radius, const char trim);
void freeFloatSurface(struct FloatSurface *input);
void freeCoin(struct Coin *penny);
float** createFloatArray(const int cols, const int rows);
int** createIntArray(const int cols, const int rows);
void freeFloatArray(float **array, const int rows);
void freeIntArray(int **array, const int rows);

// Rolling Coin surface smoothing (safe for navigation): (rolling_coin_smoothing.c)
void coinRollSurface(struct FloatSurface *src, struct Coin *penny);
char getShoalestDepthOnCoin(struct FloatSurface *src, struct Coin *penny, float
**shoalest, const int row_index, const int col_index);
void maxFilterSurface(struct FloatSurface *src);

// Laplacian surface smoothing (safe for navigation): (laplacian_smoothing.c)
void smoothLaplacian(const int iterations, struct FloatSurface *src);
char isNodata(struct FloatSurface *src, int rowindex, int colindex);
float getInterpolatedDepth(struct FloatSurface *src, int row, int col);
float getSafeSmoothDepth(struct FloatSurface *src, int row, int col);

// 2D Rolling Coin "LimitSurface" functions: (limitsurface_manipulation.c).
struct LimitSurface *getLimitSurface(struct FloatSurface *surface);
```

```
void makeGoNoGo(const struct FloatSurface *src, int **dst, const double limit);
void bufferShoals(int **src, int **dst, const int rows, const int cols);
void rollCoin(struct LimitSurface *src, struct Coin *penny, int **bufferedarray, const
int valdco);
char checkCoin(int **src, const struct Coin *penny, const int row_index, const int
col_index, const int total_rows, const int total_cols);
void makeLimitSurfacePositive(struct LimitSurface *src);
double parseDepthLimit(const int valdco, char contourType);
void freeLimitSurface(struct LimitSurface *input);

// File output functions: (fileoutput.c)
char *parsePath(char *inputfp, char *addon);
int *convertIntArray(struct LimitSurface *input);
float *convertFloatArray(struct FloatSurface *input);
void writeLimitSurfaceToFile(struct LimitSurface *input, const char calculate_contours);
void writeSurfaceToFile(struct FloatSurface *input);

// Development helper functions (infoprinters.c), these are not strictly necessary:
void printFloatSurfaceInfo(struct FloatSurface *input);
void printLimitSurfaceInfo(struct LimitSurface *input);
void printCoin(struct Coin *penny);



//
//            - - - Main functions (main.c) - - -
//


#include "bathymetrictools.h"

/*
*   This file contains:
*   - Main function
*   - Input parameter functions
*   - Process functions to export contours and
*     smoothed surfaces
*/


/*
*   Simple main function:
*/
int main() {

    clearScreen();

    while (1) {
        // Print menu:
        printf("Bathymetric surface tools - select option:\n\
            \n    1. Rolling Coin surface smoothing (Navigationally safe 2.5D surface
smoothing) \
            \n    2. Laplacian surface smoothing    (Navigationally safe 2.5D surface
smoothing)\
            \n    3. Generate contours based on depth model (original 2D Rolling Coin
method) \
            \n    4. Test coin radius and trimming \
            \n    5. Clear screen \
            \n    6. Exit \
            \n\n");

        char action = intInput(1, 6, "Choose action: ");

        if (action == 1) {
            rollingCoinSmoothing();

        }   else if (action == 2) {
```

```
            laplacianSmoothing();

        }   else if (action == 3) {
            createContours();

        }   else if (action == 4) {
            testCoins();

        }   else if (action == 5) {
            clearScreen();

        }   else if (action == 6) {
            clearScreen();
            break;
        }
    }

    return 0;
}


/*
*   Rolling Coin surface smoothing process.
*   - 2.5D
*   - Safe for navigation
*   - With or without shoal buffering (3x3 max filter)
*/
void rollingCoinSmoothing(void) {
    clearScreen();
    printf("Rolling Coin surface smoothing (2.5D):\n");

    // Import data and build a "FloatSurface":
    struct FloatSurface *surf = inputDepthModel();

    // Use shoal buffering?
    char buffering = intInput(0, 1, "Buffer shoals to ensure contour safety? (0: No
buffering, 1: Buffer shoals): ");

    // Build a "Coin":
    int coin_r = intInput(1, 50, "Enter coin radius in cells (for example 5): ");
    char trim = intInput(0, 1, "Trim coin edges? (0: No trim, 1: Trim outer edges): ");
    struct Coin *penny = createCoin(coin_r, trim);

    // Buffer shoals if buffering is selected:
    if (buffering == 1) {
        printf("\nBuffering shoals..\n");
        maxFilterSurface(surf);
    }

    // Generalize/smooth surface:
    printf("Smoothing surface..\n");
    fflush(stdout);
    coinRollSurface(surf, penny);

    // Export file (GeoTIFF format):
    printf("Exporting file.. ");
    fflush(stdout);
    writeSurfaceToFile(surf);

    // Free allocated memory:
    printf("Freeing memory.. ");
    freeCoin(penny);
    freeFloatSurface(surf);
    printf("Done\n\n");
}
```

```
/*
*   Iterative Laplacian interpolation based surface smoothing.
*   Safe for navigation.
*/
void laplacianSmoothing(void) {
    clearScreen();
    printf("Laplacian surface smoothing (2.5D):\n");

    // Import data and build a "FloatSurface":
    struct FloatSurface *surf = inputDepthModel();

    // Number of iterations:
    int iterations = intInput(1, 500, "Enter number of smoothing iterations (1 - 500):
");

    // Smooth:
    printf("Smoothing surface.. ");
    fflush(stdout);
    smoothLaplacian(iterations, surf);
    printf("Done\n");
    fflush(stdout);

    // Export file (GeoTIFF format):
    printf("Exporting file.. ");
    fflush(stdout);
    writeSurfaceToFile(surf);

    // Free allocated memory:
    printf("Freeing memory.. ");
    freeFloatSurface(surf);
    printf("Done\n\n");
}


/*
*   Original Rolling Coin 2D functionality.
*   - Creates a generalized (smoothed) contour limit surface
*   - Contour limits will be safe for navigation
*/
void createContours(void) {
    const int contourlist[] = {0, -3, -6, -10, -15, -20, -30, -50, -100, -150, -200, -
250, -300};  // List of possible (FTA production) contours
    const char lenlist = 13;
    double depthlimit = 0;

    clearScreen();
    printf("Contour line generation tool (Rolling Coin):\n");

    // Import data and build a "FloatSurface":
    struct FloatSurface *surf = inputDepthModel();

    // Select contour type:
    char contourtype = intInput(1, 2, "Select contour type: \
        \n  1: True depth limits (e.g. 6.00 m, 10.00 m, 20.00 m)\
        \n  2: IHO Navigation chart contour limits (e.g. 6.09 m, 10.09 m, 50.99 m) \
        \n: ");

    // Build Coin:
    int coin_r = intInput(1, 50, "\nEnter coin radius in cells (for example 5): ");
    char trim = intInput(0, 1, "Trim coin edges? (0:No trim, 1:Trim outer edges): ");
    struct Coin *penny = createCoin(coin_r, trim);

    // Create "LimitSurface":
    struct LimitSurface *limitsurf = getLimitSurface(surf);

    // Build 2 extra int** arrays to hold processing stages:
    int **nogoarray = createIntArray(limitsurf->cols, limitsurf->rows);
```

```
    int **bufferarray = createIntArray(limitsurf->cols, limitsurf->rows);

    // Loop trough contour limits:
    for (int i = 0; i < lenlist; i++) {
        // Parse depth limit based on valdco and contour type:
        depthlimit = parseDepthLimit(contourlist[i], contourtype);

        printf("\nCalculating contour: %4d m (depth limit: %7.2f m)", contourlist[i],
depthlimit);

        // Update go/nogo array:
        makeGoNoGo(surf, nogoarray, depthlimit);

        // Update buffered array:
        bufferShoals(nogoarray, bufferarray, limitsurf->rows, limitsurf->cols);

        // Roll Coin and update LimitSurface array:
        rollCoin(limitsurf, penny, bufferarray, contourlist[i]);
    }

    // Write surface to file:
    makeLimitSurfacePositive(limitsurf);      // Negate surface (to positive) to push
contours to safe side
    writeLimitSurfaceToFile(limitsurf, 1);    // 1 = export contours using command line
gdal_contour

    // Free all allocated memory:
    freeIntArray(nogoarray, limitsurf->rows);
    freeIntArray(bufferarray, limitsurf->rows);
    freeCoin(penny);
    freeFloatSurface(surf);
    freeLimitSurface(limitsurf);
}


/*
*   Function to test coin creation.
*   - Prints coin on screen
*   - Development aid
*/
void testCoins(void) {
    clearScreen();
    printf("Test coins:\n");
    int coin_r = intInput(1, 50, "Enter coin radius in cells (for example 5): ");
    char trim = intInput(0, 1, "Trim coin edges? (0: No trim, 1: Trim outer edges): ");
    struct Coin *penny = createCoin(coin_r, trim);
    printCoin(penny);
    freeCoin(penny);
    printf("\n");
}


/*
*   Function to clear the screen.
*   - Somewhat portable
*   - Current solution is somewhat ugly
*/
void clearScreen(void) {
    system("clear||cls");
}


/*
*   Function for integer input:
*/
int intInput(const int lower, const int upper, const char *text) {
    char st[40];
```

```
    int result = 0;
    int number;
    printf("%s", text);

    while (result != 1 || number < lower || number > upper) {
        fgets(st, sizeof(st), stdin);
        result = sscanf(st, "%d", &number);
        if (result != 1 || (number < lower || number > upper)){
            printf("Invalid input. Enter a number between [%d, %d]: ", lower, upper);
        }
    }

    return number;
}


/*
*   Function for floating point input:
*/
double doubleInput(const double lower, const double upper, const char *text) {
    char st[40];
    double number;
    int result = 0;
    printf("%s", text);

    while (result != 1 || number < lower || number > upper) {
        fgets(st, sizeof(st), stdin);
        result = sscanf(st, "%lf", &number);
        if (result != 1 || (number < lower || number > upper)){
            printf("Invalid input. Enter a number between [%f, %f]: ", lower, upper);
        }
    }

    return number;
}



//
//            - - - File input, Struct Surface, memory management (inputandmemory.c) - -
-
//


#include "bathymetrictools.h"

/*
*   This file contains:
*   - File input functions
*   - Stuctured datatype builders
*   - Memory management functions: allocates and frees
*/


/*
*   Builds a structured datatype (type "FloatSurface") variable from input depth model.
*   - Allocates memory and populates struct
*   - Returns pointer to FloatSurface
*/
struct FloatSurface *inputDepthModel(void) {
    // Stage 1: Get file path
    int len = 1000;                                                    // Space for
1000 characters
    char filepath[len];
    printf("\nEnter full filepath to bathymetric surface (e.g.
/users/john/desktop/test.bag):\n");
    fgets(filepath, len, stdin);
```

```
    len = strlen(filepath);
    filepath[len-1] = '\0';

    // Stage 2: Open dataset
    GDALDatasetH dataset = NULL;
    GDALRasterBandH band;
    GDALAllRegister();                                              // Register all
GDAL drivers
    int success;

    if (access(filepath, R_OK|W_OK) != -1) {                        // Check that
file exists (read & write permissions ok)
        dataset = GDALOpen(filepath, GA_ReadOnly);                  // Try to open
dataset
    } else {
        printf("File read error. Recheck file path.\nExiting.\n");
        exit(0);
    }

    if (dataset == NULL) {
        printf("File read error. Recheck file path.\nExiting.\n");
        exit(0);
    }   else{
        printf("File read successful. Building surface..");
    }

    band = GDALGetRasterBand(dataset, 1);                           // Get raster
band

    // Allocate and populate struct:
    struct FloatSurface *ret = calloc(1, sizeof(struct FloatSurface));  // Allocate
memory for FloatSurface
    len = strlen(filepath);                                         // Get filepath
length
    ret->inputfp = calloc(len + 1, 1);                              // Allocate
memory for null character too (+1)
    ret->inputfp = strcpy(ret->inputfp, filepath);                  // Copy filepath
string to struct

    const char *src_projection =  GDALGetProjectionRef(dataset);    // Get projec-
tion information
    len = strlen(src_projection);
    ret->projection = calloc(len + 1, 1);                           // +1 for null
character
    ret->projection = strcpy(ret->projection, (char*)src_projection);   // Set projec-
tion information

    ret->geotransform = calloc(6, sizeof(double));                  // Allocate
memory for 6 doubles
    GDALGetGeoTransform(dataset, ret->geotransform);                // Set geotrans-
form parameters

    ret->nodata = GDALGetRasterNoDataValue(band, &success);         // Set nodata
value
    ret->rows = GDALGetRasterBandYSize(band);                       // Set row count
    ret->cols = GDALGetRasterBandXSize(band);                       // Set column
count

    // Allocate memory & get depth model data as an array (float**):
    float **array = calloc(ret->rows, sizeof(float *));             // Allocate
memory for rows

    for (int row = 0; row < ret->rows; row++ ) {
        array[row] = calloc(ret->cols, sizeof(float));              // Allocate
memory for columns
```

```
        char err = GDALReadBlock(band, 0, row, array[row]);             // Read input
depth model row by row
        if (err != 0) {
            printf("File read not successful. GDALReadBlock returned a warning or an
error.\n");
        }
    }

    ret->array = array;                                                 // Store data
array pointer to Struct

    GDALClose(dataset);                                                 // Data is now
stored in struct, file can be closed
    printf("Done\n\n");
    return ret;                                                         // Return struct
pointer
}


/*
*   Builds a Coin.
*   - Allocates memory, returns a pointer to Coin
*   - Input parameters:
*       - Coin radius (diameter = 2r+1)
*       - Trim flag:
*           - 0: No trim
*           - 1: Trim outer edges (diameter -= 2)
*/
struct Coin *createCoin(const int radius, const char trim) {
    struct Coin *ret = calloc(1, sizeof(struct Coin));
    int diameter = 1 + (2 * radius);  // Diameter
    int trimmeddiam;

    // Trimmed or not trimmed:
    if (trim == 1) {
        trimmeddiam = diameter - 2;  // Diameter of trimmed coin
    }   else {
        trimmeddiam = diameter;
    }

    ret->originalRadius = radius;
    ret->originalDiameter = diameter;
    ret->trimmedDiameter = trimmeddiam;
    ret->maxIndex = trimmeddiam - 1;

    // Array building:
    int **untrimmed = createIntArray(diameter, diameter);
    int **trimmed = createIntArray(trimmeddiam, trimmeddiam);

    // Create coin:
    for (int i = 0; i < diameter; i++) {
        for (int j = 0; j < diameter; j++) {
            int distY = i - radius;
            int distX = j - radius;
            if ((distX * distX) + (distY * distY) <= (radius * radius)) {  // Round
coin, value 1 = on coin
                untrimmed[i][j] = 1;
            }   else {
                untrimmed[i][j] = 0;  // Not on coin
            }
        }
    }

    // Trim coin if selected:
    if (trim == 1) {
        for (int i = 0; i < trimmeddiam; i++) {
            for (int j = 0; j < trimmeddiam; j++) {
```

```
                trimmed[i][j] = untrimmed[i + 1][j + 1];
            }
        }

        freeIntArray(untrimmed, diameter);
        ret->array = trimmed;

    }   else {
        freeIntArray(trimmed, diameter);
        ret->array = untrimmed;
    }

    return ret;
}


/*
*   Frees all allocated memory of parameter FloatSurface.
*   All FloatSurfaces must be freed in order to avoid memory leaks.
*/
void freeFloatSurface(struct FloatSurface *input) {
    float **array = input->array;
    int rows = input->rows;
    free(input->inputfp);               // Free input filepath
    free(input->projection);            // Free CRS WKT string
    free(input->geotransform);          // Free geotrans parameters
    freeFloatArray(array, rows);        // Free data array
    free(input);                        // Free struct
}


/*
*   Frees all allocated memory of parameter Coin.
*   All Coins must be freed in order to avoid memory leaks.
*/
void freeCoin(struct Coin *penny) {
    int **array = penny->array;
    int rows = penny->maxIndex + 1;
    freeIntArray(array, rows);          // Free array
    free(penny);                        // Free Struct
}


/*
*   - Allocates memory for (2D) float** array of given size
*   - Returns a pointer to array
*/
float** createFloatArray(const int cols, const int rows) {
    float **ret;                                    // Array pointer to be returned
    ret = calloc(rows, sizeof(float*));             // Allocate memory for rows

    for (int row = 0; row < rows; row++) {
        ret[row] = calloc(cols, sizeof(float));     // Allocate memory for columns
    }

    return ret;
}


/*
*   - Allocates memory for (2D) int** array of given size
*   - Returns a pointer to array
*/
int** createIntArray(const int cols, const int rows) {
    int **ret;                                      // Array pointer to be returned
    ret = calloc(rows, sizeof(int*));               // Allocate memory for rows
```

```
    for (int row = 0; row < rows; row++) {
        ret[row] = calloc(cols, sizeof(int));       // Allocate memory for columns
    }

    return ret;
}


/*
*   Frees allocated memory of 2D float (float**) array
*/
void freeFloatArray(float **array, const int rows) {
    for (int i = 0; i < rows; i++) {
        free(array[i]);
    }
    free(array);
}


/*
*   Frees allocated memory of 2D integer (int**) array
*/
void freeIntArray(int **array, const int rows) {
    for (int i = 0; i < rows; i++) {
        free(array[i]);
    }
    free(array);
}



//
//            - - - Rolling Coin 2.5D surface manipulation (rolling_coin_smoothing.c) -
- -
//


#include "bathymetrictools.h"

/*
*   This file contains:
*   - Navigationally safe Rolling Coin surface manipulation (2.5D) functions
*   - Coin is defined by structured datatype "Coin"
*   - Very fast implementation
*/


/*
*   Surface manipulation (~smoothing) function.
*   - Iterates over cells
*   - Modifies the surface
*   - Memory management and no data handling
*/
void coinRollSurface(struct FloatSurface *src, struct Coin *penny) {
    const int radius = penny->originalRadius - 1;  // Valid indexes of trimmed coin
    float nodata = src->nodata;
    char coinok = 0;
    float *shoalest;
    float placeholder = -999.0;
    shoalest = &placeholder;

    // Create new float** (2D) array:
    float **temp = createFloatArray(src->cols, src->rows);

    // Initialize all cell to 10 000 (meters):
    for (int row = 0; row < src->rows; row++) {
        for (int col = 0; col < src->cols; col++) {
```

```
            temp[row][col] = 10000.0;
        }
    }

    // Iterate and smooth surface:
    for (int row = 0; row < src->rows; row++) {
        for (int col = 0; col < src->cols; col++) {

            coinok = getShoalestDepthOnCoin(src, penny, &shoalest, row, col);
// Check Coin area

            if (coinok == 1) {
                for (int row_coin = -radius; row_coin <= radius; row_coin++) {
// Coin Row indexes [-radius, radius]
                    for (int col_coin = -radius; col_coin <= radius; col_coin++) {
// Coin Col indexes [-radius, radius]

                        if (penny->array[row_coin + radius][col_coin + radius] == 1) {
// On the coin
                            if (temp[row + row_coin][col + col_coin] > *shoalest) {
// If current depth of cell is shoaler than shoalest
                                temp[row + row_coin][col + col_coin] = *shoalest;
// "Press" depth to coin area
                            }
                        }
                    }
                }
            }
        }
    }

    // Restore original nodata:
    for (int row = 0; row < src->rows; row++) {
        for (int col = 0; col < src->cols; col++) {
            if ((fabs(src->array[row][col] - nodata) < EPSILON)) {  // Value == nodata
                temp[row][col] = nodata;
            }
        }
    }

    // Free memory allocated for temp array:
    float **destruct = src->array;          // Store original data array pointer
    src->array = temp;                      // Replace original data array with smoothed
data array
    freeFloatArray(destruct, src->rows);    // Free original data array
}


/*
*   Check coin area, finds shoalest depth (maximum elevation).
*   - Returns either 0 or 1 depending on whether or not the coin fits inside data array
*       - Edge effects not taken into account in this implementation - returns 0 near
edges
*   - Another "return value", shoalest depth, is saved to "shoalest" variable that is
passed
*     as a parameter. In case the primary return value is 0, the shoalest depth value
will be NULL.
*/
char getShoalestDepthOnCoin(struct FloatSurface *src, struct Coin *penny, float
**shoalest, const int row_index, const int col_index) {
    const int radius = penny->originalRadius - 1;
    const int min_row_index = radius;   // Smallest index in src array that can be
checked
    const int min_col_index = radius;   // Smallest index in src array that can be
checked
    const int max_row_index = src->rows - 1 - radius;
    const int max_col_index = src->cols - 1 - radius;
```

```
    float shoalholder = -99999.0;       // initial for shoalest

    // Return false (0) if coin does not fit inside data array:
    if (row_index < min_row_index || row_index > max_row_index || col_index <
min_col_index || col_index > max_col_index) {
        shoalest = NULL;    // Shoalest = NULL
        return 0;           // Cannot check, False
    }

    // Check the coin area:
    for (int row_coin = -radius; row_coin <= radius; row_coin++) {
        for (int col_coin = -radius; col_coin <= radius; col_coin++) {

            if (penny->array[radius + row_coin][radius + col_coin] == 1) {
// If cell is on coin --> check
                if (src->array[row_index + row_coin][col_index + col_coin] > shoalhold-
er) {
                    shoalholder = src->array[row_index + row_coin][col_index +
col_coin];
                    *shoalest = &src->array[row_index + row_coin][col_index + col_coin];
// Update shoalest value
                }
            }
        }
    }

    return 1;   // Coin OK
}


/*
*   Buffers shoals by one cell to all directions.
*   - Used to expand shoals to ensure the safety of depth contours
*   - Uses a 3 x 3 cell focal maximum filter:
*
*                   + + +
*                   + X +
*                   + + +
*
*   - Cell X gets the shoalest value of neighborhood
*   - Neighborhood is marked with "+"
*       - If X is shoalest, cell value doesn't change
*/
void maxFilterSurface(struct FloatSurface *src) {
    float nodata = src->nodata;
    float max_elev = -15000.0;  // Placeholder for shoalest depth
    const float placeholder = max_elev;
    int lenlist = 8;
    float neighborhood[lenlist];

    // Create new float** (2D) array:
    float **temp = createFloatArray(src->cols, src->rows);

    // Make a temporary copy of the original data array:
    for (int row = 0; row < src->rows; row++) {
        for (int col = 0; col < src->cols; col++) {
            temp[row][col] = src->array[row][col];
        }
    }

    // Iterate over cells and filter surface:
    for (int row = 0; row < src->rows; row++) {
        for (int col = 0; col < src->cols; col++) {

            // Reset max_elev to placeholder value:
            max_elev = placeholder;
```

```
            // Initialize / reset neighborhood array:
            lenlist = 8;
            for (int i = 0; i < lenlist; i++) {
                neighborhood[i] = placeholder;
            }

            if (row == 0) {                                 // Top row
                if (col == 0) {                                 // Top-left
                    neighborhood[0] = temp[row][col + 1];
                    neighborhood[1] = temp[row + 1][col];
                    neighborhood[2] = temp[row + 1][col + 1];
                    lenlist = 3;

                }   else if (col == src->cols - 1) {            // Top-right
                    neighborhood[0] = temp[row][col - 1];
                    neighborhood[1] = temp[row + 1][col];
                    neighborhood[2] = temp[row + 1][col -1];
                    lenlist = 3;

                }   else {                                      // Between corners
                    neighborhood[0] = temp[row][col - 1];
                    neighborhood[1] = temp[row][col + 1];
                    neighborhood[2] = temp[row + 1][col];
                    neighborhood[3] = temp[row + 1][col - 1];
                    neighborhood[4] = temp[row + 1][col + 1];
                    lenlist = 5;
                }

            }   else if (row == src->rows - 1) {            // Bottom row
                if (col == 0) {                                 // Bottom-left
                    neighborhood[0] = temp[row][col + 1];
                    neighborhood[1] = temp[row - 1][col];
                    neighborhood[2] = temp[row - 1][col + 1];
                    lenlist = 3;

                }   else if (col == src->cols - 1) {            // Bottom-right
                    neighborhood[0] = temp[row][col - 1];
                    neighborhood[1] = temp[row - 1][col];
                    neighborhood[2] = temp[row - 1][col - 1];
                    lenlist = 3;

                }   else {                                      // Between corners
                    neighborhood[0] = temp[row][col - 1];
                    neighborhood[1] = temp[row][col + 1];
                    neighborhood[2] = temp[row - 1][col];
                    neighborhood[3] = temp[row - 1][col - 1];
                    neighborhood[4] = temp[row - 1][col + 1];
                    lenlist = 5;
                }

            }   else {                                      // Rows in between top and
bottom
                if (col == 0) {                                 // Left edge
                    neighborhood[0] = temp[row][col + 1];
                    neighborhood[1] = temp[row + 1][col];
                    neighborhood[2] = temp[row - 1][col];
                    neighborhood[3] = temp[row - 1][col + 1];
                    neighborhood[4] = temp[row + 1][col + 1];
                    lenlist = 5;

                }   else if (col == src->cols - 1) {            // Right edge
                    neighborhood[0] = temp[row][col - 1];
                    neighborhood[1] = temp[row + 1][col];
                    neighborhood[2] = temp[row - 1][col];
                    neighborhood[3] = temp[row - 1][col - 1];
                    neighborhood[4] = temp[row + 1][col - 1];
                    lenlist = 5;
```

```
                }   else {                                          // Between edges
                    neighborhood[0] = temp[row - 1][col - 1];
                    neighborhood[1] = temp[row - 1][col];
                    neighborhood[2] = temp[row - 1][col + 1];
                    neighborhood[3] = temp[row][col - 1];
                    neighborhood[4] = temp[row][col + 1];
                    neighborhood[5] = temp[row + 1][col - 1];
                    neighborhood[6] = temp[row + 1][col];
                    neighborhood[7] = temp[row + 1][col + 1];
                    lenlist = 8;
                }
            }

            // Get max elevation (shoalest depth):
            for (int i = 0; i < lenlist; i++) {
                if (neighborhood[i] > max_elev) {
                    max_elev = neighborhood[i];
                }
            }

            // Update source array cell values if max_elev is shoaler and max_elev is
not nodata:
            if (max_elev > src->array[row][col] && fabs(max_elev - nodata) > EPSILON &&
fabs(src->array[row][col] - nodata) > EPSILON) {
                src->array[row][col] = max_elev;
            }
        }
    }

    // Free temporary data array
    freeFloatArray(temp, src->rows);
}



//
//            - - - Navigationally safe 2.5D Laplacian surface smoothing (laplaci-
an_smoothing.c) - - -
//


#include "bathymetrictools.h"

/*
*   This file contains:
*   - Navigationally safe Laplace interpolation implementation to smooth surfaces itera-
tively
*   - Neighborhood is only the immediate neighboring cells, nodata is handled as a miss-
ing neighbor
*   - Weights calculated from spatial resolutions in X- and Y-directions
*
*   If spatial resolution X = Y, the used neighborhood (convolution kernel) will be
like:
*
*           |0   1   0|
*   1/4  *  |1   0   1|
*           |0   1   0|
*/



/*
*   Controls the iterative smoothing process.
*   - Iterates over surface cells
*   - Memory management
*/
void smoothLaplacian(const int iterations, struct FloatSurface *src) {
```

```
    const double nodata = src->nodata;

    // Build extra array to hold smoothed surface (type float**):
    float **smooth_array = createFloatArray(src->cols, src->rows);
    float **holder = NULL;  // Pointer placeholder

    // Iterate and smooth surface N times:
    for (int i = 0; i < iterations; i++) {
        for (int row = 0; row < src->rows; row++) {
            for (int col = 0; col < src->cols; col++) {
                if (fabs(src->array[row][col] - nodata) > EPSILON) {
                    smooth_array[row][col] = getSafeSmoothDepth(src, row, col);
                }   else {
                    smooth_array[row][col] = nodata;
                }
            }
        }

        // Swap surface data array:
        holder = src->array;        // Store pointer temporarily
        src->array = smooth_array;  // Change surface struct data array
        smooth_array = holder;      // Use the same temporary array again
    }

    // Free memory of the temporary array:
    freeFloatArray(smooth_array, src->rows);
}


/*
*   Helper function to check if given cell holds a No Data value.
*   - Returns 1 if cell value == No Data
*   - Returns 0 if cell value != No Data
*/
char isNodata(struct FloatSurface *src, int rowindex, int colindex) {
    if (fabs(src->array[rowindex][colindex] - src->nodata) > EPSILON) {  // != NO DATA
        return 0;
    }  else {
        return 1;
    }
}


/*
*   Interpolates a value based on immediate neighborhood.
*   - A minimum of 2 valid (NoData handled as not valid)
*     neighbor cells is needed to interpolate a value
*   - Handles all cells, note behaviour in corners and borders
*/
float getInterpolatedDepth(struct FloatSurface *src, int row, int col) {
    const double xres = fabs(src->geotransform[1]);  // W-E grid cell spatial resolution
    const double yres = fabs(src->geotransform[5]);  // N-S grid cell spatial resolution
(negative value)

    // Get kernel weights (Wi = dVi / di)
    const double xWeight = yres / xres;  // --> X-direction: Yres / Xres
    const double yWeight = xres / yres;  // --> Y-direction: Xres / Yres

    double weightSum = 0.0;
    double list[4] = {0.0, 0.0, 0.0, 0.0};
    double sum = 0;
    int count = 0;

    // Top left corner:
    if (row == 0 && col == 0) {
        if (isNodata(src, row, col + 1) == 0) {
            list[count] = src->array[row][col + 1] * xWeight;
```

```
        weightSum += xWeight;
        count ++;
    }
    if (isNodata(src, row + 1, col) == 0) {
        list[count] = src->array[row + 1][col] * yWeight;
        weightSum += yWeight;
        count ++;
    }

    // Top right corner:
}   else if (row == 0 && col == src->cols - 1) {
    if (isNodata(src, row, col - 1) == 0) {
        list[count] = src->array[row][col - 1] * xWeight;
        weightSum += xWeight;
        count ++;
    }
    if (isNodata(src, row + 1, col) == 0) {
        list[count] = src->array[row + 1][col] * yWeight;
        weightSum += yWeight;
        count ++;
    }

    // Lower left corner:
}   else if (row == src->rows - 1 && col == 0) {
    if (isNodata(src, row, col + 1) == 0) {
        list[count] = src->array[row][col + 1] * xWeight;
        weightSum += xWeight;
        count ++;
    }
    if (isNodata(src, row - 1, col) == 0) {
        list[count] = src->array[row - 1][col] * yWeight;
        weightSum += yWeight;
        count ++;
    }

    // Lower right corner:
}   else if (row == src->rows - 1 && col == src->cols - 1) {

    if (isNodata(src, row, col - 1) == 0) {
        list[count] = src->array[row][col - 1] * xWeight;
        weightSum += xWeight;
        count ++;
    }
    if (isNodata(src, row - 1, col) == 0) {
        list[count] = src->array[row - 1][col] * yWeight;
        weightSum += yWeight;
        count ++;
    }

    // Left border:
}   else if (col == 0) {
    if (isNodata(src, row, col + 1) == 0) {
        list[count] = src->array[row][col + 1] * xWeight;
        weightSum += xWeight;
        count ++;
    }
    if (isNodata(src, row - 1, col) == 0) {
        list[count] = src->array[row - 1][col] * yWeight;
        weightSum += yWeight;
        count ++;
    }
    if (isNodata(src, row + 1, col) == 0) {
        list[count] = src->array[row + 1][col] * yWeight;
        weightSum += yWeight;
        count ++;
    }
```

```
    // Right border:
}   else if (col == src->cols - 1) {
    if (isNodata(src, row, col - 1) == 0) {
        list[count] = src->array[row][col - 1] * xWeight;
        weightSum += xWeight;
        count ++;
    }
    if (isNodata(src, row - 1, col) == 0) {
        list[count] = src->array[row - 1][col] * yWeight;
        weightSum += yWeight;
        count ++;
    }
    if (isNodata(src, row + 1, col) == 0) {
        list[count] = src->array[row + 1][col] * yWeight;
        weightSum += yWeight;
        count ++;
    }

    // Top row:
}   else if (row == 0) {
    if (isNodata(src, row + 1, col) == 0) {
        list[count] = src->array[row + 1][col] * yWeight;
        weightSum += yWeight;
        count ++;
    }
    if (isNodata(src, row, col - 1) == 0) {
        list[count] = src->array[row][col - 1] * xWeight;
        weightSum += xWeight;
        count ++;
    }
    if (isNodata(src, row, col + 1) == 0) {
        list[count] = src->array[row][col + 1] * xWeight;
        weightSum += xWeight;
        count ++;
    }

    // Bottom row:
}   else if (row == src->rows - 1) {
    if (isNodata(src, row - 1, col) == 0) {
        list[count] = src->array[row - 1][col] * yWeight;
        weightSum += yWeight;
        count ++;
    }
    if (isNodata(src, row, col - 1) == 0) {
        list[count] = src->array[row][col - 1] * xWeight;
        weightSum += xWeight;
        count ++;
    }
    if (isNodata(src, row, col + 1) == 0) {
        list[count] = src->array[row][col + 1] * xWeight;
        weightSum += xWeight;
        count ++;
    }

    // All other cells (in the middle):
}   else {
    if (isNodata(src, row - 1, col) == 0) {
        list[count] = src->array[row - 1][col] * yWeight;
        weightSum += yWeight;
        count ++;
    }
    if (isNodata(src, row + 1, col) == 0) {
        list[count] = src->array[row + 1][col] * yWeight;
        weightSum += yWeight;
        count ++;
    }
    if (isNodata(src, row, col - 1) == 0) {
```

```
            list[count] = src->array[row][col - 1] * xWeight;
            weightSum += xWeight;
            count ++;
        }
        if (isNodata(src, row, col + 1) == 0) {
            list[count] = src->array[row][col + 1] * xWeight;
            weightSum += xWeight;
            count ++;
        }
    }


    // If less than 2 valid (!= No Data) neighbors, do not interpolate but return cell
value itself:
    if (count < 2) {
        return src->array[row][col];

    }   else {  // Return weighted mean of neighbours

        // Calculate sum of cell values:
        for (int i = 0; i < count; i++) {
            sum += list[i];
        }

        return (float)(sum / weightSum);
    }
}


/*
*   Function to guarantee the safety of the process.
*   - Gets an interpolated value based on neighborhood and
*     compares it to the original cell value
*   - Returns the one that is shoaler
*/
float getSafeSmoothDepth(struct FloatSurface *src, int row, int col) {
    const float z = src->array[row][col];                           // Original value
    const float estimate = getInterpolatedDepth(src, row, col);     // Interpolated val-
ue

    // Return safer value:
    if (fabs(estimate) < fabs(z)) {
        return estimate;
    }   else {
        return z;
    }
}



//
//            - - - Rolling Coin contouring tool, 2D (limitsurface_manipulation.c) - - -
//


#include "bathymetrictools.h"

/*
*   This file contains:
*   - 2D array manipulation functions for LimitSurfaces
*   - Original 2D Rolling Coin functionality
*/


/*
*   Builds a structured datatype (type "LimitSurface") variable from input surface
*   - Output surface will be naive NODATA only
```

```
*   - Allocates memory and populates struct
*   - Returns a pointer to LimitSurface
*/
struct LimitSurface *getLimitSurface(struct FloatSurface *surface) {
    struct LimitSurface *ret = calloc(1, sizeof(struct LimitSurface));  // Allocate
memory for LimitSurface
    char *temp;
    int len;

    temp = surface->projection;
    len = strlen(temp);
    ret->projection = calloc(len + 1, 1);  // +1 for null character
    ret->projection = strcpy(ret->projection, temp);  // Set projection information

    ret->geotransform = calloc(6, sizeof(double));
    memcpy(ret->geotransform, surface->geotransform, 6 * sizeof(double));  // Set ge-
otransform parameters

    ret->rows = surface->rows;  // Set row count
    ret->cols = surface->cols;  // Set column count

    temp = surface->inputfp;
    len = strlen(temp);
    ret->inputfp = calloc(len + 1, 1);  // +1 for null character
    ret->inputfp = strcpy(ret->inputfp, temp);  // Set input filepath

    ret->array = createIntArray(surface->cols, surface->rows);  // Allocate memory for
array

    // Fill data array with NO DATA:
    for (int row = 0; row < surface->rows; row++) {
        for (int col = 0; col < surface->cols; col++) {
            ret->array[row][col] = NODATA;
        }
    }

    return ret;
}


/*
*   Builds GO/NOGO/NODATA array. Array will be built in
*   "dst" (passed as a parameter) array. Uses absolute values.
*/
void makeGoNoGo(const struct FloatSurface *src, int **dst, const double limit) {
    for (int row = 0; row < src->rows; row++) {
        for (int col = 0; col < src->cols; col++) {
            if (fabs(src->array[row][col] - src->nodata) < EPSILON) {   // Value == no-
data
                dst[row][col] = NODATA;
            }   else if (fabs(src->array[row][col]) > fabs(limit)) {    // Deeper than
limit
                dst[row][col] = GO;
            }   else {
                dst[row][col] = NOGO;
            }
        }
    }
}


/*
*   Buffers shoals (NO GO cells) one cell to all directions.
*   - Buffering is based on "src" array and is done to "ret" array (passed as parame-
ters)
*   - Output is buffered GO/NOGO/NODATA array
*   - Also ensures that all NO DATA cells remain as NO DATA
```

```
*/
void bufferShoals(int **src, int **ret, const int rows, const int cols) {
    const int colmax = cols - 1;
    const int rowmax = rows - 1;

    // Copy contents from "src" to "ret":
    for (int row = 0; row < rows; row++) {
        for (int col = 0; col < cols; col++) {
            ret[row][col] = src[row][col];
        }
    }

    for (int row = 0; row < rows; row++) {
        for (int col = 0; col < cols; col++) {

            // Test current cell:
            if (src[row][col] == GO) {                      // GO:
                continue;                                   // OK --> next cell

            }   else if (src[row][col] == NODATA) {         // NODATA:
                ret[row][col] = NODATA;                     // Ret = nodata --> next
cell
                continue;

            }   else if (src[row][col] == NOGO) {           // NOGO: Shoal cell --> ex-
pand
                ret[row][col] = NOGO;                       // Set current cell to be
shallow

                if (row == 0) {                             // Top row:
                    if (col == 0) {                         // Top-left corner - 3 di-
rections
                        ret[row][col + 1] = NOGO;
                        ret[row + 1][col] = NOGO;
                        ret[row + 1][col + 1] = NOGO;
                    }   else if (col == colmax) {           // Top-right corner - 3 di-
rections
                        ret[row][col - 1] = NOGO;
                        ret[row + 1][col] = NOGO;
                        ret[row + 1][col - 1] = NOGO;
                    }   else {                              // Top row, in between cor-
ner cells - 5 directions
                        ret[row][col - 1] = NOGO;
                        ret[row][col + 1] = NOGO;
                        ret[row + 1][col] = NOGO;
                        ret[row + 1][col - 1] = NOGO;
                        ret[row + 1][col + 1] = NOGO;
                    }

                }   else if (row > 0 && row < rowmax) {     // Row in between top and
bottom rows:
                        if (col == 0) {                     // Left edge - 5 directions
                            ret[row][col + 1] = NOGO;
                            ret[row + 1][col] = NOGO;
                            ret[row - 1][col] = NOGO;
                            ret[row - 1][col + 1] = NOGO;
                            ret[row + 1][col + 1] = NOGO;
                        }   else if (col == colmax) {       // Right edge - 5 directions
                            ret[row][col - 1] = NOGO;
                            ret[row + 1][col] = NOGO;
                            ret[row - 1][col] = NOGO;
                            ret[row - 1][col - 1] = NOGO;
                            ret[row + 1][col - 1] = NOGO;
                        }   else {                          // In the middle - 8 direc-
tions
                            ret[row - 1][col - 1] = NOGO;
                            ret[row - 1][col] = NOGO;
```

```
                            ret[row - 1][col + 1] = NOGO;
                            ret[row][col - 1] = NOGO;
                            ret[row][col + 1] = NOGO;
                            ret[row + 1][col - 1] = NOGO;
                            ret[row + 1][col] = NOGO;
                            ret[row + 1][col + 1] = NOGO;
                        }

                }   else if (row == rowmax) {               // Bottom row:
                    if (col == 0) {                         // Bottom-left corner - 3
directions
                        ret[row][col + 1] = NOGO;
                        ret[row - 1][col] = NOGO;
                        ret[row - 1][col + 1] = NOGO;
                    }   else if (col == colmax) {           // Bottom-right corner - 3
directions
                        ret[row][col - 1] = NOGO;
                        ret[row - 1][col] = NOGO;
                        ret[row - 1][col - 1] = NOGO;
                    }   else {                              // Bottom row, in between
corner cells - 5 directions
                        ret[row][col - 1] = NOGO;
                        ret[row][col + 1] = NOGO;
                        ret[row - 1][col] = NOGO;
                        ret[row - 1][col - 1] = NOGO;
                        ret[row - 1][col + 1] = NOGO;
                    }
                }
            }
        }
    }

    // Recheck/restore NO DATA:
    for (int row = 0; row < rows; row++) {
        for (int col = 0; col < cols; col++) {
            if (src[row][col] == NODATA) {
                ret[row][col] = NODATA;
            }
        }
    }
}


/*
*   Iterates trough all cells in src_array
*   Possible scenarios:
*   1. No Data
*       -> writes no data to ret array
*   2. No GO (shoal)
*       -> does nothing (all cells No Data or No Go by default)
*   3. Deep enough
*       -> calls coin checking function
*/
void rollCoin(struct LimitSurface *src, struct Coin *penny, int **bufferedarray, const
int valdco) {
    char isclean = 0;                               // "False"
    const int radius = penny->originalRadius - 1;   // Valid indexes of trimmed coin

    // Iterate over cells:
    for (int row = 0; row < src->rows; row++) {
        for (int col = 0; col < src->cols; col++) {
            isclean = 0;

            // Scenario 1 & 2: No Data or No Go
            if (bufferedarray[row][col] == NODATA || bufferedarray[row][col] == NOGO) {
                continue;
```

```
            // Scenario 3: Deep enough, check coin
            }   else if (bufferedarray[row][col] == GO) {
                isclean = checkCoin(bufferedarray, penny, row, col, src->rows, src-
>cols);

                if (isclean == 1) {
                    for (int row_coin = -radius; row_coin <= radius; row_coin++) {
// Coin Row indexes [-radius, radius]
                        for (int col_coin = -radius; col_coin <= radius; col_coin++) {
// Coin Col indexes [-radius, radius]
                            if (penny->array[row_coin + radius][col_coin + radius] == 1)
{      // On the coin
                                if (bufferedarray[row + row_coin][col + col_coin] ==
NODATA) {  // If NODATA cell on coin area
                                    src->array[row + row_coin][col + col_coin] = NODATA;
                                }   else {
                                    src->array[row + row_coin][col + col_coin] = valdco;
// Write valdco to LimitSurface array
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}


/*
*   Returns either true (1) or false (0), depending on the cells tested
*   Cell to be tested is always in the middle of the coin
*   MISSING: Corners, top & bottom rows (edge effects)!
*/
char checkCoin(int **src, const struct Coin *penny, const int row_index, const int
col_index, const int total_rows, const int total_cols) {
    const int radius = penny->originalRadius - 1;
    const int min_row_index = radius;   // Smallest index in src array that can be
checked
    const int min_col_index = radius;   // Smallest index in src array that can be
checked
    const int max_row_index = total_rows - 1 - radius;
    const int max_col_index = total_cols - 1 - radius;

    // Return false (0) if coin does not fit inside data array:
    if (row_index < min_row_index || row_index > max_row_index || col_index <
min_col_index || col_index > max_col_index) {
        return 0;   // Cannot check, return 0 (False)
    }

    // Check coin area:
    for (int row_coin = -radius; row_coin <= radius; row_coin++) {      // Coin Row in-
dexes [-radius, radius]
        for (int col_coin = -radius; col_coin <= radius; col_coin++) {  // Coin Col in-
dexes [-radius, radius]

            // If cell is on coin --> check:
            if (penny->array[radius + row_coin][radius + col_coin] == 1) {  // Indexes:
[radius - radius, radius + radius]
                // Calculate index on src array --> if NO GO or NO DATA, return false
(0):
                if (src[row_index + row_coin][col_index + col_coin] == NOGO) {
                    return 0;   // Shoal, return 0 (False)
                }
            }
        }
    }
```

```
    return 1;   // Coin clean, return 1 (True)
}


/*
*   Negates array values. Used with GDAL contouring tool to
*   draw contours on the safe side of cell boundaries.
*   Makes all cell values positive (absolute values), keeps original NO DATA.
*/
void makeLimitSurfacePositive(struct LimitSurface *src) {
    for (int row = 0; row < src->rows; row++) {
        for (int col = 0; col < src->cols; col++) {
            if (src->array[row][col] != NODATA) {
                src->array[row][col] = abs(src->array[row][col]);
            }
        }
    }
}


/*
*   Parses depth contour limit based on nominal depth contour value
*   and contour type code:
*   contourType = 1:
*       - "True depth" contours
*       - Negates positive input values
*   contourType = 2:
*       - Negates positive input values
*       - Uses IHO Charting specifiaction (S4) contour values:
*           - For example -10.09, -15.09, -20.09, -30.49, -50.99
*/
double parseDepthLimit(const int valdco, const char contourType) {
    double ret = abs(valdco);

    if (valdco == 0) {
        return 0;
    }

    if (contourType == 2) {
        if (ret < 30) {
            ret += 0.09;
        }   else if (ret < 40) {    // 30 m contour
            ret += 0.49;
        }   else {
            ret += 0.99;
        }
    }

    return -1 * ret;
}


/*
*   Frees all allocated memory of parameter LimitSurface.
*   All LimitSurfaces must be freed in order to avoid memory leaks.
*/
void freeLimitSurface(struct LimitSurface *input) {
    int **array = input->array;
    int rows = input->rows;
    free(input->inputfp);               // Free input filepath
    free(input->projection);            // Free CRS WKT string
    free(input->geotransform);          // Free geotrans parameters
    freeIntArray(array, rows);          // Free data array
    free(input);                        // Free Struct
}
```

```
//
//            - - - File output functions (fileoutput.c) - - -
//


#include "bathymetrictools.h"

/*
*   This file contains:
*   - File output related functions
*/


/*
*   Parses output filepaths.
*/
char *parsePath(char *inputfp, char *addon) {
    int leninput = strlen(inputfp);
    int lenaddon = strlen(addon);
    char *ret = calloc(leninput + lenaddon, 1);

    int index = 0;
    while (index < leninput) {
        if (inputfp[index] == '.') {
            ret[index] = '\0';
            break;
        }
        ret[index] = inputfp[index];
        index++;
    }

    strcat(ret, addon);
    return ret;
}


/*
*   Converts LimitSurface 2D int** arrays
*   to (1D) int* arrays.
*/
int *convertIntArray(struct LimitSurface *input) {
    int size = input->rows * input->cols;
    int index = 0;

    int *ret = calloc(size, sizeof(int));

    for (int row = 0; row < input->rows; row++) {
        for (int col = 0; col < input->cols; col++) {
            ret[index] = input->array[row][col];
            index++;
        }
    }

    return ret;
}


/*
*   Converts FloatSurface 2D float** arrays
*   to (1D) float* arrays.
*/
float *convertFloatArray(struct FloatSurface *input) {
    int size = input->rows * input->cols;
    int index = 0;
```

```
    float *ret = calloc(size, sizeof(float));

    for (int row = 0; row < input->rows; row++) {
        for (int col = 0; col < input->cols; col++) {
            ret[index] = input->array[row][col];
            index++;
        }
    }

    return ret;
}


/*
*   Writes LimitSurface to a GeoTIFF file.
*   - Uses GDAL for I/O
*   - If calculate_contours == 1, calculates contours using command line gdal_contour
tool
*/
void writeLimitSurfaceToFile(struct LimitSurface *input, const char calculate_contours)
{
    GDALAllRegister();
    const char *format = "GTiff";
    GDALDriverH driver = GDALGetDriverByName(format);
    char **papszOptions = NULL;
    char *outputfp = parsePath(input->inputfp, "_contour_limits.tif");

    papszOptions = CSLSetNameValue(papszOptions, "COMPRESS", "DEFLATE" );
    GDALDatasetH outdataset = GDALCreate(driver, outputfp, input->cols, input->rows, 1,
GDT_Int32, papszOptions);
    GDALRasterBandH outband = GDALGetRasterBand(outdataset, 1);
    GDALSetGeoTransform(outdataset, input->geotransform);
    GDALSetProjection(outdataset, input->projection);

    int *datalist = convertIntArray(input);


    char ret = GDALRasterIO(outband, GF_Write, 0, 0, input->cols, input->rows, datalist,
input->cols, input->rows, GDT_Int32, 0, 0);

    if (ret != 0) {
        printf("Export was not successful.\n");
    }


    GDALSetRasterNoDataValue(outband, NODATA);

    free(datalist);
    GDALClose(outdataset);
    printf("\n\nSurface exported to file: %s\n\n", outputfp);

    // Quick and dirty way to export contours:
    // Uses command line gdal_contour tool
    if (calculate_contours == 1) {
        // Parse command:
        char *command = calloc(1000, 1);
        char *base = "gdal_contour -f \"ESRI Shapefile\" -i 1 -a VALDCO -b 1 ";
        char *outshape = parsePath(input->inputfp, "_RAW_contours.shp");
        strcat(command, base);
        strcat(command, outputfp);
        strcat(command, " ");
        strcat(command, outshape);

        printf("Creating contours:\n");
        system(command);  // Execute command
        printf("\nContours exported to file: %s\n\n", outshape);
```

```
        free(outshape);
        free(command);
    }

    free(outputfp);
}


/*
*   Writes FloatSurface to a GeoTIFF file.
*   - Uses GDAL for I/O
*/
void writeSurfaceToFile(struct FloatSurface *input) {
    GDALAllRegister();
    const char *format = "GTiff";
    GDALDriverH driver = GDALGetDriverByName(format);
    char **papszOptions = NULL;
    char *outputfp = parsePath(input->inputfp, "_smoothed_surface.tif");

    papszOptions = CSLSetNameValue(papszOptions, "COMPRESS", "DEFLATE" );
    GDALDatasetH outdataset = GDALCreate(driver, outputfp, input->cols, input->rows, 1,
GDT_Float32, papszOptions);
    GDALRasterBandH outband = GDALGetRasterBand(outdataset, 1);
    GDALSetGeoTransform(outdataset, input->geotransform);
    GDALSetProjection(outdataset, input->projection);

    float *datalist = convertFloatArray(input);


    char ret = GDALRasterIO(outband, GF_Write, 0, 0, input->cols, input->rows, datalist,
input->cols, input->rows, GDT_Float32, 0, 0);

    if (ret != 0) {
        printf("Export was not successful.\n");
    }


    GDALSetRasterNoDataValue(outband, input->nodata);

    free(datalist);
    GDALClose(outdataset);
    printf("\n\nSurface exported to file: %s\n\n", outputfp);

    free(outputfp);
}



//
//            - - - Information printers (infoprinters.c) - - -
//


#include "bathymetrictools.h"

/*
*   This file contains:
*   - Printer functions for structured data types
*     to help with development.
*   - These are not strictly necessary (but used in current main, so included also here)
*/


/*
*   Prints info of depth model surface
*/
void printFloatSurfaceInfo(struct FloatSurface *input) {
```

```
    printf("\n\nStruct FloatSurface content:\n");
    printf("Filepath (input): %s\n", input->inputfp);
    printf("\nProjection: %s\n", input->projection);
    printf("\nNodata value: %f\n", input->nodata);
    printf("Rows: %d, Columns: %d\n", input->rows, input->cols);
    printf("Georeferencing information: %f, %f, %f, %f, %f, %f \n", input-
>geotransform[0], input->geotransform[1], input->geotransform[2], input-
>geotransform[3], input->geotransform[4], input->geotransform[5]);
    printf("Test from array[%d][%d]: %f\n\n", input->rows / 2, input->cols / 2, input-
>array[input->rows / 2][input->cols / 2]);
}


/*
*   Prints info of contour limit surface
*/
void printLimitSurfaceInfo(struct LimitSurface *input) {
    printf("\n\nStruct LimitSurface content:\n");
    printf("Filepath (input): %s\n", input->inputfp);
    printf("\nProjection: %s\n", input->projection);
    printf("\nRows: %d, Columns: %d\n", input->rows, input->cols);
    printf("Georeferencing information: [%f, %f, %f, %f, %f, %f]\n", input-
>geotransform[0], input->geotransform[1], input->geotransform[2], input-
>geotransform[3], input->geotransform[4], input->geotransform[5]);
    printf("Test from array[%d][%d]: %d\n\n", input->rows / 2, input->cols / 2, input-
>array[input->rows / 2][input->cols / 2]);
}


/*
*   Prints coin
*/
void printCoin(struct Coin *penny) {
    printf("\nCoin information:\nDiameter: %d (Original radius: %d)\n\n", penny-
>trimmedDiameter, penny->originalRadius);
    for (int row = 0; row < penny->trimmedDiameter; row++) {
        for (int col = 0; col < penny->trimmedDiameter; col++) {
            if (penny->array[row][col] == 1) {
                printf("1");
            }   else {
                printf(" ");
            }
            printf(" ");
        }
        printf("\n");
    }
}



//
//            - - - Makefile (makefile) - - -
//


# Makefile for compiling the sources of the project
# See header file for more information

# Author: Topi Filppula
# ALL RIGHTS RESERVED

SHELL = /bin/sh

# Use GCC compiler, define paths:
CC = gcc
OBJECT_DIR = obj/
BIN_DIR = bin/
```

```
# Flags with debugging helpers:
FLAGS = -O3 -march=native -Wall -Wextra -Wfloat-equal -Werror -pedantic -pedantic-errors
-std=gnu11
LIBS = -lgdal

# Clean:
RM = rm -f
RMDIR = rmdir

# Make new directories:
$(shell mkdir -p $(OBJECT_DIR))
$(shell mkdir -p $(BIN_DIR))

all: surfacetools

surfacetools: main.o limitsurface_manipulation.o rolling_coin_smoothing.o laplaci-
an_smoothing.o inputandmemory.o fileoutput.o infoprinters.o
              $(CC) $(FLAGS) $(LIBS) $(OBJECT_DIR)*.o -o $(BIN_DIR)surfacetools

%.o: %.c
              $(CC) -c $(FLAGS) $< -o $(OBJECT_DIR)$@

clean:
              $(RM) $(OBJECT_DIR)*.o $(BIN_DIR)surfacetools
              $(RMDIR) $(OBJECT_DIR) $(BIN_DIR)
```